BAP TU Delft
ASD detection subgroup Feature Selection

F.Kreté, G.van Wingerden

June 2025

BAP TU Delft ASD detection

subgroup Feature Selection

Author: F.Krete, G.v.Wingerden
Degree Program: BSc Electrical Engineering
Student Number: 5850363, 5603900

Supervisors:
Prof. Geert Leus
Ruben Wijnands

Institution:
Faculty Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Date of Submission:
July 10, 2025

Abstract

In the Netherlands, 3% of people above 4 years old are diagnosed with autism. Diagnosing is currently done with a psychological assessment, but classifying people with autism using resting state functional magnetic imaging, or rs-fMRI, has become promising. The goal of this project was to see if new features could be found, based on the graph of rs-fMRI data stored in the Autism Brain Imaging Data Exchange (ABIDE) dataset, that had a significant positive influence on the accuracy of a predictive model. To do this, several feature selection modules were researched and coded in Python. Subsequently, these were tested with the features and classification methods created by our partner subgroups. The best performing model, using all data, had an accuracy of 74.26% and a sensitivity of 65.35%. The best performing model using graph features, on all data, had an accuracy of 60.4% and a sensitivity of 46.6%. This indicates that there were no graph features developed that had a significant positive influence on classifying whether someone has autism.

Contents

1 Introduction **1**

2 Pre-required knowledge **3**
2.1 ASD in the brain 3
2.2 MRI data 3
2.3 ABIDE dataset 4
2.4 Graph features 5

3 Program of requirements **6**
3.1 Functional requirements 6
3.2 System requirements 6

4 Feature selection methods **7**
4.1 Feature selection 7
4.2 Permutation Importance 8
4.2.1 Methodology 8
4.2.2 Pros and cons 9
4.3 Lasso 9
4.3.1 Methodology 9
4.3.2 HSIC Lasso 10
4.3.3 Pros and cons 11
4.4 Sequential Feature Selection 11
4.4.1 Methodology 11
4.4.2 Pros and cons 12

5 Preprocessing **13**
5.1 mRMR 13
5.2 Clustering 14
5.3 Hyperparameter tuning 14

6 Pipeline **16**
6.1 Loading file and pre-processing 16
6.2 Train and test data 16
6.3 Feature selection 16
6.4 Classification and evaluation 16
6.5 Parallel running 17

7 Results **18**
7.1 Classification methods 18
7.2 Used data 18
7.2.1 Full Correlation 18
7.2.2 Graph Data 19
7.3 Full correlation 20
7.3.1 Performance on multisite data 20
7.3.2 Performance on single-site data 21

8 Graph results **23**
8.1 Dataset 1 23
8.2 Dataset 2 (tuned parameters) 25

9 Discussion **27**
9.1 Full correlation 27
9.1.1 Sex-specific observations 27
9.1.2 Multi- and single site 27
9.1.3 Feature selection behavior across subsets 27
9.2 Graph features 27
9.3 Future work 28

10 Conclusion **29**

A Appendix A **32**
A.1 Full correlation multisite 32
A.1.1 Combined 32
A.1.2 Female data 33
A.1.3 Male data 34
A.2 Full correlation single site 35
A.2.1 Combined data 35
A.2.2 Female data 36
A.2.3 Male data 37

B Appendix B **38**
B.1 Graph multisite 38
B.2 graph NYU 43
B.3 Laplacian NYU 47
B.4 Laplacian multisite 48
B.5 rspect NYU 49
B.6 rspect multisite 50

C Appendix C **52**
C.1 pipeline 52
C.2 Feature selection methods 73

D Appendix D **88**

E Appendix E **89**
E.1 Graph Features 89
E.1.1 Dataset 1 89
E.1.2 Dataset 2 (tuned parameters) 92

1 Introduction

Autism Spectrum Disorder (ASD), often simply denoted as autism, is a neurodevelopmental disorder. It is mainly characterized by challenges in social interactions and communication and by restricted or repetitive behaviors. According to the 'Centraal Bureau voor de Statistiek' (CBS), 3% of Dutch people above 4 years of age indicated having ASD between 2022 and 2024 [1]. This is equivalent to about 280.000 men and 140.000 women in the Netherlands alone. This statistic can be seen in Figure 1. At this moment, ASD is diagnosed through a comprehensive psychological assessment. In the Netherlands, this is done by a psychiatrist, or 'gz-psycholoog'. According to the 'Nederlandse Vereniging voor Autisme' (NVA), there is no biomarker for ASD, and diagnosis relies only on identifying specific behavioral characteristics [2]. The diagnostic process typically involves multiple interviews and observations to assess aspects such as social interaction, communication, and repetitive behaviors. According to the 'Zorgstandaard Autisme', a thorough diagnostic trajectory takes approximately twelve to fourteen hours to complete. It is important to know if someone has ASD, because they have different needs. In addition to this, according to CBS, 74% of people with ASD above 12 years of age had suffered depressive or fearful feelings in the last 4 weeks, compared to 43% of people without ASD.

Research has started to try to find certain biomarkers to classify ASD. The search for biomarkers could speed up diagnosis and make it more reliable. It could also help deepen our understanding of ASD and how it affects the brain. To find these biomarkers, rs-fMRI data is used. The use of machine learning classifiers on rs-fMRI data has been reported to be promising. Some results indicate an overall summary sensitivity and specificity estimates of 73.8% and 74.8%, respectively [3]. With the addition of other brain imaging data or phenotypic data, achieving even higher sensitivities compared to rs-fMRI data alone (84.7% versus 72.8%).

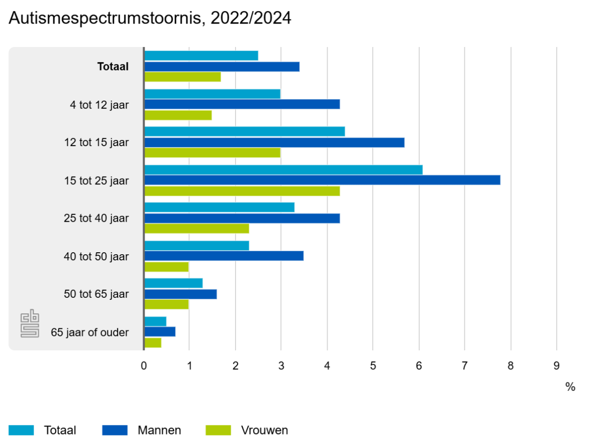

Figure 1: Reported ASD 2022-2024 CBS [1]

Classifying ASD using rs-fMRI data is done by machine learning. Machine learning is the practice of teaching computers to recognize patterns and make predictions on data without being explicitly programmed. It works in 4 steps. The first step is data collection, where samples are collected. Here, the type of data can vary. Second is training, where the data is fed into an algorithm, and this algorithm should try and learn the patterns. It then creates a model based on what it has learned, and finally, this model is used to make predictions on new data. It is important to understand the idea of feature selection. Here, features are evaluated and only the best features are held on to, in an attempt to make the model perform better and faster. The exact process will be explained in the thesis, but is visually illustrated in Figure 4.

It seems that right now, no one has tried to convert the rs-fMRI data to graph features and use these to predict ASD. These graph features could introduce new information on how the brain of someone with ASD differs from that of someone without ASD, called an allistic individual. The goal is to identify new features that have a positive effect on the precision of predictive models when trying to classify ASD. To do this, the second version of the Autism Brain Imaging Data Exchange (ABIDE) [4] was used. All of this is to hopefully promote further research.

To execute this process, the work was divided among 3 subgroups.

1. Feature design
2. Classification
3. Feature selection

Their relationship can be seen in Figure 2.

Feature design will transfer the rs-fMRI data into a graph and extract graph features, along with possible other features. They will then push these features to the other subgroups. Classification will design several classification methods to try to get the best performance with the given features. Finally, feature selection will try to eliminate irrelevant features and identify the features most important for the performance of the process. Next to this, it was decided that there should be a GUI for the project. This task was taken up by the classification subgroup. This thesis will detail the design and subsequent challenges of the feature selection subgroup. It will talk about which feature selection methods have been applied and which features, if any, are the most effective, the design of these features, or the design of the classification methods can be found in their respective thesis.

The thesis will go through the different feature selection methods, how they work, and why they were selected. It will show the results gathered from the feature selection methods and draw a conclusion based on these results.

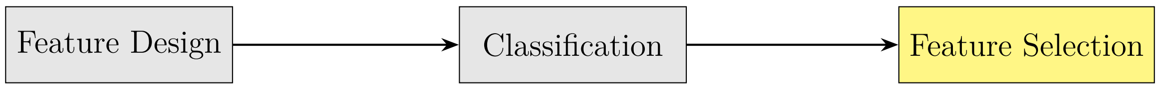

Figure 2: Overview of the subgroups in the project pipeline.

2 Pre-required knowledge

2.1 ASD in the brain

While the NVA says that there are no biomarkers for ASD [2], there is still significant research done on the workings of ASD in the brain. This research has been mostly done using MRI.
Studies show that the brain structure of people with ASD is different than that of allistic people. In the brain of a person with ASD, high connectivity can be measured within local regions of interest, or ROIs, while low long-range connectivity is measured between these different ROIs [5]. ROI pairs with the strongest correlation are shown to be the most abnormal in people with ASD. Especially negatively correlated ROI pairs showed less anti-correlation, possibly representing weaker long-range connections between different regions of interest pairs [6, 7].

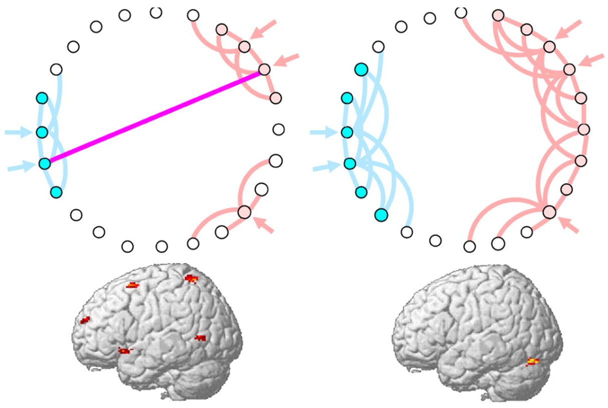

Figure 3: Network connectivity in a neuro-typical brain (left) and a brain with ASD (right) [5]

It is important to know that ASD behaves differently in the brains of females than in males. ASD is more prevalent in males, with around 70% of diagnosed ASD cases being described as males [8]. Males with ASD demonstrate more externalizing behavior, like aggressiveness or hyperactivity, while females with ASD are more likely to experience internalizing problems, like anxiety and other emotional problems [8]. Female children are more likely to camouflage their social challenges, having less intense ASD symptoms, while male children are more likely to play alone, showing clear symptoms of ASD [9, 10]. This shows a clear male bias in our understanding of ASD.

Differences are also found when looking at the brains of males and females with ASD using MRI. For example, research shows that, relative to allistic peers, females had more extensive cortical differences than autistic males [11]. This also shows for the classification process of people with ASD using MRI, where different classifiers have a contrasting performance based on gender [12].

2.2 MRI data

fMRI is an imaging scan that shows activity in specific areas of the brain. A standard MRI scan uses an extremely powerful magnet, radio waves, and computer processing to generate highly detailed 3D pictures of the inside of your body. An fMRI scan uses the same MRI machine, but tracks blood flow in different parts of your brain. This, in combination with the fact that brain cells use more oxygen when utilized for certain tasks, means that the areas of your brain that are working the hardest appear brighter on an fMRI scan [13].

To investigate brain disorders such as ASD, rs-fMRI has been considered because of the minimal need for participants' cooperation, does not rely on cognitive task design, and eliminates the need for additional equipment during imaging. Using rs-fMRI, it is possible to examine resting-state network (RSN) abnormalities in individuals with ASD. The RSNs are a set of brain regions between which there are consistent spatial and temporal fluctuations and provide valuable information about the brain functions in healthy individuals and those with neurological disorders, such as individuals with ASD. Numerous studies have suggested patterns of abnormalities in RSNs as potential biomarkers for the diagnosis of ASD [12].

2.3 ABIDE dataset

All data used in this project is from the ABIDE dataset. This is a collection of rs-fMRI scans from different universities, related to subjects who are either allistic or have ASD. The use of this dataset makes new research easier and cheaper, because there is no need for access to subjects, an MRI machine, or professional staff to operate the machine. Also, since more than 20 international research sites have uploaded their data, the data is extensive and diverse. Extensiveness means the influence of outliers will be minimized, and diversity means the result will apply to a broader population. It contains 1112 rs-fMRI data sets with corresponding structural MRI and phenotypic information from 539 individuals with ASD and 573 age-matched typical controls [14]. While ABIDE is useful and extensive, there are also things to consider when using the dataset, like the fact that it uses data from different research sites. This means that there are also differences in the way the sites did their research. This leads to differences in methods, which can impact research findings.

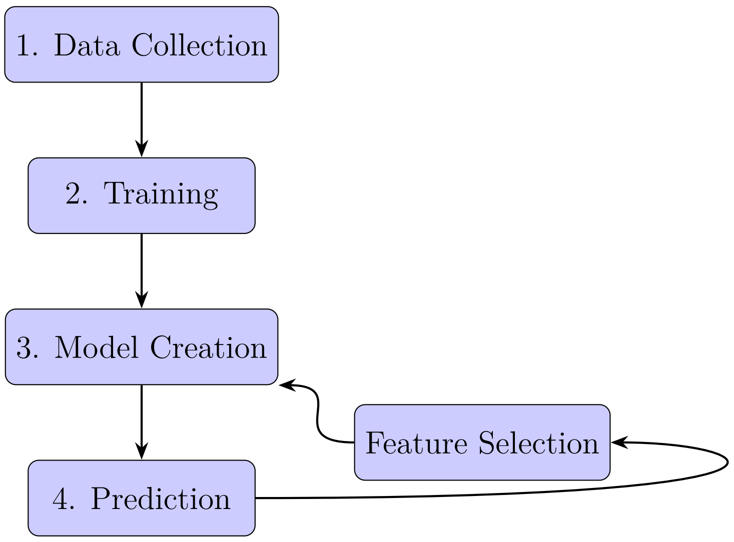

Figure 4: The machine learning pipeline illustrating the main steps and where most of the feature selection takes place.

2.4 Graph features

An important term in this project is 'graph features'. A graph is a collection of nodes that are either connected or not connected to other nodes. A graph and its nodes can represent a lot of things. In a social network, the nodes might represent people, and in a computer network, they might represent computers or routers. In the case of rs-fMRI, the nodes represent ROIs in the brain. The connections between nodes can also have weights or directions, but do not have to. When imaging a collection of nodes and their connections, a graph is created as seen in Figure 5. A graph can also be represented as can be seen in Table 1. Here, the nodes are present on both the x-axis and the y-axis. A connection is represented as a 1 and no connection is represented as a 0. There are several graph-specific features. Examples are connectivity, how many different nodes a node is connected to, centrality, what is the average distance to each node, etc.

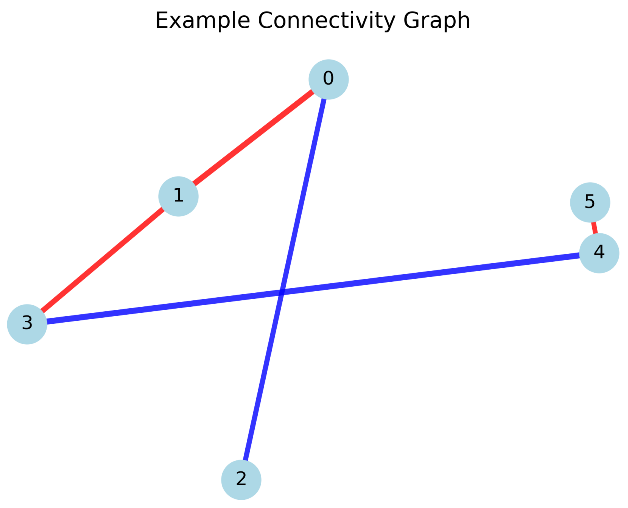

Figure 5: Example connectivity graph showing correlations between regions. Red edges indicate positive correlations, and blue edges indicate negative correlations.

Table 1: Adjacency matrix representing the weighted connections between nodes.

	node 0	node 1	node 2	node 3	node 4	node 5
node 0		0.80	-0.60	0.00	0.00	0.00
node 1	0.80		0.00	0.70	0.00	0.00
node 2	-0.60	0.00		0.00	0.00	0.00
node 3	0.00	0.70	0.00		-0.90	0.00
node 4	0.00	0.00	0.00	-0.90		0.50
node 5	0.00	0.00	0.00	0.00	0.50	

3 Program of requirements

For this project, the feature selection process is quite important. First of all, the feature selection process makes sure only the important features for classification are used in the classification process. This makes the process less computationally expensive and prevents overfitting of the model. This way, accuracy can also be improved by this process.

Apart from that, for the purpose of further research, the features most prominent in the classification process should be shown in the interface. It is the goal of the feature selection group to find and record these features.

It is important to note that, for a performance indicator, reduced misclassifications has been used instead of improved accuracy. This is because improved accuracy can be insensitive to changes if the initial accuracy is already high. Reduced misclassifications focus on the actual errors reduced, making it an informative indicator of feature selection performance.

To make sure the end product is in line with the given goals, a program of requirements was set up. Below the functional requirements, what the system should do, and the system requirements, how the program should perform, are given.

3.1 Functional requirements

- The system must use the ABIDE dataset
- The system works with different classification methods
- The system works with different graph inference methods
- The system shows which features it selects
- The system selects all features that have a positive effect on the performance of the program with the current classification method
- The system shows the accuracy of the model
- The system uses a selection method based on the classification method chosen.

3.2 System requirements

- The system reduces the misclassifications by 25%
- The system runs in less than an hour
- The system is scalable to datasets with 500 features and 1000 subjects

4 Feature selection methods

4.1 Feature selection

An important step in the process of machine learning is the selection of features. By reducing the features in the dataset to only the most important ones, the system will be less computationally intensive. This means that the classification process will take less time to run, which is crucial if this process needs to take a certain amount of time.
Feature selection is also useful because of another problem, called the curse of dimensionality. This is a phenomenon that happens when a machine learning program is trained on extremely high-dimensional data, or a lot of features. At first, the accuracy increases with higher dimensionality, but as dimensionality rises further, the accuracy can go down [15]. This is because the training model begins to overfit. This overfitting happens when a machine learning algorithm has been learning from many particular details from a training set, including noise and outliers. This way, the model performs very well on the training set, but when it runs on the test data, it performs a lot worse. By only training the model on the features deemed important, the problem of overfitting will be reduced, and accuracy goes up, despite using fewer features.

Feature selection can be done in many ways, but the different techniques almost always fall into three categories: filter, wrapped, and embedded methods [16]. Filter methods rank features without the use of a classifier. It usually consists of two steps. The first of which ranks features based on certain criteria. Either in a univariate scheme, independently ranking features, or in a multivariate scheme, which evaluates features in a batch. The second step chooses the highest ranked features [17]. Examples of filter methods include filtering the features using Pearson correlation between the features and the class label, or filtering the features by only keeping the features that cross a certain threshold of variance. Filter methods are usually fast and have a low computational cost, but lack accuracy. They are useful for pre-processing large amounts of data.

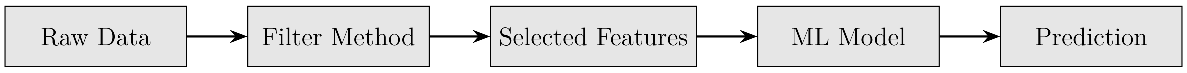

Figure 6: Filter-based feature selection: Independent method applied before model training.

Wrapper methods utilize the performance of the predictor as a way to select features. The predictor is wrapped on a search algorithm that tries to find a subset of features that performs the best. It does this in a couple of steps. It first finds a subset of features, it then evaluates this subset by the performance of the chosen classifier. It repeats steps 1 and 2 until a desired performance is found [18]. This produces a high accuracy but is very computationally expensive. The size of the search space for m features is 2^m [17]. These algorithms are very slow to run and scale exponentially with large amounts of features.
Both these methods have their advantages and disadvantages. Filter methods are computationally efficient, but do not take into account the biases of the classifiers. Wrapper methods generally have a high performance, but have to evaluate the dataset many times using a classifier, which can take a lot of time. The third sort of feature selection method has the advantages of both filter and wrapper methods. They incorporate the classifier into the selection, but are computationally less expensive than wrapper methods [17, 16].

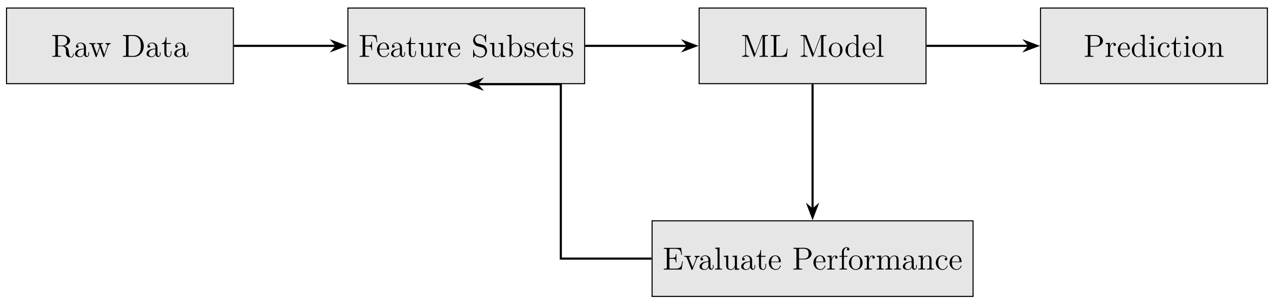

Figure 7: Wrapper-based feature selection: Model is trained on various feature subsets, performance guides selection.

They are called embedded methods. Embedded methods incorporate feature selection as part of the training process. Like wrapper methods, these methods search for an optimal subset of features, but try to limit the computational cost by building the feature selection inside the classifier.

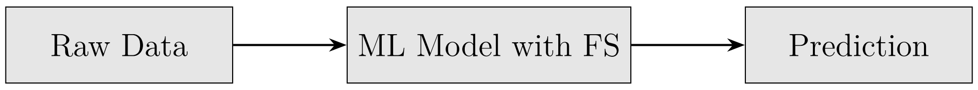

Figure 8: Embedded feature selection: Feature selection happens during model training.

To find the best features in the process of this report, several feature selection methods were developed. Because this process prioritizes performance over computational cost up until it takes longer than an hour to run, wrapper and embedded methods were chosen over filter methods. Also, feature selection methods should be compatible with our data, because the rs-fMRI graph data is very high-dimensional, and the features can be expected to be nonlinear.

Each method has its upsides and downsides, especially in combination with different classification methods. The next section will help explain each feature selection method and why they were chosen. For a quick overview, Table 2 can be referenced.

4.2 Permutation Importance

The first feature selection method evaluated is a wrapped method called Permutation Importance. The method changes every sample of a single feature and evaluates the change in scoring of the classifier model. It repeats this to give each feature an 'importance'. To implement this method, the *scikit learn* library was used in Python [19].

4.2.1 Methodology

Permutation importance is model agnostic, which means that it can work with any classification model. To implement permutation importance, the model needs to run normally first. It creates a baseline metric by scoring the model with all features intact. This scoring method can be in different parameters, but when set to default, it will use the scoring method of the given model. So it will use accuracy for a classifier, in our case, this is always. After this is done, it will randomly assign all samples for a given feature a new value. By randomizing these samples, the connection between the feature and the outcome should be broken. After randomizing, the program checks the scorer again to see how it has changed. It then assigns a feature an importance equal to the amount the score has dropped. This value is thus negative if the score has increased, indicating a negative influence on the accuracy of the classification in our case. The model also repeats this process multiple times per feature. It does this for robustness, by testing multiple times, it reduces the chance of outliers. The equation for finding the final importance score can be found in equation 1

$$i_j = s - \frac{1}{K} \sum_{k=1}^{K} s_{k,j} \tag{1}$$

Here s is the initial score, $\rangle_|$ is the importance score for feature j, and K is the number of repetitions.

4.2.2 Pros and cons

Permutation importance is a very useful method because it works with nonlinear data and with every classifier. There are, however, downsides; some of them make permutation importance less useful for certain classifiers.

First of all, permutation importance is computationally expensive. It needs to check every feature and has to do this repeatedly for a more robust importance score. This means that when your data is high-dimensional, this method becomes slow. It becomes almost impossible to use when the classifier also has a high computational training cost or cost per prediction, like complex neural networks.

The method is not good at handling data that is highly correlated, once again, something that occurs more often in high-dimensional data. Since permutation importance checks each feature one by one, it might mark two highly correlated features as unimportant, since it doesn't affect the score when only one of the features is missing. However, the score can still drop when both features are removed. Thus, permutation importance might miss important features.

Finally, permutation importance does not work well with unstable models or models with high variance, like single decision trees. Since permutation importance randomly shuffles a feature's values, if the score is vastly different each time, then this will lead to inconsistent importance scores.

4.3 Lasso

The next feature selection method is called the Least Absolute Shrinkage and Selection Operator (LASSO). The selection method was implemented using the *pyHSICLasso* package made by Yamada and Climente [20, 21].

4.3.1 Methodology

Lasso is a modification of the Ordinary Least Squares (OLS) cost function. Lasso adds an L1 penalty to OLS, minimizing the absolute sum of the coefficients. This penalty shrinks certain coefficients to zero and effectively performs feature selection [22]. The formula for Lasso can be found in Equation 2.

$$\hat{\boldsymbol{\beta}}^{\text{lasso}} = \arg\min_{\boldsymbol{\beta}} \left\{ \frac{1}{2} \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\} \tag{2}$$

Where $\mathbf{y}$ is the vector of the observed target values, $\mathbf{X}$ is the matrix of input features, β is the vector of regression coefficients to be estimated, and λ is the regularization parameter. The higher this parameter, the more the coefficients shrink to zero. As can be seen, the equation can be divided into two parts: the first of which is the residual sum of squares. The second part represents the L1-penalty function.

By shrinking different feature coefficients to zero, the L1-penalty improves the prediction accuracy and makes the model easily interpretable. It can handle high-dimensionality and high correlation in features by choosing one feature among a group of highly correlated features and shrinking the rest to zero.
Lasso is particularly useful when the number of features is larger than the number of training samples [23]. This is exactly the case with the data of this project, which has a lot of samples. It also has a relatively low computational cost, making it a good feature selection method for our data.

4.3.2 HSIC Lasso

While Lasso is a useful feature selection tool and can handle the high-dimensionality present in our dataset, there is still a critical limitation of Lasso. On its own, Lasso cannot capture non-linear dependency. Because the dataset can be expected to have non-linear features, the Lasso method should be modified to capture non-linearity. A good method that can still handle high-dimensional feature selection uses the Hilbert-Schmidt Independence Criterion (HSIC) [20]. This feature-wise non-linear Lasso, called HSIC Lasso, can be seen in Equation 3.

$$\min_{\beta \in \mathbb{R}^d} \frac{1}{2} \left\| \bar{\mathbf{L}} - \sum_{k=1}^{d} \beta_k \bar{\mathbf{K}}^{(k)} \right\|_{\text{Frob}}^2 + \lambda \|\boldsymbol{\beta}\|_1 \tag{3}$$

As can be seen, the equation looks like Equation 2, with some notable differences. $|| * ||_{Frob}$ is the Frobenius norm, which is the square root of the sum of the squares of all elements in the matrix. $\bar{\mathbf{L}}$ and $\bar{\mathbf{K}}$ are centered Gram Matrices, based on Gram Matrices $K_{i,j} = K(x_{k,i}, x_{k,j})$ and $L_{i,j} = L(y_i, y_j)$ [20]. d here is the total number of features.

According to Equation 4, the first part of Equation 3 can be rewritten as

$$\frac{1}{2} \left\| \bar{\mathbf{L}} - \sum_{k=1}^{d} \beta_k \bar{\mathbf{K}}^{(k)} \right\|_{\text{Frob}}^2 = \frac{1}{2}\text{HSIC}(\mathbf{y}, \mathbf{y}) - \sum_{k=1}^{d} \beta_k \text{HSIC}(u_k, \mathbf{y}) + \frac{1}{2} \sum_{k,l=1}^{d} \beta_k \beta_l \text{HSIC}(u_k, u_l) \tag{4}$$

$HSIC(u_k, y)$, is the Hilbert-Schmidt independence criterion, a kernel-based independence measure [20]. $u_k = [x_{k,1}, ...x_{k,n}]^\intercal \in \mathbb{R}^n$ is the vector of the kth feature for all samples. Here, $HSIC(y, y)$ is a constant and can be ignored. When using a kernel such as the Gaussian kernel, $HSIC$ goes to zero if two variables are statistically independent. This way, HSIC Lasso is functionally a minimum redundancy maximum relevancy, or mRMR, based feature selection method, a feature selection method that will be explained later in the report. Equation 5 is the function that finds the relevancy to the output label, and Equation 6 is the function that finds the redundancy between every feature.

$$\sum_{k=1}^{d} \alpha_k \mathrm{HSIC}(u_k, \mathbf{y}) \tag{5}$$

$$\frac{1}{2}\sum_{k,l=1}^{d} \alpha_k \alpha_l \mathrm{HSIC}(u_k, u_l) \tag{6}$$

The fact that HSIC Lasso is an mRMR-based feature selection method makes it a filter method, not an embedded method like basic Lasso. Even though it uses an L1 penalty to select a sparse subset of features, like Lasso, it selects features before training a classifier, like filter methods do. Besides that, HSIC Lasso is still a very useful feature selection method. It works with non-linear features and, like basic Lasso, it works well with datasets that have a large number of features and a relatively low number of samples. This makes it a good feature selection method for our dataset.

4.3.3 Pros and cons

Lasso is a great method because it performs feature selection and regression simultaneously. It handles high-dimensional data well, which makes it very efficient. The HSIC method should be even better, since it works with non-linear data.

However, Lasso may discard all but one arbitrary feature among correlated features. This can hurt interpretability, making it hard to see which feature influences the classification.

The HSIC lasso has some other specific downsides. It is more complex to implement, but more importantly, it depends on the choice of kernel and the hyperparameters. This means that more parameters can differ and have to be taken into account.

4.4 Sequential Feature Selection

Sequential Feature Selection (SFS) is a wrapped feature selection method used. It creates subsets of features by adding or removing features sequentially based on an estimator. To implement this method, the *scikit learn* library was used in Python.

4.4.1 Methodology

SFS has two opposite versions: forwards SFS and backwards SFS. Forward SFS starts with an empty set. It then runs the program for each feature individually and evaluates the performance based on a specific indicator. The feature that increases the performance the most is then added to this set. After this, the program repeats these steps with every feature not yet added to the set, adding them to the features that have already been chosen and evaluating their performance. The program only stops when a given number of features have been added, the performance stops increasing by a given amount, or all features have been added.

When SFS works backwards, the program starts with a subset containing every feature and removes each feature individually. It then checks which removed feature drops the accuracy the least, or even improves it, and removes that feature from the subset permanently. Like with forwards SFS, the program then repeats these steps with the remaining features. It only stops when a given number of features is left, the performance is not incremented by at least a given amount, while this amount can be negative, or when all features have been removed.

4.4.2 Pros and cons

SFS is an intuitive and powerful method for feature selection because it directly measures the impact of each feature on model performance, in the context of all other features. Besides that, it works with any model. There are, however, downsides.

To start, SFS can be computationally expensive. Both methods first have n evaluations for an amount of chosen features n. The iteration after this becomes $n - 1$ evaluations. This leads to $O(n^2)$ evaluations, which grow exponentially with more features. Still, the backwards variation is quite more expensive than the opposite. This is because, while the forwards variant begins with an empty set, the backward SFS begins its program with every feature. This makes the first evaluations very expensive. Also, depending on the chosen amount of selected features n, forward SFS only has n iterations, while backwards SFS has $m - n$ iterations, where m is the total number of features. This makes backward feature selection very computationally expensive for high-dimensional data.

Performance-wise, the forward model can be lacking. By adding features iteratively, the program can select features that contribute significantly in isolation, but may not be as useful when combined in a set with others. By removing features iteratively, backwards SFS is much better at handling these features in the context of a set, which can lead to a better performing model.
Also, the program is not always good at correlated features. The forwards method might not add features, because on itself they add nothing to the performance of the model, while they do have a positive effect when combined with other features. The backwards method handles correlated features better, since it would see a significant drop in performance when removing one of the two.

Finally, it should be noted that these methods are greedy. When a feature is removed or added, it is not considered again. In forwards SFS, this means that when a feature is added, it will never be removed again, even though in combination with other features, it might not have as much of an effect. In backwards this means that when a feature is removed, it will never be added again, even though when a different feature is removed, it might be relevant again.

5 Preprocessing

Now, all the proposed feature selection methods have been explained, and their pros and cons have been elaborated on. It seems some cons are shared by multiple methods. The biggest problem is the high dimensionality, which often results in computationally expensive feature selection and more highly correlated features. To try and remedy this, the data can be processed, and the features going into the final feature selection method are reduced before trying to select the most influential features. This has been implemented in two ways: a filter method called mRMR and clustering.

5.1 mRMR

mRMR is a very useful method of pre-processing large-scale feature selection problems for more accurate wrapper methods. This method is good for filtering the very large number of features into the few that are most important, using only these for a much more computationally expensive wrapper method. This way, both the advantage of faster filter methods as well as the higher accuracy of a wrapper method are present in the feature selection. mRMR has been implemented using the $scikit\ feature$ package [24].

mRMR is based on Mutual Information for its selection, which can define the dependency of variables. The equation for mutual information is given in Equation 7 [25].

$$I(x;y) = \iint p(x,y) \log \frac{p(x,y)}{p(x)p(y)}\, dx\, dy \tag{7}$$

Here, $p(x)$, $p(y)$ and $p(x,y)$ are probabilistic density functions of x and y. mRMR consists of two different criteria: minimal redundancy and maximal relevance.
First, the algorithm searches for a subset of features S with the maximal relevance to label y. It does this by using the mean value of all mutual information values between an individual feature x_i and label y. This criterion can be seen in Equation 8 [25].

$$\max D(S,y), \quad D = \frac{1}{|S|} \sum_{x_i \in S} I(x_i;y) \tag{8}$$

Equation 8 on its own could already perform feature selection, choosing the set of features most relevant to the label y. However, this method would not consider the dependency of features in the subset of features. When two features have a very high dependency, the subset should not care if one of the features is removed. This is where the minimal redundancy comes into play. The minimal redundancy part of mRMR also uses mutual information, but this is used to evaluate how much the features depend on each other. The minimal redundancy part can be seen in Equation 9 [25].

$$\min R(S), \quad R = \frac{1}{|S|^2} \sum_{x_i, x_j \in S} I(x_i, x_j) \tag{9}$$

In equation 10, it can be seen how both equations come together to form an mRMR score. [25]

$$\max \Phi(D,R), \quad \Phi = D - R \tag{10}$$

In reality, this method works with a greedy approach by starting with an empty set. It first searches for the feature with the most relevance to the label y. After that, it uses a forward selection technique with a sequential search strategy to iteratively find new features with the highest relevancy to the label, while having a low redundancy to the created subset [26].

Although mRMR is a filter, it balances both relevance and redundancy, making sure all selected features are not redundant, while being computationally inexpensive. It is, however, still a filter, which means that it will probably have a lower accuracy than the more intricate wrapper and embedded methods. At least when it is used alone.
Next to this, the strategy is greedy. It works towards a local optimum and not a global optimum. The program iteratively chooses the best feature and the next best feature. It doesn't check each possible combination, meaning it can miss a global optimum.

5.2 Clustering

Another method for pre-processing is clustering. In this method similar features are grouped or 'clustered'. This helps, not only because it removes the total amount of features, but it also groups highly correlated features, meaning that feature selection methods that struggle with highly correlated features should work better with clustered features. An example method for clustering features is provided in the documentation of scikit learn.

In this method, first the Spearman's correlation is used. The Spearman correlation between two variables is equal to the Pearson correlation between the rank values of those two variables; while Pearson's correlation assesses linear relationships, Spearman's correlation assesses monotonic relationships, whether they are linear or not. If two feature ranks are the same, they get a value of +1, and when they are the exact opposites, they get a value of -1. The function used to calculate the Pearson correlation is given in equation 11.

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{11}$$

The program creates a Spearman correlation matrix, where each feature is related to each other. Using hierarchical clustering with Ward's linkage, features are grouped based on their correlation structure. From each cluster, a representative feature is chosen to represent the core data. A threshold value is added to determine how far the features have to be clustered.

To visualize this method of clustering, Figure 9 has been added. In this figure, only 30 features are used and subsequently clustered to make the process clearer. In this figure, the left subplot shows a hierarchical clustering tree, showing what features are clustered at what threshold. The features that are used are shown on the x-axis. The y-axis shows the threshold value. The right graph is a heatmap of the Spearman correlation matrix. Both axes show the used features. If a spot is bright, then these features are highly correlated; if the spot is dark, then these features are not correlated or even anti-correlated.

5.3 Hyperparameter tuning

Some feature selection methods have hyperparameters. These are parameters that are not trained by the model, but set by the user before the program runs. In the context of feature selection, this is mostly how punishing the method is, like the α in the Lasso

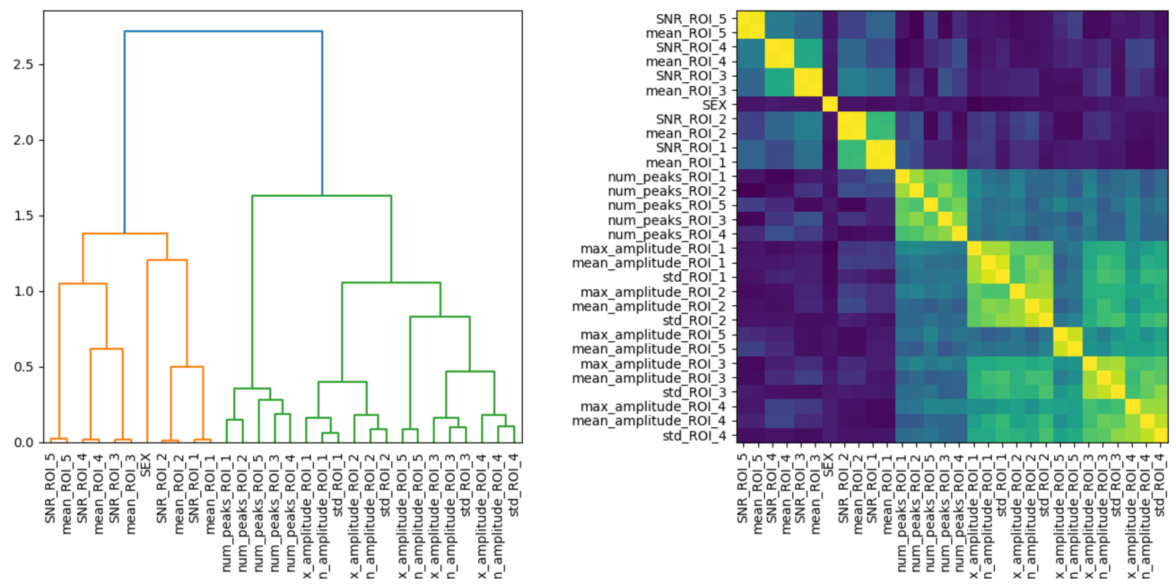

Figure 9: Visualisation of clustering

Table 2: Overview of Feature Selection Methods

Method	**Upsides**	**Downsides**
Permutation Importance (Wrapper)	- Model-agnostic (works with any classifier) - Captures non-linear dependencies - Robust through repeated testing	- Computationally expensive, especially with many features - Struggles with correlated features - Poor performance with unstable models
mRMR (Filter)	- Low computational cost - Balances relevance and redundancy - Useful as pre-selection for wrapper methods	- May have lower final accuracy than wrapper/embedded methods - Greedy strategy may miss the global optimum
Lasso (Embedded)	- Performs feature selection and regression simultaneously - Handles high-dimensional and correlated features - Efficient for large datasets	- Cannot capture non-linear dependencies - May discard all but one among correlated features
HSIC Lasso (Filter)	- Captures non-linear dependencies - Handles high-dimensional data well - Effective with a few samples	- More complex implementation - Depends on choice of kernel and hyperparameters
Sequential Feature Selection (Wrapper)	- Simple and intuitive - Evaluates feature impact in context - Works with any classifier	- Very computationally expensive, especially backward SFS - May miss correlated features - Greedy strategy

method, or how many features the method should select. The best accuracy possible can only be achieved when these hyperparameters are set correctly. Using hyperparameter tuning, this can be done.
Using hyperparameter tuning, the feature selection method won't be used only once, but several times over a certain range of values for a single hyperparameter. Every time, the tuning checks the accuracy and records the highest accuracy of the entire tuning process. The hyperparameter value with the highest accuracy gets used in the final feature selection process.
Lasso has a special function called *alpha_* that selects the best alpha for the dataset. HSIC Lasso needs to be run several times to find the prime alpha.

6 Pipeline

To be able to test and run the program, the classification methods and features of the other subgroups had to be used. For this end, an overarching file was created, called 'pipeline'. This pipeline would combine both the feature design module and the classification module, designed by their respective subgroups, with the feature selection module.

6.1 Loading file and pre-processing

The first part of the pipeline is loading and pre-processing the data supplied by the feature design group. Before loading this data, a choice can be made between the male and female repository. After this choice is made, the selected data will be loaded. Using code from the feature design group, features will be computed and stored in a dataframe per individual. After this, the phenotypic data, which determines if a person has ASD, are loaded and merged into the dataframe.
The data will then be split into X, which contains the features, and y, which contains $DXGroup$, that determines if a person has ASD. After that, the features are preprocessed. Every feature is made to be numeric. Columns with more than 50% NaN values and columns without any variation are discarded. Last, every NaN value is filled with the median value of the column.

6.2 Train and test data

After this, the pipeline can use two methods to train the model using the classifiers made by the classification group. Splitting the data into train and test data early is very important for machine learning, as doing it later might lead to leakage from the test data into the training data. This leads to a skewed accuracy. These options are splitting the data into test and train data with an 80-20 split and using cross-validation. For most feature selection methods, 5-fold cross-validation is used, but for the methods Permutation Importance and backwards Sequential Feature Selection, it is too computationally expensive to run five times. This is why these methods are run using only a train-test split.

6.3 Feature selection

After this, the training data goes through the selected feature selection process. Sometimes this method is combined with a pre-processing method, like a filter or clustering. The feature selection module presents a list of features that the method thinks will give the best result. After the features are selected, the model is run again to see how the set of selected features affects the accuracy and possibly other performance metrics. Depending on the feature selection method, the classification method will already be taken into account. The feature selection part outputs the indices of the selected features, ready for the last classification process.

6.4 Classification and evaluation

As a last step in the process, the selected features will be used in the classification process. This process gets trained on features and labels of the training data. It uses this training to predict new labels for the test data. These predicted labels are then compared to the true labels of the test data.

Performance is evaluated on a couple of factors. The first indicator is the accuracy. This is the fraction of correct predictions, as can be seen in equation 12.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{12}$$

This is a good indicator, but can sometimes be misleading. It doesn't work well when there is a class imbalance, for example. If the sample group is 95% allistic, then classifying everyone as allistic will result in a high accuracy of 95%, even though the model has no true positives. Such a model would not help in finding features that indicate ASD. That is why other indicators are also considered.
The second factor is precision, or how many of the positives were classified correctly 13.

$$Precision = \frac{TP}{TP + FP} \tag{13}$$

After that, the next factor is sensitivity, or how many positives were caught 14. This is important when false negatives are costly, like with classifying people with a potential condition like ASD. This is why we consider sensitivity as an extra important indication.

$$Sensitivity = \frac{TP}{TP + FN} \tag{14}$$

The F1 score combines the precision and sensitivity into a single score 15. This is good for imbalanced datasets.

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{15}$$

Last, AUC, or the Area Under ROC Curve, measures how well the classifier ranks positive vs. negative examples.

After all methods have been evaluated the methods are compared and the best combination of inference method, classifier, and feature selection method is shown alongside the features selected by that method.

6.5 Parallel running

Because the pipeline needs to run the same code five times, each for every classifier, an overarching pipeline has been built using *joblib* that can run this code for each classifier in parallel on different CPU cores. This significantly speeds up the execution time for tasks that can be done independently, such as training the different classifiers using all the feature selection methods. Using this parallelization, the runtime of the program is reduced to the runtime of the slowest classifier.
The developed pipeline code can be found in Appendix C. This does not include the code developed by other subgroups.

7 Results

The goal of this part of the project was to try to find which features were important for classifying someone with ASD and whether or not graph features improved in classification. To do this, all developed feature selection methods were tested with every developed classification method for both the graph features and the full Pearson correlation features.

7.1 Classification methods

Several classification methods have been developed by the classification subgroup. Although their contents are not the main subject of this thesis, it was decided that a summary of their methodology was beneficial. The classification methods used are:

- Support Vector Machine
- Logarithmic Regression
- Random Forrest
- Linear Discriminant Analysis
- K-neighbours classifier

A support vector machine (SVM) is a classification method that uses a hyperplane to separate the data into classes. It tries to find the optimal hyperplane that maximizes the margin between the classes.
Logistic regression (LogR) is a linear model that predicts the probability of a class using a logistic function.

A Random Forest (RandForrest) is a machine learning algorithm that combines multiple decision trees to improve prediction accuracy. It's like a "forest" of trees, where each tree makes its prediction, and the final prediction is an average of all the trees' predictions.

A linear discriminant analysis (LDA) is a classifier with a linear decision boundary, generated by fitting class conditional densities to the data and using Bayes' rule [19].
A K-neighbors classifier (KNN) is a classifier implementing the k-nearest neighbors vote [19].

7.2 Used data

7.2.1 Full Correlation

The non-graph data used as a baseline is called the full-correlation data. Full correlation is calculated by computing the Pearson correlation coefficient between all pairs of features in a dataset. In the context of rs-fMRI, this means computing the Pearson correlation coefficient between every ROI in the brain and using this as a feature. Full correlation has proven to be a good method for classifying ASD using rs-fMRI data [27]. In the full correlation dataset used, 115 ROI's are considered.

7.2.2 Graph Data

The goal of the project is to determine if graph data can be used to help classify people with ASD. This graph data is extracted from the rs-fMRI. The pipeline was run for 4 different inference methods. Each of these methods is a unique way to collect graph features from the rs-fMRI data.

- partial correlation
- mutual info
- normalised Laplacian
- Regularized Logarithmic Spectrum

All of these methods were designed by the feature design subgroup, and their methodology and the choices behind them should be explained there. Although there were more methods developed by the feature selection subgroup, only these methods were working within the pipeline when testing began; thus, it was decided to use them instead of spending more time trying to get the remaining methods to work within the pipeline. The following is the list of graph features used in the feature selection process.

- Closeness Centrality ROI X
- CLustering Coefficient ROI X
- Degree Centrality ROI X
- Eigenvector Centrality ROI X
- Average Clustering
- Diameter
- Spectral Entropy
- Mean Laplacian Eigenvalue
- Max Laplacian Eigenvalue
- Frobenius Norm
- Algebraic Connectivity
- Graph Energy

Next to these graph methods, two separate datasets were developed using Regularized Spectrum and normalized Laplacian. The 20 ROIs were derived from the resting-state network atlas published in [28]. Only the first 10 have gotten names; these can be found in Appendix D.

7.3 Full correlation

In the following sections, an overview of all the results will be provided, both for the full correlation features, based on Pearson's correlation, as well as the graph features. First, a summary of full correlation features results is shown in the following tables; the complete overview can be seen in A. In these results, because of the computational costs of the methods, the permutation importance and the Sequential Features Selection methods were evaluated using a train-test split of $0.80 - 0.20$ respectively. The rest of the methods were evaluated using a stratified 5-K fold cross-validation. Permutation importance is preprocessed with clustered features, using only 56, while forward SFS is tested with mRMR filtered features, running on 70 remaining features. This is because clustering reduces correlated groups of features, which is a struggle for permutation importance. Forward SFS can struggle if it's adding redundant features frequently, which can hurt performance. This is why mRMR is chosen, which gets rid of redundant features.
While backwards SFS was tested on full correlation data, the method proved to be too computationally expensive to be run without the method being fully redundant due to extensive pre-filtering of harsh clustering or mRMR. This is why backward SFS is not included in the results.

In results show the feature selection method with the best accuracy for every classifier. It also shows the percentage of reduced misclassifications using the feature selection method compared to the accuracy of the whole set. This is calculated using Equation 16. Next to that, sensitivity is shown, which is shown to be extra important in classifying people with autism. The equation to calculate the improved sensitivity is seen in Equation 17. The extensive tables covering performances of every feature selection method on every classifier can be found in Appendix A.

$$\%\, Reduced\, Misclassification = \frac{(1 - Acc\, without)\ -\ (1 - Acc)}{(1 - Acc\, without)} \times 100\% \tag{16}$$

$$\% Improved\; Sensitivity = \frac{Sens - Sens\, raw}{Sens\, raw} \times 100\% \tag{17}$$

7.3.1 Performance on multisite data

First, the feature selection methods are considered on the full correlation dataset containing every site available. First, in Table 3, the performance of the combined data is evaluated. After this, the results are split into only male or female data in Tables 4 and 5.
In Table 3, it is clear that Lasso is a dominant feature selection method with the full correlation dataset, having the best accuracy of all feature selection methods with four out of five classifiers. Still, the highest accuracy measured is from forward SFS using Random Forest as a classifier with an accuracy of 0.7426. This is also the highest reduction of misclassifications, with 28.46%. Another high performer is LDA, with a 23.45% reduction of misclassifications and a 54.06% improvement of sensitivity.
Low relative performers are SVM and LogR. These methods have a high initial performance, but have the worst relative improvement using Lasso feature selection. The lowest reduction in misclassifications is achieved by LogR with only a reduction of 2.45%.

Table 3: Best performance per classifier on multisite data

Classifiers	Best	Acc	Acc raw	Sens	Sens raw	Reduced Misclas. (%)	Improved sens (%)
SVM	Lasso	0.6786	0.6549	0.6027	0.5413	6.87	10.19
LogR	Lasso	0.6583	0.6368	0.6272	0.5930	2.45	5.77
RandomForest	fSFS	0.7426	0.6345	0.6535	0.4680	28.46	39.64
LDA	Lasso	0.6583	0.5701	0.6224	0.4040	23.45	54.06
KNN	Lasso	0.6210	0.5475	0.5292	0.4193	16.15	26.21

In Table 4, the female split is evaluated. In the female split, permutation importance is the most prominent feature selection method. The best relative performers in misclassifications are LogR, LDA, and KNN, all using permutation importance, with 8.87%, 12.28%, and 16.22% respectively. With improved sensitivity, the classifiers perform further apart, with the top performers being SVM, which uses forward SFS for a 1400% improvement, and Random Forest, which doubles its sensitivity. The sensitivity of KNN goes down by 40.12%. Still, all these sensitivities are relatively low, around 0.3. Table 5, the male split is evaluated. The best feature selection methods are very diverse, with only Permutation importance being the best twice. The best improvement in the classifier is LDA using forward SFS with 15.63% reduced misclassifications and 86.45% improvement in sensitivity. The lowest performer is LogR using Lasso, with a worse sensitivity and a 1.51% reduced misclassification.

Table 4: Best performance per classifier on multisite female data

Classifiers	Best	Acc	Acc raw	Sens	Sens raw	Reduced Misclas. (%)	Improved sens (%)
SVM	fSFS	0.6428	0.6233	0.3	0.02	5.18	1400
LogR	Permutation	0.6378	0.6026	0.3753	0.36	8.87	4.25
RandomForest	mRMR	0.6381	0.6309	0.2095	0.1	2.01	109.5
LDA	Permutation	0.6429	0.5929	0.5	0.28	12.28	78.57
KNN	Permutation	0.6786	0.6164	0.2	0.3340	16.22	-40.12

Table 5: Best performance per classifier on multisite male data

Classifiers	Best	Acc	Acc raw	Sens	Sens raw	Reduced Misclas. (%)	Improved sens (%)
SVM	fSFS	0.68	0.6435	0.6527	0.5724	10.24	14.03
LogR	Lasso	0.6287	0.6230	0.6174	0.6227	1.51	-0.85
RandomForest	Permutation	0.68	0.6206	0.6667	0.5278	15.63	26.32
LDA	fSFS	0.6333	0.5617	0.625	0.3354	16.34	86.45
KNN	Permutation	0.6067	0.5482	0.625	0.3965	12.95	57.64

7.3.2 Performance on single-site data

To eliminate the effect of the difference between site data on the performance of the feature selection methods, the methods have also been tested on a single site. NYU has been chosen as the single site, because it has the most samples out of all sites. Like the multisite results, these results are also first evaluated using the combined data, after which the results have also been split between male and female samples.

In Table 6, more variety can be seen in the best-performing feature selection method per classifier, while Lasso is still the best-performing feature selection method. The best performing classifiers according to reduced misclassifications are LDA using forward SFS and KNN using Lasso, with 14.17% and 8.81% respectively. The worst relative performers are SVM and logR, with only 1.54% and 3.43%.
The raw sensitivities of the classifiers are very low, with the lowest being of SVM with only 0.2467. All classifiers improve their sensitivity significantly using a feature selection method.

Table 6: Best performance per classifier using the entire NYU data.

Classifiers	Best	Acc	Acc raw	Sens	Sens raw	Reduced Misclas. (%)	Improved Sens (%)
SVM	Lasso	0.6546	0.6492	0.4781	0.2467	1.54	93.84
LogR	mRMR	0.6440	0.6314	0.5733	0.4781	3.43	19.91
RandForest	Lasso	0.6723	0.6482	0.4895	0.3552	6.84	37.78
LDA	fSFS	0.7142	0.6669	0.6	0.5467	14.17	9.75
KNN	Lasso	0.6373	0.6023	0.3819	0.3171	8.81	20.44

Table 7 shows female performance and has the best accuracies of all, but shows why only looking at accuracy can give a skewed impression. Looking at SVM, it has a raw accuracy of 0.7142, while having a sensitivity of 0.0, meaning it got this accuracy by classifying everyone as allistic. This reflects the limited number of female samples available. SVM, LogR, and Random Forest have been able to reduce the misclassifications using Lasso, while improving the sensitivity. LDA and KNN have also improved their sensitivity, but not reduced their misclassifications, with KNN even increasing by 14.33% using HSIC Lasso.
Table 8 shows KNN with forward SFS as the best relative performer, reducing misclassifications by 11.62% and improving sensitivity by 106.19%. The lowest performer based on misclassifications is Random Forest using Lasso with −2.00%, while SVM using mRMR does not increase sensitivity at all.

Table 7: Best performance per classifier on females from the NYU data.

Classifiers	Best	Acc	Acc raw	Sens	Sens raw	Reduced Misclas. (%)	Improved Sens (%)
SVM	Lasso	0.8000	0.7142	0.4	0.0	29.93	–
LogR	Lasso	0.8286	0.7429	0.4	0.2	33,26	100.0
RandomForest	Lasso	0.8286	0.7714	0.5	0.2	25.07	150.0
LDA	Lasso	0.7714	0.7714	0.5	0.3	0.00	66.67
KNN	HSIC Lasso	0.7714	0.8000	0.4	0.1	-14.33	300.0

Table 8: Best performance per classifier using the male NYU data.

Classifiers	Best	Acc	Acc raw	Sens	Sens raw	Reduced Misclas. (%)	Improved Sens (%)
SVM	mRMR	0.6251	0.6177	0.5308	0.5308	1.93	0.0
LogR	mRMR	0.6188	0.5891	0.5782	0.5333	7.24	8.42
RandomForest	Lasso	0.6471	0.6540	0.6538	0.5282	-2.00	23.68
LDA	Permutation	0.6465	0.6397	0.6256	0.6270	1.88	-0.22
KNN	fSFS	0.6429	0.5960	0.7693	0.3731	11.62	106.19

8 Graph results

Each set of features was run through each combination of classifiers and feature selection methods. The complete list of performance metrics will be found in Appendix B. In this section, only the summary tables will be covered.

8.1 Dataset 1

The program was run for both multi-site as well as single-site. For the same reasons as noted above. In Table 9, the most selected features for multi-site and single-site are shown. Their score in this case is the number of times they are selected by a feature selection method. A full list of used graph features can be found in Appendix E. The top 3 in both cases are: Mean Laplacian Eigenvalue, Spectral Entropy, and Graph Energy. Although these might have a significant effect, their results are most likely skewed. Some of the inference methods, when examined more closely, had very little to no difference in all but 6 features, leaving only these features to be considered:

- Average Clustering
- Diameter
- Spectral Entropy
- Mean Laplacian Eigenvalue
- Max Laplacian Eigenvalue
- Frobenius Norm
- Algebraic Connectivity
- Graph Energy

Considering this, it is the remaining features that are of more interest.

- Eigenvector Centrality ROI 2
- Eigenvector Centrality ROI 4
- Clustering Coefficient ROI 3

Eigenvector centrality and clustering coefficient are graph features. Eigenvector centrality can be calculated with equation 18. It represents not just how many connections a node has, but whether those connections are to important nodes.

$$x_i = \lambda_1 \sum_{j \in \text{neighbors}(i)} A_{ij} x_j \tag{18}$$

The clustering coefficient can be calculated with equation 19. This represents how interconnected a node's neighbours are.

$$C_i = \frac{2e_i}{k_i(k_i - 1)} \tag{19}$$

Table 10 and Table 12 show the best-performing combinations. The performance metric here is the F1 score. The main goal was, however, to decrease misclassifications, which means that a higher accuracy is the most important metric. Tables 10 and 12 show the best-performing combination in accuracy. Coincidentally, these are the same methods at the top of our previous tables.

- norm Laplacian, KNN, backwards SFS (for multisite)
- mutual info, LDA, forwards SFS (for single site)

Between these two, the single site has a better reduction in missclassification. It results in $\% \, Reduced \, Misclassification = \frac{(1-0.4934)-(1-0.6765)}{(1-0.4934)} \times 100\% = 36.14\%$. Compared to $\% \, Reduced \, Misclassification = \frac{(1-0.5059)-(1-0.5854)}{(1-0.5059)} \times 100\% = 16.09\%$. Although both of these methods seem promising, it should be noted that the SFS methods were not cross-validated, meaning their results may not be replicated and are thus unreliable.

Table 9: Top selected features for Multisite and NYU datasets

Selected Feature (Multisite)	**Score**
Mean Laplacian Eigenvalue	97
Spectral Entropy	95
Graph Energy	79
Eigenvector Centrality_ROI_2	51
Eigenvector Centrality_ROI_4	47

Selected Feature (NYU)	**Score**
Mean Laplacian Eigenvalue	64
Spectral Entropy	56
Graph Energy	50
Clustering Coefficient_ROI_3	47
Frobenius Norm (Laplacian Spectrum)	45

Table 10: Top multisite classifier performance

Classifier	**Graph Method**	**Feature Selector**	**F1 score**
KNN	norm_laplacian	backward SFS	0.527778
KNN	partial_corr	forwards SFS	0.513158
KNN	partial_corr	backward SFS	0.513158
KNN	partial_corr	mRMR	0.489523
KNN	partial_corr	Raw data	0.489523

Table 11: Multisite detailed classification metrics for the highest accuracy

Metric	**Value**
Classifier	KNN
Graph Method	norm_laplacian
Feature Selector	backward SFS
Number of Features	20.0
Accuracy	0.585366
Precision	0.550725
sensitivity	0.506667
F1 Score	0.527778
AUROC	0.553708
Sensitivity	0.651685

Table 12: Top NYU classifier performance

Classifier	**Graph Method**	**Feature Selector**	**F1 score**
LDA	mutual_info	forwards SFS	0.592593
LDA	norm_laplacian	forwards SFS	0.580645
LDA	mutual_info	backward SFS	0.551724
LogR	norm_laplacian	forwards SFS	0.551724
LDA	norm_laplacian	HSIC_Lasso	0.510297

Table 13: NYU detailed classification metrics for the highest accuracy.

Metric	**Value**
Classifier	LDA
Graph Method	mutual_info
Feature Selector	forwards SFS
Selected Feature	Closeness Centrality_ROI_1
Number of Features	20.0
Accuracy	0.676471
Precision	0.666667
sensitivity	0.533333
F1 Score	0.592593
AUROC	0.698246
Sensitivity	0.789474

8.2 Dataset 2 (tuned parameters)

Next to these methods, there were also two datasets designed separately. These datasets used regularized spectrum and normalised Laplacian inference methods, but had parameters that were more specifically tuned, which should lead to better results. The best overall performing feature selection method for each classifier can be found in Table 14. Next to this, the feature selection methods with the highest accuracy can be found in Table 15. As you can see, the best performing feature selection method and classifier combinations, when setting accuracy as the most important performance metric, are:

- KNN & forwards SFS
- LogR & forwards SFS
- Random forest & forwards SFS
- SVM & Permutation

Only two of these get to 60% accuracy. Which is lower than every result obtained from the full correlation features. The top 20 most influential features can be found in Table 16. These features correspond to the edge weights between ROIs. There is no feature with a significantly higher prevalence than the others.

Table 14: Best overall feature selection method per classifier

Classifier	Feature Selection	Accuracy	Precision	sensitivity	F1_score	Auroc	Sensitivity
KNN	forwards SFS	0.566 ± 0.055	0.517 ± 0.079	0.444 ± 0.098	0.474 ± 0.076	0.551 ± 0.035	0.660 ± 0.095
LDA	forwards SFS	0.580 ± 0.015	0.532 ± 0.023	0.470 ± 0.044	0.498 ± 0.033	0.583 ± 0.014	0.666 ± 0.043
LogR	forwards SFS	0.590 ± 0.025	0.546 ± 0.033	0.473 ± 0.046	0.506 ± 0.036	0.587 ± 0.018	0.682 ± 0.052
RandomForest	forwards SFS	0.604 ± 0.047	0.576 ± 0.075	0.464 ± 0.046	0.511 ± 0.044	0.598 ± 0.051	0.715 ± 0.083
SVM	Permutation	0.604 ± 0.010	0.582 ± 0.015	0.456 ± 0.058	0.500 ± 0.040	0.618 ± 0.014	0.456 ± 0.058

Table 15: Best accuracy feature selection method per classifier

Classifier	Feature Selection	Accuracy	Precision	sensitivity	F1_score	Auroc	Sensitivity
KNN	forwards SFS	0.566 ± 0.055	0.517 ± 0.079	0.444 ± 0.098	0.474 ± 0.076	0.551 ± 0.035	0.660 ± 0.095
LDA	Lasso_selection	0.584 ± 0.048	0.559 ± 0.062	0.409 ± 0.045	0.465 ± 0.048	0.586 ± 0.055	0.409 ± 0.045
LogR	forwards SFS	0.590 ± 0.025	0.546 ± 0.033	0.473 ± 0.046	0.506 ± 0.036	0.587 ± 0.018	0.682 ± 0.052
RandomForest	forwards SFS	0.604 ± 0.047	0.576 ± 0.075	0.464 ± 0.046	0.511 ± 0.044	0.598 ± 0.051	0.715 ± 0.083
SVM	Permutation	0.604 ± 0.010	0.582 ± 0.015	0.456 ± 0.058	0.500 ± 0.040	0.618 ± 0.014	0.456 ± 0.058

Table 16: Top selected features across all methods and both datasets

Feature	Count
A_0_2	41
A_1_2	40
A_11_18	37
A_0_12	36
A_9_13	35
A_8_10	34
A_12_18	33
A_2_17	32
A_3_10	32
A_0_17	31
A_4_13	31
A_1_11	31
A_18_19	31
A_9_17	30
A_11_17	30
A_3_9	30
A_3_7	30
A_4_9	30
A_5_6	29
A_1_5	28

9 Discussion

9.1 Full correlation

9.1.1 Sex-specific observations

The female sample size is far smaller than the male sample size, only having 138 samples in the full dataset and 35 in the single-site dataset. This high dimensionality creates an unstable accuracy and a low sensitivity. Feature selection remedies this problem and improves performance a lot. This phenomenon is less prevalent in the bigger male dataset.

9.1.2 Multi- and single site

The raw performances are similar in terms of accuracy, while the sensitivity is far lower in the single-site data. As can also be seen in the female data, the sensitivity can be improved a lot using feature selection if the sample size is low and the data is high-dimensional. This improvement in sensitivity is not translated into the misclassifications because false positives increased. Examining reduced misclassifications, the multisite data outperforms the single-site data. The number of samples ensures the selection of better features, outweighing the difference in sites.

9.1.3 Feature selection behavior across subsets

Overall, Lasso, permutation importance, and fSFS are the most prevalent feature selection methods. HSIC Lasso is far less prevalent, only being the best method once. This can be explained by HSIC Lasso being a filter method, mostly focused on nonlinear data. The fact that permutation importance and fSFS are the best-performing feature selection methods can be explained by the fact that they are wrapper methods, built for high performance, but can also be explained by them not being cross-validated and having a very fortunate data split.
The classifiers benefiting most from feature selection are LDA, KNN, and Random Forest. SVM and LogR benefit less from feature selection when looking at misclassifications. The sensitivity of SVM can improve a lot when feature selection is used.
Overall, apart from a few exceptions, in all datasets, there is at least one feature selection method that can help classifiers in their performance, especially when looking at sensitivity.

9.2 Graph features

Looking at the graph features, no significantly high accuracies were achieved. Notably, the single-site results were better than the multisite results for the first set of graph features, which was not as clear with the new dataset. This difference in performance may result from variations in measurements and how they are recorded at different universities.
It is notable that in the top multisite performers, a raw data entry is included. This means that the program had a higher F1 score when no feature selection was applied. When looking through further data, it is clear that this is due to a difference in inference method. As this particular inference method creates a relatively high F1 score overall, it is only beaten out once in the top 5.
Which feature selection method works best differs wildly between the datasets used. When looking at specific datasets, Random forest using forwards SFS with the 2nd rspect NYU

dataset does achieve 68.57% accuracy with a sensitivity of 85%. This would outperform the previous graph datasets and compete with the full correlation datasets. However, since this method could not be cross-validated due to computational limits, these results are not trustworthy.

9.3 Future work

For future work, it would be ideal to have a consistent supply of features. The immense difference in results between datasets and the fluctuation in features made it difficult to achieve concrete results, and our dependence on their changing code required us to debug essential elements each time the feature design team introduced new features.
Our reliance on their code meant that we had to wait for them before we could test and properly evaluate our methods. This caused delays in development and restricted our ability to create and test more advanced methods. Along with limiting the time to properly process and evaluate the results.
Finally, it would be beneficial to have additional documentation for each component. This would facilitate a better understanding of the code developed by the various subgroups, as well as the meanings of their outputs. Ultimately, making it easier to utilize each other's work and build upon it.

10 Conclusion

The constructed pipeline works with the ABIDE dataset along with different classification methods and graph inference methods. It shows which features it selects and shows the achieved accuracy. Each method has selected all features that have a positive effect on the performance of the classifier, according to the selection method. The program does not select the best feature selection method for each classifier beforehand, because the best feature selection method changes if the dataset changes. A compromise was made, where the pipeline now shows the best combination of inference method, classifier, and feature selection method and shows the chosen features. In some, but not all cases, the feature selection module can reduce misclassification by 25% or more. In all cases, the program runs in less than an hour with at least 500 features and 1000 subjects, barring the cross-validation of SFS. Several sets of combinations have been used to try and see if graph features have a significant impact on classifying people with ASD. Unfortunately, most of the classifiers using graph features could not even achieve an accuracy of more than 60% without decreasing other performance metrics. In general, the sensitivity is very low, even the highest values do not reach 60%, meaning that there are always more than 40% of people with ASD who are not classified. This seems to indicate that the gathered graph features do not have a significant impact on classifying whether someone has ASD. Especially when compared to the full correlation features, they underperform. These features consistently get accuracies over 60% as well as more consistent sensitivities.

References

[1] Centraal Bureau voor de Statistiek (CBS). *3 Procent van de Bevolking Geeft aan een Autismespectrumstoornis te Hebben.* Geraadpleegd op 27 mei 2025. 2025.

[2] Stichting Autisme Nederland. *Diagnose Autisme.* Geraadpleegd op 27 mei 2025. 2025.

[3] C. P. Santana et al. "rs-fMRI and machine learning for ASD diagnosis: a systematic review and meta-analysis". In: *Scientific Reports* 12 (2022), p. 6030. DOI: 10.1038/s41598-022-09821-6.

[4] Craddock Cameron et al. "The Neuro Bureau Preprocessing Initiative: open sharing of preprocessed neuroimaging data and derivatives". In: *Frontiers in Neuroinformatics* 7 (Jan. 2013). DOI: 10.3389/conf.fninf.2013.09.00041.

[5] Matthew K. Belmonte et al. "Autism and Abnormal Development of Brain Connectivity: Figure 1." In: *Journal of Neuroscience* 24.42 (Oct. 2004), pp. 9228–9231. DOI: 10.1523/jneurosci.3340-04.2004.

[6] Jeffrey S. Anderson et al. "Functional connectivity magnetic resonance imaging classification of autism". In: *Brain* 134.12 (Oct. 2011), pp. 3742–3754. DOI: 10.1093/brain/awr263.

[7] M. A. Just. "Cortical activation and synchronization during sentence comprehension in high-functioning autism: evidence of underconnectivity". In: *Brain* 127.8 (June 2004), pp. 1811–1821. DOI: 10.1093/brain/awh199.

[8] Antonio Napolitano et al. "Sex Differences in Autism Spectrum Disorder: Diagnostic, Neurobiological, and Behavioral Features". In: *Frontiers in Psychiatry* 13 (May 2022). DOI: 10.3389/fpsyt.2022.889636.

[9] Michelle Dean, Robin Harwood, and Connie Kasari. "The art of camouflage: Gender differences in the social behaviors of girls and boys with autism spectrum disorder". In: *Autism* 21.6 (Nov. 2016), pp. 678–689. DOI: 10.1177/1362361316671845.

[10] Meng-Chuan Lai and Peter Szatmari. "Sex and gender impacts on the behavioural presentation and recognition of autism". In: *Current Opinion in Psychiatry* 33.2 (Dec. 2019), pp. 117–123. DOI: 10.1097/yco.0000000000000575.

[11] Derek S Andrews et al. "Sex differences in trajectories of cortical development in autistic children from 2–13 years of age". In: *Molecular Psychiatry* 29.11 (May 2024), pp. 3440–3451. DOI: 10.1038/s41380-024-02592-8.

[12] Hossein Haghighat. "A sex-dependent functional-effective connectivity model for diagnostic classification of Autism Spectrum Disorder using resting-state fMRI". In: *Biomedical Signal Processing and Control* 85 (Mar. 2023), p. 104837. DOI: 10.1016/j.bspc.2023.104837.

[13] Cleveland Clinic Medical Professional. "Functional MRI (FMRI)". In: *Cleveland Clinic* (Mar. 2025). Accessed: 2025-06-03.

[14] Adriana Di Martino et al. "The Autism Brain Imaging Data Exchange: Towards a large-scale evaluation of the intrinsic brain architecture in autism". In: *Molecular Psychiatry* 19.6 (2014), pp. 659–667. DOI: 10.1038/mp.2013.78.

[15] Adolfo Crespo Márquez. *The Curse of Dimensionality.* Springer, Jan. 2022, pp. 67–86. DOI: 10.1007/978-3-030-97660-6\{_}7.

[16] Girish Chandrashekar and Ferat Sahin. “A survey on feature selection methods”. In: *Computers Electrical Engineering* 40.1 (Dec. 2013), pp. 16–28. DOI: 10.1016/j.compeleceng.2013.11.024.

[17] Jiliang Tang, Salem Alelyani, and Huan Liu. “Feature selection for classification: A review”. In: *Data classification: Algorithms and applications* (2014), p. 37.

[18] Ron Kohavi and George H. John. “Wrappers for feature subset selection”. In: *Artificial Intelligence* 97.1-2 (Dec. 1997), pp. 273–324. DOI: 10.1016/s0004-3702(97)00043-x.

[19] Fabian Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[20] Makoto Yamada et al. “High-dimensional feature selection by feature-wise kernelized lasso”. In: *Neural computation* 26.1 (2014), pp. 185–207.

[21] Héctor Climente-González et al. “Block HSIC Lasso: model-free biomarker detection for ultra-high dimensional data”. In: *Bioinformatics* 35.14 (2019), pp. i427–i435.

[22] Ramakrishnan Muthukrishnan and R Rohini. “LASSO: A feature selection technique in predictive modeling for machine learning”. In: *2016 IEEE international conference on advances in computer applications (ICACA)*. Ieee. 2016, pp. 18–20.

[23] Robert Tibshirani. “Regression Shrinkage and Selection Via the Lasso”. In: *Journal of the Royal Statistical Society Series B (Statistical Methodology)* 58.1 (Jan. 1996), pp. 267–288. DOI: 10.1111/j.2517-6161.1996.tb02080.x.

[24] Jundong Li et al. “Feature selection: A data perspective”. In: *ACM Computing Surveys (CSUR)* 50.6 (2018), p. 94.

[25] Hanchuan Peng, Fuhui Long, and Chris Ding. “Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy”. In: *IEEE Transactions on pattern analysis and machine intelligence* 27.8 (2005), pp. 1226–1238.

[26] Mary Walowe Mwadulo. “A review on feature selection methods for classification tasks”. In: *Unknown* (2016). Journal name not provided.

[27] Jac Fredo Agastinose Ronicko et al. “Diagnostic classification of autism using resting-state fMRI data improves with full correlation functional brain connectivity compared to partial correlation”. In: *Journal of Neuroscience Methods* 345 (July 2020), p. 108884. DOI: 10.1016/j.jneumeth.2020.108884.

[28] Stephen M Smith et al. “Correspondence of the brain’s functional architecture during activation and rest”. In: *Proceedings of the National Academy of Sciences* 106.31 (2009), pp. 13040–13045.

A Appendix A — *Results*

A.1 Full correlation multisite

A.1.1 Combined

Table 17: Performance of feature selection methods with full correlation

FS Method	Features	Acc	Precision	Sensitivity	F1	AUC
Perfomance SVM						
Raw data	6671	0.655 ± 0.040	0.652 ± 0.159	0.541 ± 0.048	0.589 ± 0.069	0.715 ± 0.038
LASSO	945	0.679 ± 0.038	0.669 ± 0.047	0.602 ± 0.050	0.633 ± 0.082	0.719 ± 0.052
HSIC LASSO	98	0.640 ± 0.046	0.631 ± 0.088	0.529 ± 0.057	0.579 ± 0.038	0.686 ± 0.057
mRMR	200	0.6018 ± 0.042	0.5870 ± 0.164	0.4681 ± 0.032	0.5192 ± 0.101	0.6474 ± 0.153
Permutation	20	0.582 ± 0.000	0.571 ± 0.000	0.390 ± 0.000	0.464 ± 0.000	0.640 ± 0.000
fSFS	Not feasible					
Perfomance LogR						
Raw data	6671	0.637 ± 0.033	0.612 ± 0.047	0.593 ± 0.058	0.600 ± 0.038	0.689 ± 0.030
LASSO	36	0.658 ± 0.026	0.631 ± 0.032	0.627 ± 0.058	0.628 ± 0.044	0.714 ± 0.043
HSIC LASSO	98	0.643 ± 0.054	0.616 ± 0.048	0.600 ± 0.033	0.608 ± 0.058	0.679 ± 0.035
mRMR	100	0.587 ± 0.093	0.560 ± 0.113	0.505 ± 0.093	0.530 ± 0.075	0.630 ± 0.092
Permutation	20	0.638 ± 0.000	0.629 ± 0.000	0.537 ± 0.000	0.579 ± 0.000	0.673 ± 0.000
fSFS	Not feasible					
Perfomance Random Forest						
Raw data	6671	0.635 ± 0.084	0.646 ± 0.050	0.468 ± 0.050	0.540 ± 0.050	0.688 ± 0.061
LASSO	36	0.682 ± 0.022	0.670 ± 0.055	0.615 ± 0.057	0.641 ± 0.034	0.732 ± 0.030
HSIC LASSO	98	0.610 ± 0.053	0.606 ± 0.121	0.559 ± 0.102	0.508 ± 0.092	0.648 ± 0.062
mRMR	100	0.593 ± 0.035	0.581 ± 0.182	0.591 ± 0.098	0.521 ± 0.100	0.618 ± 0.048
Permutation	20	0.610 ± 0.000	0.607 ± 0.000	0.549 ± 0.000	0.517 ± 0.000	0.639 ± 0.000
fSFS	Not feasible					
Perfomance LDA						
Raw data	6671	0.570 ± 0.024	0.608 ± 0.088	0.404 ± 0.298	0.385 ± 0.249	0.580 ± 0.065
LASSO	36	0.661 ± 0.029	0.635 ± 0.035	0.622 ± 0.048	0.628 ± 0.041	0.713 ± 0.027
HSIC LASSO	98	0.601 ± 0.052	0.571 ± 0.094	0.549 ± 0.072	0.559 ± 0.065	0.648 ± 0.092
mRMR	200	0.613 ± 0.068	0.587 ± 0.037	0.544 ± 0.102	0.564 ± 0.093	0.637 ± 0.050
Permutation	20	0.644 ± 0.000	0.638 ± 0.000	0.537 ± 0.000	0.582 ± 0.000	0.672 ± 0.000
fSFS	Not feasible					
Performance KNN						
Raw data	6671	0.548 ± 0.024	0.514 ± 0.037	0.419 ± 0.050	0.460 ± 0.034	0.555 ± 0.030
LASSO	36	0.618 ± 0.038	0.599 ± 0.047	0.532 ± 0.054	0.562 ± 0.050	0.645 ± 0.032
HSIC LASSO	98	0.583 ± 0.056	0.565 ± 0.068	0.579 ± 0.053	0.519 ± 0.124	0.579 ± 0.054
mRMR	100	0.546 ± 0.032	0.512 ± 0.086	0.539 ± 0.128	0.517 ± 0.119	0.561 ± 0.049
Permutation	20	0.531 ± 0.000	0.507 ± 0.000	0.537 ± 0.000	0.522 ± 0.000	0.583 ± 0.000
fSFS	Not feasible					

A.1.2 Female data

Table 18: Performance of feature selection methods with full correlation on female data.

FS Method	Features	Acc	Precision	Sensitivity	F1	AUC
Performance SVM						
Raw data	6671	0.623 ± 0.014	0.067 ± 0.133	0.020 ± 0.040	0.031 ± 0.062	0.563 ± 0.161
LASSO	10	0.616 ± 0.037	0.402 ± 0.220	0.240 ± 0.150	0.283 ± 0.151	0.501 ± 0.118
HSIC LASSO	26	0.596 ± 0.087	0.390 ± 0.410	0.100 ± 0.110	0.149 ± 0.157	0.520 ± 0.170
mRMR	100	0.623 ± 0.014	0.100 ± 0.200	0.020 ± 0.040	0.033 ± 0.067	0.493 ± 0.159
Permutation	20	0.645 ± 0.000	0.200 ± 0.000	0.020 ± 0.000	0.036 ± 0.000	0.500 ± 0.000
fSFS	20	0.643 ± 0.000	0.500 ± 0.000	0.300 ± 0.000	0.375 ± 0.000	0.522 ± 0.000
Performance LogR						
Raw data	6671	0.603 ± 0.089	0.424 ± 0.183	0.360 ± 0.242	0.372 ± 0.185	0.553 ± 0.123
LASSO	20	0.609 ± 0.031	0.417 ± 0.105	0.380 ± 0.204	0.386 ± 0.162	0.537 ± 0.100
HSIC LASSO	26	0.544 ± 0.047	0.346 ± 0.067	0.280 ± 0.075	0.305 ± 0.064	0.512 ± 0.086
mRMR	100	0.623 ± 0.099	0.507 ± 0.130	0.540 ± 0.120	0.511 ± 0.098	0.585 ± 0.114
Permutation	20	0.638 ± 0.000	0.502 ± 0.000	0.320 ± 0.000	0.375 ± 0.000	0.588 ± 0.000
fSFS	20	0.571 ± 0.000	0.417 ± 0.000	0.500 ± 0.000	0.455 ± 0.000	0.544 ± 0.000
Performance Random Forest						
Raw data	6671	0.631 ± 0.044	0.367 ± 0.371	0.100 ± 0.089	0.150 ± 0.133	0.647 ± 0.108
LASSO	10	0.610 ± 0.052	0.296 ± 0.257	0.240 ± 0.224	0.257 ± 0.226	0.555 ± 0.147
HSIC LASSO	32	0.565 ± 0.037	0.232 ± 0.192	0.140 ± 0.120	0.174 ± 0.146	0.545 ± 0.042
mRMR	100	0.638 ± 0.042	0.450 ± 0.245	0.140 ± 0.102	0.210 ± 0.142	0.560 ± 0.060
Permutation	20	0.607 ± 0.000	0.400 ± 0.000	0.200 ± 0.000	0.267 ± 0.000	0.694 ± 0.000
fSFS	20	0.571 ± 0.000	0.333 ± 0.000	0.200 ± 0.000	0.250 ± 0.000	0.528 ± 0.000
Performance LDA						
Raw data	6671	0.593 ± 0.120	0.348 ± 0.095	0.720 ± 0.306	0.452 ± 0.129	0.475 ± 0.102
LASSO	10	0.609 ± 0.031	0.425 ± 0.100	0.420 ± 0.194	0.413 ± 0.156	0.535 ± 0.090
HSIC LASSO	16	0.544 ± 0.088	0.306 ± 0.176	0.220 ± 0.160	0.248 ± 0.167	0.561 ± 0.132
mRMR	100	0.5654 ± 0.092	0.290 ± 0.093	0.400 ± 0.167	0.333 ± 0.118	0.405 ± 0.086
Permutation	20	0.643 ± 0.000	0.500 ± 0.000	0.500 ± 0.000	0.500 ± 0.000	0.617 ± 0.000
fSFS	20	0.500 ± 0.000	0.357 ± 0.000	0.500 ± 0.000	0.417 ± 0.000	0.517 ± 0.000
Performance KNN						
Raw data	6671	0.616 ± 0.052	0.476 ± 0.098	0.340 ± 0.049	0.392 ± 0.054	0.574 ± 0.070
LASSO	10	0.580 ± 0.063	0.405 ± 0.103	0.280 ± 0.075	0.325 ± 0.077	0.548 ± 0.082
HSIC LASSO	10	0.587 ± 0.039	0.367 ± 0.090	0.240 ± 0.136	0.282 ± 0.118	0.528 ± 0.114
mRMR	100	0.566 ± 0.088	0.328 ± 0.208	0.340 ± 0.314	0.318 ± 0.234	0.550 ± 0.114
Permutation	20	0.679 ± 0.000	0.667 ± 0.000	0.200 ± 0.000	0.308 ± 0.000	0.686 ± 0.000
fSFS	20	0.536 ± 0.000	0.286 ± 0.000	0.200 ± 0.000	0.235 ± 0.000	0.400 ± 0.000

A.1.3 Male data

Table 19: Performance of feature selection methods with full correlation on male data.

FS Method	Features	Acc	Precision	Sensitivity	F1	AUC
Performance SVM						
Raw data	6671	0.643 ± 0.029	0.649 ± 0.046	0.572 ± 0.062	0.605 ± 0.037	0.695 ± 0.018
LASSO	82	0.637 ± 0.020	0.629 ± 0.036	0.606 ± 0.035	0.616 ± 0.005	0.684 ± 0.015
HSIC LASSO	20	0.571 ± 0.027	0.561 ± 0.027	0.469 ± 0.083	0.509 ± 0.057	0.607 ± 0.024
mRMR	100	0.602 ± 0.025	0.596 ± 0.022	0.522 ± 0.062	0.556 ± 0.045	0.653 ± 0.024
Permutation	20	0.633 ± 0.000	0.627 ± 0.000	0.583 ± 0.000	0.604 ± 0.000	0.689 ± 0.000
fSFS	20	0.680 ± 0.000	0.671 ± 0.000	0.653 ± 0.000	0.662 ± 0.000	0.688 ± 0.000
Performance LogR						
Raw data	6671	0.630 ± 0.031	0.613 ± 0.030	0.623 ± 0.048	0.617 ± 0.036	0.679 ± 0.037
LASSO	82	0.629 ± 0.007	0.613 ± 0.015	0.617 ± 0.030	0.614 ± 0.011	0.670 ± 0.029
HSIC LASSO	20	0.613 ± 0.030	0.610 ± 0.040	0.544 ± 0.038	0.574 ± 0.030	0.631 ± 0.038
mRMR	100	0.596 ± 0.046	0.581 ± 0.052	0.567 ± 0.054	0.574 ± 0.053	0.614 ± 0.041
Permutation	20	0.613 ± 0.000	0.595 ± 0.000	0.611 ± 0.000	0.603 ± 0.000	0.698 ± 0.000
fSFS	20	0.620 ± 0.000	0.600 ± 0.000	0.625 ± 0.000	0.612 ± 0.000	0.664 ± 0.000
Performance Random Forest						
Raw data	6671	0.621 ± 0.027	0.626 ± 0.043	0.528 ± 0.044	0.571 ± 0.033	0.672 ± 0.012
LASSO	82	0.668 ± 0.022	0.667 ± 0.035	0.620 ± 0.012	0.642 ± 0.014	0.716 ± 0.012
HSIC LASSO	20	0.598 ± 0.035	0.591 ± 0.039	0.528 ± 0.065	0.556 ± 0.050	0.622 ± 0.037
mRMR	100	0.564 ± 0.049	0.552 ± 0.058	0.469 ± 0.074	0.507 ± 0.066	0.592 ± 0.048
Permutation	20	0.680 ± 0.000	0.667 ± 0.000	0.667 ± 0.000	0.667 ± 0.000	0.699 ± 0.000
fSFS	20	0.667 ± 0.000	0.662 ± 0.000	0.625 ± 0.000	0.643 ± 0.000	0.688 ± 0.000
Performance LDA						
Raw data	6671	0.562 ± 0.033	0.594 ± 0.071	0.335 ± 0.195	0.390 ± 0.156	0.616 ± 0.034
LASSO	82	0.630 ± 0.015	0.618 ± 0.021	0.606 ± 0.027	0.611 ± 0.013	0.671 ± 0.022
HSIC LASSO	20	0.606 ± 0.023	0.601 ± 0.018	0.525 ± 0.079	0.558 ± 0.052	0.629 ± 0.036
mRMR	100	0.606 ± 0.023	0.601 ± 0.018	0.525 ± 0.079	0.558 ± 0.052	0.629 ± 0.036
Permutation	20	0.613 ± 0.000	0.595 ± 0.000	0.611 ± 0.000	0.603 ± 0.000	0.699 ± 0.000
fSFS	20	0.633 ± 0.000	0.616 ± 0.000	0.625 ± 0.000	0.621 ± 0.000	0.662 ± 0.000
Performance KNN						
Raw data	6671	0.548 ± 0.021	0.540 ± 0.032	0.396 ± 0.028	0.457 ± 0.029	0.569 ± 0.021
LASSO	82	0.583 ± 0.038	0.576 ± 0.041	0.492 ± 0.068	0.529 ± 0.053	0.629 ± 0.028
HSIC LASSO	20	0.563 ± 0.029	0.556 ± 0.033	0.436 ± 0.053	0.488 ± 0.044	0.563 ± 0.038
mRMR	100	0.543 ± 0.019	0.529 ± 0.021	0.416 ± 0.068	0.464 ± 0.047	0.559 ± 0.027
Permutation	20	0.607 ± 0.000	0.584 ± 0.000	0.625 ± 0.000	0.604 ± 0.000	0.633 ± 0.000
fSFS	20	0.580 ± 0.000	0.562 ± 0.000	0.569 ± 0.000	0.566 ± 0.000	0.617 ± 0.000

A.2 Full correlation single site

A.2.1 Combined data

Table 20: Performance of feature selection methods with full correlation on NYU data.

FS Method	Features	Acc	Precision	Sensitivity	F1	AUC
Perfomance SVM						
Raw data	6671	0.649 ± 0.040	0.820 ± 0.165	0.247 ± 0.072	0.370 ± 0.088	0.703 ± 0.077
LASSO	29	0.655 ± 0.093	0.626 ± 0.159	0.478 ± 0.115	0.541 ± 0.128	0.714 ± 0.124
HSIC LASSO	17	0.585 ± 0.053	0.547 ± 0.148	0.287 ± 0.045	0.372 ± 0.063	0.628 ± 0.053
mRMR	100	0.638 ± 0.074	0.751 ± 0.216	0.290 ± 0.161	0.382 ± 0.182	0.567 ± 0.140
Permutation	20	0.571 ± 0.000	0.500 ± 0.000	0.467 ± 0.000	0.483 ± 0.000	0.553 ± 0.000
fSFS	20	0.600 ± 0.000	0.556 ± 0.000	0.333 ± 0.000	0.417 ± 0.000	0.573 ± 0.000
Performance LogR						
Raw data	6671	0.631 ± 0.067	0.575 ± 0.100	0.478 ± 0.148	0.516 ± 0.122	0.677 ± 0.118
LASSO	29	0.643 ± 0.119	0.587 ± 0.165	0.490 ± 0.180	0.531 ± 0.175	0.712 ± 0.126
HSIC LASSO	40	0.602 ± 0.081	0.527 ± 0.117	0.435 ± 0.222	0.457 ± 0.170	0.604 ± 0.132
mRMR	100	0.644 ± 0.093	0.600 ± 0.117	0.573 ± 0.099	0.580 ± 0.095	0.680 ± 0.080
Permutation	30	0.567 ± 0.117	0.499 ± 0.153	0.426 ± 0.118	0.458 ± 0.131	0.562 ± 0.121
fSFS	20	0.629 ± 0.000	0.571 ± 0.000	0.533 ± 0.000	0.552 ± 0.000	0.597 ± 0.000
Performance Random Forest						
Raw data	6671	0.648 ± 0.081	0.639 ± 0.112	0.355 ± 0.164	0.449 ± 0.156	0.668 ± 0.109
LASSO	29	0.672 ± 0.105	0.637 ± 0.158	0.490 ± 0.179	0.549 ± 0.170	0.697 ± 0.127
HSIC LASSO	31	0.614 ± 0.059	0.591 ± 0.150	0.395 ± 0.103	0.462 ± 0.091	0.606 ± 0.053
mRMR	100	0.614 ± 0.030	0.662 ± 0.174	0.288 ± 0.115	0.374 ± 0.101	0.590 ± 0.065
Permutation	20	0.657 ± 0.000	0.600 ± 0.000	0.600 ± 0.000	0.600 ± 0.000	0.533 ± 0.000
fSFS	20	0.543 ± 0.000	0.462 ± 0.000	0.400 ± 0.000	0.429 ± 0.000	0.562 ± 0.000
Performance LDA						
Raw data	6671	0.667 ± 0.037	0.636 ± 0.070	0.547 ± 0.087	0.581 ± 0.052	0.688 ± 0.040
LASSO	29	0.666 ± 0.077	0.638 ± 0.117	0.519 ± 0.120	0.567 ± 0.108	0.703 ± 0.113
HSIC LASSO	16	0.550 ± 0.056	0.468 ± 0.083	0.383 ± 0.065	0.421 ± 0.072	0.562 ± 0.100
mRMR	100	0.591 ± 0.087	0.535 ± 0.087	0.549 ± 0.088	0.536 ± 0.066	0.587 ± 0.067
Permutation	20	0.600 ± 0.000	0.538 ± 0.000	0.467 ± 0.000	0.500 ± 0.000	0.563 ± 0.000
fSFS	20	0.714 ± 0.000	0.692 ± 0.000	0.600 ± 0.000	0.643 ± 0.000	0.633 ± 0.000
Performance KNN						
Raw data	6671	0.602 ± 0.068	0.564 ± 0.128	0.317 ± 0.148	0.391 ± 0.132	0.600 ± 0.097
LASSO	29	0.637 ± 0.072	0.619 ± 0.144	0.382 ± 0.103	0.470 ± 0.118	0.681 ± 0.086
HSIC LASSO	26	0.568 ± 0.055	0.509 ± 0.070	0.423 ± 0.068	0.454 ± 0.030	0.590 ± 0.054
mRMR	100	0.573 ± 0.037	0.494 ± 0.095	0.290 ± 0.139	0.348 ± 0.123	0.562 ± 0.059
Permutation	20	0.543 ± 0.000	0.400 ± 0.000	0.133 ± 0.000	0.200 ± 0.000	0.450 ± 0.000
fSFS	20	0.486 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.625 ± 0.000

A.2.2 Female data

Table 21: Performance of feature selection methods with full correlation on female NYU data.

FS Method	Features	Acc	Precision	Sensitivity	F1	AUC
Performance SVM						
Raw data	6671	0.714 ± 0.090	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.387 ± 0.380
LASSO	19	0.800 ± 0.194	0.600 ± 0.490	0.400 ± 0.374	0.467 ± 0.400	0.667 ± 0.286
HSIC LASSO	42	0.800 ± 0.146	0.500 ± 0.447	0.400 ± 0.374	0.433 ± 0.389	0.727 ± 0.176
mRMR	100	0.714 ± 0.090	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.560 ± 0.463
Permutation	20	0.743 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.340 ± 0.000
fSFS	20	0.714 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.400 ± 0.000
Performance LogR						
Raw data	6671	0.743 ± 0.167	0.400 ± 0.490	0.200 ± 0.245	0.267 ± 0.327	0.807 ± 0.210
LASSO	20	0.829 ± 0.140	0.600 ± 0.490	0.400 ± 0.374	0.467 ± 0.400	0.760 ± 0.224
HSIC LASSO	16	0.657 ± 0.114	0.380 ± 0.371	0.400 ± 0.374	0.348 ± 0.289	0.647 ± 0.265
mRMR	100	0.743 ± 0.107	0.200 ± 0.400	0.100 ± 0.200	0.133 ± 0.267	0.700 ± 0.261
Permutation	20	0.657 ± 0.000	0.167 ± 0.000	0.300 ± 0.000	0.213 ± 0.000	0.560 ± 0.000
fSFS	20	0.714 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.600 ± 0.000
Performance Random Forest						
Raw data	6671	0.771 ± 0.114	0.400 ± 0.490	0.200 ± 0.245	0.267 ± 0.327	0.807 ± 0.219
LASSO	19	0.829 ± 0.140	0.533 ± 0.452	0.500 ± 0.447	0.493 ± 0.417	0.707 ± 0.266
HSIC LASSO	18	0.686 ± 0.107	0.300 ± 0.400	0.200 ± 0.245	0.233 ± 0.291	0.527 ± 0.164
mRMR	100	0.800 ± 0.146	0.400 ± 0.490	0.300 ± 0.400	0.333 ± 0.422	0.680 ± 0.299
Permutation	20	0.771 ± 0.146	0.400 ± 0.490	0.300 ± 0.400	0.333 ± 0.422	0.470 ± 0.328
fSFS	20	0.714 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.800 ± 0.000
Performance LDA						
Raw data	6671	0.771 ± 0.194	0.400 ± 0.490	0.300 ± 0.400	0.333 ± 0.422	0.793 ± 0.231
LASSO	19	0.771 ± 0.114	0.600 ± 0.389	0.500 ± 0.316	0.507 ± 0.285	0.777 ± 0.282
HSIC LASSO	26	0.686 ± 0.107	0.367 ± 0.371	0.400 ± 0.374	0.347 ± 0.299	0.560 ± 0.361
mRMR	100	0.743 ± 0.107	0.200 ± 0.400	0.100 ± 0.200	0.133 ± 0.267	0.747 ± 0.165
Permutation	20	0.514 ± 0.000	0.200 ± 0.000	0.300 ± 0.000	0.240 ± 0.000	0.467 ± 0.000
fSFS	20	0.571 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.400 ± 0.000
Performance KNN						
Raw data	6671	0.743 ± 0.107	0.200 ± 0.400	0.100 ± 0.200	0.133 ± 0.267	0.520 ± 0.282
LASSO	19	0.771 ± 0.114	0.500 ± 0.447	0.400 ± 0.374	0.400 ± 0.327	0.693 ± 0.291
HSIC LASSO	10	0.800 ± 0.146	0.500 ± 0.447	0.400 ± 0.374	0.433 ± 0.389	0.777 ± 0.146
mRMR	100	0.686 ± 0.140	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.613 ± 0.202
Permutation	20	0.714 ± 0.000	0.300 ± 0.000	0.300 ± 0.000	0.300 ± 0.000	0.430 ± 0.000
fSFS	20	0.714 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.400 ± 0.000

A.2.3 Male data

Table 22: Performance of feature selection methods with full correlation on male NYU data.

FS Method	Features	Acc	Precision	Sensitivity	F1	AUC
Performance SVM						
Raw data	6671	0.618 ± 0.106	0.603 ± 0.133	0.531 ± 0.130	0.564 ± 0.130	0.686 ± 0.070
LASSO	10	0.581 ± 0.076	0.563 ± 0.085	0.531 ± 0.120	0.541 ± 0.083	0.650 ± 0.091
HSIC LASSO	23	0.611 ± 0.041	0.595 ± 0.042	0.517 ± 0.108	0.550 ± 0.080	0.687 ± 0.055
mRMR	100	0.625 ± 0.025	0.617 ± 0.042	0.531 ± 0.051	0.570 ± 0.045	0.672 ± 0.054
Permutation	30	0.522 ± 0.080	0.475 ± 0.117	0.392 ± 0.143	0.426 ± 0.134	0.441 ± 0.078
fSFS	20	0.571 ± 0.000	0.571 ± 0.000	0.308 ± 0.000	0.400 ± 0.000	0.585 ± 0.000
Performance LogR						
Raw data	6671	0.589 ± 0.072	0.570 ± 0.076	0.533 ± 0.102	0.548 ± 0.082	0.627 ± 0.098
LASSO	10	0.597 ± 0.107	0.570 ± 0.107	0.592 ± 0.118	0.579 ± 0.109	0.642 ± 0.128
HSIC LASSO	25	0.566 ± 0.073	0.536 ± 0.059	0.578 ± 0.124	0.552 ± 0.083	0.599 ± 0.058
mRMR	100	0.619 ± 0.094	0.577 ± 0.136	0.550 ± 0.211	0.555 ± 0.180	0.595 ± 0.150
Permutation	30	0.529 ± 0.065	0.505 ± 0.080	0.486 ± 0.146	0.483 ± 0.098	0.529 ± 0.092
fSFS	20	0.536 ± 0.000	0.500 ± 0.000	0.538 ± 0.000	0.519 ± 0.000	0.538 ± 0.000
Performance Random Forest						
Raw data	6671	0.654 ± 0.058	0.657 ± 0.081	0.528 ± 0.124	0.583 ± 0.104	0.705 ± 0.042
LASSO	10	0.647 ± 0.037	0.617 ± 0.034	0.654 ± 0.133	0.629 ± 0.070	0.639 ± 0.074
HSIC LASSO	44	0.581 ± 0.126	0.545 ± 0.115	0.546 ± 0.204	0.538 ± 0.155	0.640 ± 0.139
mRMR	100	0.626 ± 0.058	0.642 ± 0.082	0.451 ± 0.167	0.517 ± 0.116	0.669 ± 0.075
Permutation	30	0.558 ± 0.074	0.534 ± 0.103	0.467 ± 0.102	0.496 ± 0.096	0.571 ± 0.108
fSFS	20	0.536 ± 0.000	0.500 ± 0.000	0.462 ± 0.000	0.480 ± 0.000	0.500 ± 0.000
Performance LDA						
Raw data	6671	0.640 ± 0.079	0.613 ± 0.072	0.627 ± 0.119	0.618 ± 0.091	0.681 ± 0.109
LASSO	10	0.626 ± 0.105	0.606 ± 0.112	0.608 ± 0.104	0.605 ± 0.104	0.635 ± 0.124
HSIC LASSO	30	0.574 ± 0.062	0.547 ± 0.078	0.483 ± 0.119	0.511 ± 0.097	0.614 ± 0.088
mRMR	100	0.412 ± 0.075	0.392 ± 0.063	0.456 ± 0.129	0.417 ± 0.082	0.496 ± 0.130
Permutation	30	0.647 ± 0.081	0.647 ± 0.108	0.626 ± 0.099	0.625 ± 0.070	0.645 ± 0.109
fSFS	20	0.536 ± 0.000	0.500 ± 0.000	0.538 ± 0.000	0.519 ± 0.000	0.544 ± 0.000
Performance KNN						
Raw data	6671	0.596 ± 0.058	0.607 ± 0.092	0.373 ± 0.137	0.453 ± 0.124	0.596 ± 0.030
LASSO	10	0.640 ± 0.078	0.641 ± 0.126	0.546 ± 0.173	0.577 ± 0.120	0.656 ± 0.069
HSIC LASSO	48	0.604 ± 0.066	0.609 ± 0.123	0.405 ± 0.109	0.485 ± 0.117	0.639 ± 0.103
mRMR	100	0.470 ± 0.091	0.437 ± 0.116	0.373 ± 0.097	0.398 ± 0.091	0.530 ± 0.102
Permutation	30	0.588 ± 0.085	0.583 ± 0.118	0.453 ± 0.130	0.502 ± 0.117	0.581 ± 0.099
fSFS	20	0.357 ± 0.000	0.273 ± 0.000	0.231 ± 0.000	0.250 ± 0.000	0.408 ± 0.000

B Appendix B — *Results*

B.1 Graph multisite

Table 23: Performance summary for classifier: KNN

Inference	Feature Selection	Accuracy	Precision	Sensitivity	F1 Score	AUROC
mutual_info	backward SFS	0.549 ± 0.000	0.508 ± 0.000	0.440 ± 0.000	0.471 ± 0.000	0.550 ± 0.000
mutual_info	forwards SFS	0.506 ± 0.000	0.456 ± 0.000	0.413 ± 0.000	0.434 ± 0.000	0.535 ± 0.000
mutual_info	HSIC_Lasso	0.533 ± 0.078	0.490 ± 0.085	0.463 ± 0.094	0.475 ± 0.087	0.540 ± 0.088
mutual_info	Lasso_selection	0.509 ± 0.031	0.459 ± 0.040	0.388 ± 0.046	0.419 ± 0.037	0.511 ± 0.054
mutual_info	mRMR	0.507 ± 0.026	0.462 ± 0.030	0.444 ± 0.028	0.452 ± 0.027	0.507 ± 0.028
mutual_info	Permutation	0.512 ± 0.034	0.466 ± 0.041	0.441 ± 0.058	0.452 ± 0.046	0.519 ± 0.030
mutual_info	Raw data	0.529 ± 0.022	0.484 ± 0.029	0.447 ± 0.064	0.463 ± 0.045	0.533 ± 0.025
norm_Laplacian	backward SFS	0.585 ± 0.000	0.551 ± 0.000	0.507 ± 0.000	0.528 ± 0.000	0.554 ± 0.000
norm_Laplacian	forwards SFS	0.488 ± 0.000	0.435 ± 0.000	0.400 ± 0.000	0.417 ± 0.000	0.489 ± 0.000
norm_Laplacian	HSIC_Lasso	0.510 ± 0.048	0.463 ± 0.057	0.398 ± 0.056	0.426 ± 0.050	0.498 ± 0.042
norm_Laplacian	Lasso_selection	0.496 ± 0.047	0.448 ± 0.051	0.428 ± 0.057	0.437 ± 0.054	0.475 ± 0.052
norm_Laplacian	mRMR	0.497 ± 0.065	0.450 ± 0.072	0.420 ± 0.065	0.434 ± 0.068	0.494 ± 0.075
norm_Laplacian	Permutation	0.507 ± 0.033	0.454 ± 0.042	0.372 ± 0.039	0.409 ± 0.039	0.486 ± 0.028
norm_Laplacian	Raw data	0.527 ± 0.051	0.479 ± 0.067	0.353 ± 0.057	0.406 ± 0.061	0.510 ± 0.070
partial_corr	backward SFS	0.549 ± 0.000	0.506 ± 0.000	0.520 ± 0.000	0.513 ± 0.000	0.508 ± 0.000
partial_corr	forwards SFS	0.549 ± 0.000	0.506 ± 0.000	0.520 ± 0.000	0.513 ± 0.000	0.508 ± 0.000
partial_corr	HSIC_Lasso	0.522 ± 0.014	0.479 ± 0.016	0.503 ± 0.055	0.490 ± 0.032	0.510 ± 0.025
partial_corr	Lasso_selection	0.522 ± 0.014	0.479 ± 0.016	0.503 ± 0.055	0.490 ± 0.032	0.510 ± 0.025
partial_corr	mRMR	0.522 ± 0.014	0.479 ± 0.016	0.503 ± 0.055	0.490 ± 0.032	0.510 ± 0.025
partial_corr	Permutation	0.522 ± 0.014	0.479 ± 0.016	0.503 ± 0.055	0.490 ± 0.032	0.510 ± 0.025
partial_corr	Raw data	0.522 ± 0.014	0.479 ± 0.016	0.503 ± 0.055	0.490 ± 0.032	0.510 ± 0.025
rlogspect	backward SFS	0.470 ± 0.000	0.414 ± 0.000	0.387 ± 0.000	0.400 ± 0.000	0.481 ± 0.000
rlogspect	forwards SFS	0.470 ± 0.000	0.414 ± 0.000	0.387 ± 0.000	0.400 ± 0.000	0.481 ± 0.000
rlogspect	HSIC_Lasso	0.489 ± 0.031	0.435 ± 0.033	0.380 ± 0.029	0.405 ± 0.028	0.461 ± 0.035
rlogspect	Lasso_selection	0.489 ± 0.031	0.435 ± 0.033	0.380 ± 0.029	0.405 ± 0.028	0.461 ± 0.035
rlogspect	mRMR	0.489 ± 0.031	0.435 ± 0.033	0.380 ± 0.029	0.405 ± 0.028	0.461 ± 0.035
rlogspect	Permutation	0.489 ± 0.031	0.435 ± 0.033	0.380 ± 0.029	0.405 ± 0.028	0.461 ± 0.035
rlogspect	Raw data	0.489 ± 0.031	0.435 ± 0.033	0.380 ± 0.029	0.405 ± 0.028	0.461 ± 0.035

Table 24: Performance summary for classifier: LDA

Inference	Feature Selection	Accuracy	Precision	Sensitivity	F1 Score	AUROC
mutual_info	backward SFS	0.494 ± 0.000	0.423 ± 0.000	0.293 ± 0.000	0.346 ± 0.000	0.508 ± 0.000
mutual_info	forwards SFS	0.524 ± 0.000	0.471 ± 0.000	0.320 ± 0.000	0.381 ± 0.000	0.526 ± 0.000
mutual_info	HSIC_Lasso	0.521 ± 0.019	0.470 ± 0.027	0.361 ± 0.055	0.406 ± 0.043	0.516 ± 0.023
mutual_info	Lasso_selection	0.543 ± 0.028	0.498 ± 0.052	0.313 ± 0.064	0.383 ± 0.064	0.535 ± 0.021
mutual_info	mRMR	0.517 ± 0.025	0.458 ± 0.041	0.302 ± 0.055	0.363 ± 0.050	0.520 ± 0.030
mutual_info	Permutation	0.549 ± 0.031	0.507 ± 0.038	0.462 ± 0.056	0.483 ± 0.047	0.574 ± 0.042
mutual_info	Raw data	0.549 ± 0.034	0.508 ± 0.042	0.468 ± 0.055	0.486 ± 0.049	0.575 ± 0.037
norm_Laplacian	backward SFS	0.561 ± 0.000	0.531 ± 0.000	0.347 ± 0.000	0.419 ± 0.000	0.571 ± 0.000
norm_Laplacian	forwards SFS	0.579 ± 0.000	0.558 ± 0.000	0.387 ± 0.000	0.457 ± 0.000	0.575 ± 0.000
norm_Laplacian	HSIC_Lasso	0.547 ± 0.039	0.508 ± 0.065	0.350 ± 0.044	0.414 ± 0.052	0.537 ± 0.047
norm_Laplacian	Lasso_selection	0.498 ± 0.038	0.406 ± 0.086	0.190 ± 0.044	0.256 ± 0.052	0.478 ± 0.021
norm_Laplacian	mRMR	0.516 ± 0.022	0.446 ± 0.043	0.201 ± 0.033	0.274 ± 0.028	0.489 ± 0.029
norm_Laplacian	Permutation	0.553 ± 0.023	0.513 ± 0.031	0.420 ± 0.056	0.461 ± 0.046	0.550 ± 0.036
norm_Laplacian	Raw data	0.549 ± 0.035	0.509 ± 0.047	0.441 ± 0.051	0.472 ± 0.047	0.550 ± 0.037
partial_corr	backward SFS	0.537 ± 0.000	0.471 ± 0.000	0.107 ± 0.000	0.174 ± 0.000	0.567 ± 0.000
partial_corr	forwards SFS	0.537 ± 0.000	0.471 ± 0.000	0.107 ± 0.000	0.174 ± 0.000	0.567 ± 0.000
partial_corr	HSIC_Lasso	0.538 ± 0.028	0.503 ± 0.070	0.187 ± 0.061	0.265 ± 0.057	0.527 ± 0.047
partial_corr	Lasso_selection	0.538 ± 0.028	0.503 ± 0.070	0.187 ± 0.061	0.265 ± 0.057	0.527 ± 0.047
partial_corr	mRMR	0.538 ± 0.028	0.503 ± 0.070	0.187 ± 0.061	0.265 ± 0.057	0.527 ± 0.047
partial_corr	Permutation	0.538 ± 0.028	0.503 ± 0.070	0.187 ± 0.061	0.265 ± 0.057	0.527 ± 0.047
partial_corr	Raw data	0.538 ± 0.028	0.503 ± 0.070	0.187 ± 0.061	0.265 ± 0.057	0.527 ± 0.047
rlogspect	backward SFS	0.506 ± 0.000	0.250 ± 0.000	0.040 ± 0.000	0.069 ± 0.000	0.485 ± 0.000
rlogspect	forwards SFS	0.506 ± 0.000	0.250 ± 0.000	0.040 ± 0.000	0.069 ± 0.000	0.485 ± 0.000
rlogspect	HSIC_Lasso	0.518 ± 0.022	0.140 ± 0.173	0.045 ± 0.056	0.068 ± 0.084	0.458 ± 0.037
rlogspect	Lasso_selection	0.518 ± 0.022	0.140 ± 0.173	0.045 ± 0.056	0.068 ± 0.084	0.458 ± 0.037
rlogspect	mRMR	0.518 ± 0.022	0.140 ± 0.173	0.045 ± 0.056	0.068 ± 0.084	0.458 ± 0.037
rlogspect	Permutation	0.518 ± 0.022	0.140 ± 0.173	0.045 ± 0.056	0.068 ± 0.084	0.458 ± 0.037
rlogspect	Raw data	0.518 ± 0.022	0.140 ± 0.173	0.045 ± 0.056	0.068 ± 0.084	0.458 ± 0.037

Table 25: Performance summary for classifier: LogR

Inference	Feature Selection	Accuracy	Precision	Sensitivity	F1 Score	AUROC
mutual_info	backward SFS	0.488 ± 0.000	0.418 ± 0.000	0.307 ± 0.000	0.354 ± 0.000	0.515 ± 0.000
mutual_info	forwards SFS	0.530 ± 0.000	0.479 ± 0.000	0.307 ± 0.000	0.374 ± 0.000	0.502 ± 0.000
mutual_info	HSIC_Lasso	0.517 ± 0.021	0.460 ± 0.036	0.339 ± 0.067	0.389 ± 0.057	0.523 ± 0.034
mutual_info	Lasso_selection	0.532 ± 0.012	0.476 ± 0.029	0.254 ± 0.056	0.329 ± 0.055	0.528 ± 0.012
mutual_info	mRMR	0.534 ± 0.014	0.485 ± 0.025	0.273 ± 0.050	0.347 ± 0.044	0.498 ± 0.033
mutual_info	Permutation	0.543 ± 0.031	0.499 ± 0.043	0.406 ± 0.066	0.447 ± 0.056	0.563 ± 0.035
mutual_info	Raw data	0.534 ± 0.042	0.491 ± 0.055	0.404 ± 0.048	0.443 ± 0.049	0.554 ± 0.043
norm_Laplacian	backward SFS	0.573 ± 0.000	0.544 ± 0.000	0.413 ± 0.000	0.470 ± 0.000	0.572 ± 0.000
norm_Laplacian	forwards SFS	0.561 ± 0.000	0.524 ± 0.000	0.440 ± 0.000	0.478 ± 0.000	0.570 ± 0.000
norm_Laplacian	HSIC_Lasso	0.528 ± 0.032	0.477 ± 0.054	0.324 ± 0.047	0.385 ± 0.050	0.524 ± 0.047
norm_Laplacian	Lasso_selection	0.507 ± 0.030	0.425 ± 0.077	0.214 ± 0.056	0.282 ± 0.061	0.482 ± 0.029
norm_Laplacian	mRMR	0.510 ± 0.025	0.437 ± 0.047	0.241 ± 0.052	0.308 ± 0.050	0.501 ± 0.043
norm_Laplacian	Permutation	0.567 ± 0.032	0.533 ± 0.042	0.441 ± 0.048	0.482 ± 0.044	0.548 ± 0.035
norm_Laplacian	Raw data	0.565 ± 0.034	0.531 ± 0.047	0.444 ± 0.043	0.483 ± 0.042	0.559 ± 0.026
partial_corr	backward SFS	0.543 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.504 ± 0.000
partial_corr	forwards SFS	0.543 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.504 ± 0.000
partial_corr	HSIC_Lasso	0.531 ± 0.013	0.307 ± 0.220	0.045 ± 0.038	0.078 ± 0.063	0.466 ± 0.040
partial_corr	Lasso_selection	0.531 ± 0.013	0.307 ± 0.220	0.045 ± 0.038	0.078 ± 0.063	0.466 ± 0.040
partial_corr	mRMR	0.531 ± 0.013	0.307 ± 0.220	0.045 ± 0.038	0.078 ± 0.063	0.466 ± 0.040
partial_corr	Permutation	0.531 ± 0.013	0.307 ± 0.220	0.045 ± 0.038	0.078 ± 0.063	0.466 ± 0.040
partial_corr	Raw data	0.531 ± 0.013	0.307 ± 0.220	0.045 ± 0.038	0.078 ± 0.063	0.466 ± 0.040
rlogspect	backward SFS	0.567 ± 0.000	0.667 ± 0.000	0.107 ± 0.000	0.184 ± 0.000	0.648 ± 0.000
rlogspect	forwards SFS	0.567 ± 0.000	0.667 ± 0.000	0.107 ± 0.000	0.184 ± 0.000	0.648 ± 0.000
rlogspect	HSIC_Lasso	0.533 ± 0.037	0.489 ± 0.083	0.214 ± 0.053	0.293 ± 0.061	0.549 ± 0.049
rlogspect	Lasso_selection	0.533 ± 0.037	0.489 ± 0.083	0.214 ± 0.053	0.293 ± 0.061	0.549 ± 0.049
rlogspect	mRMR	0.533 ± 0.037	0.489 ± 0.083	0.214 ± 0.053	0.293 ± 0.061	0.549 ± 0.049
rlogspect	Permutation	0.533 ± 0.037	0.489 ± 0.083	0.214 ± 0.053	0.293 ± 0.061	0.549 ± 0.049
rlogspect	Raw data	0.533 ± 0.037	0.489 ± 0.083	0.214 ± 0.053	0.293 ± 0.061	0.549 ± 0.049

Table 26: Performance summary for classifier: RandomForest

Inference	Feature Selection	Accuracy	Precision	Sensitivity	F1 Score	AUROC
mutual_info	backward SFS	0.518 ± 0.000	0.466 ± 0.000	0.360 ± 0.000	0.406 ± 0.000	0.530 ± 0.000
mutual_info	forwards SFS	0.543 ± 0.000	0.500 ± 0.000	0.373 ± 0.000	0.427 ± 0.000	0.564 ± 0.000
mutual_info	HSIC_Lasso	0.540 ± 0.053	0.493 ± 0.077	0.385 ± 0.105	0.430 ± 0.090	0.536 ± 0.065
mutual_info	Lasso_selection	0.501 ± 0.029	0.447 ± 0.040	0.391 ± 0.067	0.415 ± 0.054	0.463 ± 0.034
mutual_info	mRMR	0.504 ± 0.042	0.443 ± 0.065	0.356 ± 0.078	0.393 ± 0.074	0.496 ± 0.047
mutual_info	Permutation	0.526 ± 0.031	0.478 ± 0.038	0.393 ± 0.056	0.431 ± 0.047	0.506 ± 0.039
mutual_info	Raw data	0.543 ± 0.024	0.499 ± 0.036	0.393 ± 0.067	0.438 ± 0.054	0.534 ± 0.037
norm_Laplacian	backward SFS	0.512 ± 0.000	0.460 ± 0.000	0.387 ± 0.000	0.420 ± 0.000	0.510 ± 0.000
norm_Laplacian	forwards SFS	0.476 ± 0.000	0.418 ± 0.000	0.373 ± 0.000	0.394 ± 0.000	0.489 ± 0.000
norm_Laplacian	HSIC_Lasso	0.518 ± 0.021	0.463 ± 0.029	0.334 ± 0.040	0.388 ± 0.036	0.511 ± 0.028
norm_Laplacian	Lasso_selection	0.543 ± 0.015	0.503 ± 0.021	0.382 ± 0.023	0.434 ± 0.016	0.504 ± 0.021
norm_Laplacian	mRMR	0.505 ± 0.030	0.450 ± 0.039	0.374 ± 0.056	0.408 ± 0.047	0.475 ± 0.034
norm_Laplacian	Permutation	0.517 ± 0.024	0.468 ± 0.032	0.372 ± 0.029	0.413 ± 0.025	0.521 ± 0.037
norm_Laplacian	Raw data	0.502 ± 0.025	0.441 ± 0.040	0.326 ± 0.035	0.375 ± 0.037	0.505 ± 0.038
partial_corr	backward SFS	0.494 ± 0.000	0.444 ± 0.000	0.427 ± 0.000	0.435 ± 0.000	0.442 ± 0.000
partial_corr	forwards SFS	0.506 ± 0.000	0.456 ± 0.000	0.413 ± 0.000	0.434 ± 0.000	0.462 ± 0.000
partial_corr	HSIC_Lasso	0.511 ± 0.022	0.463 ± 0.028	0.430 ± 0.045	0.446 ± 0.037	0.514 ± 0.027
partial_corr	Lasso_selection	0.509 ± 0.034	0.461 ± 0.038	0.423 ± 0.042	0.440 ± 0.037	0.510 ± 0.020
partial_corr	mRMR	0.521 ± 0.041	0.474 ± 0.050	0.436 ± 0.058	0.454 ± 0.054	0.509 ± 0.024
partial_corr	Permutation	0.516 ± 0.025	0.471 ± 0.028	0.452 ± 0.033	0.461 ± 0.029	0.501 ± 0.011
partial_corr	Raw data	0.516 ± 0.030	0.471 ± 0.032	0.444 ± 0.023	0.457 ± 0.026	0.512 ± 0.020
rlogspect	backward SFS	0.457 ± 0.000	0.400 ± 0.000	0.373 ± 0.000	0.386 ± 0.000	0.458 ± 0.000
rlogspect	forwards SFS	0.494 ± 0.000	0.443 ± 0.000	0.413 ± 0.000	0.428 ± 0.000	0.464 ± 0.000
rlogspect	HSIC_Lasso	0.469 ± 0.010	0.417 ± 0.011	0.393 ± 0.031	0.404 ± 0.019	0.444 ± 0.015
rlogspect	Lasso_selection	0.467 ± 0.013	0.413 ± 0.016	0.391 ± 0.046	0.401 ± 0.030	0.446 ± 0.008
rlogspect	mRMR	0.472 ± 0.024	0.418 ± 0.025	0.382 ± 0.044	0.398 ± 0.027	0.453 ± 0.007
rlogspect	Permutation	0.494 ± 0.013	0.443 ± 0.015	0.409 ± 0.035	0.425 ± 0.024	0.455 ± 0.007
rlogspect	Raw data	0.474 ± 0.013	0.424 ± 0.012	0.404 ± 0.035	0.413 ± 0.017	0.459 ± 0.007

Table 27: Performance summary for classifier: SVM

Inference	Feature Selection	Accuracy	Precision	Sensitivity	F1 Score	AUROC
mutual_info	backward SFS	0.537 ± 0.000	0.490 ± 0.000	0.320 ± 0.000	0.387 ± 0.000	0.528 ± 0.000
mutual_info	forwards SFS	0.543 ± 0.000	0.500 ± 0.000	0.293 ± 0.000	0.370 ± 0.000	0.490 ± 0.000
mutual_info	HSIC_Lasso	0.533 ± 0.018	0.485 ± 0.026	0.350 ± 0.052	0.406 ± 0.044	0.524 ± 0.022
mutual_info	Lasso_selection	0.496 ± 0.050	0.411 ± 0.084	0.241 ± 0.076	0.301 ± 0.081	0.518 ± 0.058
mutual_info	mRMR	0.517 ± 0.020	0.444 ± 0.049	0.262 ± 0.080	0.326 ± 0.079	0.508 ± 0.025
mutual_info	Permutation	0.548 ± 0.022	0.514 ± 0.042	0.350 ± 0.051	0.414 ± 0.037	0.547 ± 0.033
mutual_info	Raw data	0.569 ± 0.003	0.545 ± 0.006	0.364 ± 0.053	0.434 ± 0.037	0.558 ± 0.025
norm_Laplacian	backward SFS	0.518 ± 0.000	0.463 ± 0.000	0.333 ± 0.000	0.388 ± 0.000	0.516 ± 0.000
norm_Laplacian	forwards SFS	0.524 ± 0.000	0.468 ± 0.000	0.293 ± 0.000	0.361 ± 0.000	0.510 ± 0.000
norm_Laplacian	HSIC_Lasso	0.497 ± 0.038	0.449 ± 0.072	0.308 ± 0.066	0.356 ± 0.037	0.479 ± 0.021
norm_Laplacian	Lasso_selection	0.504 ± 0.033	0.440 ± 0.050	0.259 ± 0.032	0.323 ± 0.017	0.482 ± 0.016
norm_Laplacian	mRMR	0.506 ± 0.040	0.442 ± 0.069	0.262 ± 0.062	0.324 ± 0.055	0.495 ± 0.031
norm_Laplacian	Permutation	0.542 ± 0.045	0.507 ± 0.075	0.347 ± 0.053	0.409 ± 0.049	0.488 ± 0.070
norm_Laplacian	Raw data	0.526 ± 0.053	0.484 ± 0.085	0.353 ± 0.056	0.405 ± 0.056	0.472 ± 0.065
partial_corr	backward SFS	0.573 ± 0.000	0.632 ± 0.000	0.160 ± 0.000	0.255 ± 0.000	0.456 ± 0.000
partial_corr	forwards SFS	0.573 ± 0.000	0.632 ± 0.000	0.160 ± 0.000	0.255 ± 0.000	0.456 ± 0.000
partial_corr	HSIC_Lasso	0.550 ± 0.009	0.536 ± 0.041	0.131 ± 0.027	0.210 ± 0.037	0.477 ± 0.035
partial_corr	Lasso_selection	0.550 ± 0.009	0.536 ± 0.041	0.131 ± 0.027	0.210 ± 0.037	0.466 ± 0.023
partial_corr	mRMR	0.550 ± 0.009	0.536 ± 0.041	0.131 ± 0.027	0.210 ± 0.037	0.508 ± 0.041
partial_corr	Permutation	0.550 ± 0.009	0.536 ± 0.041	0.131 ± 0.027	0.210 ± 0.037	0.489 ± 0.040
partial_corr	Raw data	0.550 ± 0.009	0.536 ± 0.041	0.131 ± 0.027	0.210 ± 0.037	0.491 ± 0.040
rlogspect	backward SFS	0.524 ± 0.000	0.333 ± 0.000	0.040 ± 0.000	0.071 ± 0.000	0.559 ± 0.000
rlogspect	forwards SFS	0.524 ± 0.000	0.333 ± 0.000	0.040 ± 0.000	0.071 ± 0.000	0.559 ± 0.000
rlogspect	HSIC_Lasso	0.536 ± 0.017	0.190 ± 0.263	0.016 ± 0.021	0.028 ± 0.037	0.522 ± 0.048
rlogspect	Lasso_selection	0.536 ± 0.017	0.190 ± 0.263	0.016 ± 0.021	0.028 ± 0.037	0.549 ± 0.023
rlogspect	mRMR	0.536 ± 0.017	0.190 ± 0.263	0.016 ± 0.021	0.028 ± 0.037	0.549 ± 0.023
rlogspect	Permutation	0.536 ± 0.017	0.190 ± 0.263	0.016 ± 0.021	0.028 ± 0.037	0.548 ± 0.023
rlogspect	Raw data	0.536 ± 0.017	0.190 ± 0.263	0.016 ± 0.021	0.028 ± 0.037	0.549 ± 0.022

B.2 graph NYU

Table 28: Performance summary for classifier: KNN

Inference	Feature Selection	Accuracy	Precision	Sensitivity	F1 Score	AUROC
mutual_info	backward SFS	0.471 ± 0.000	0.333 ± 0.000	0.200 ± 0.000	0.250 ± 0.000	0.421 ± 0.000
mutual_info	forwards SFS	0.500 ± 0.000	0.444 ± 0.000	0.533 ± 0.000	0.485 ± 0.000	0.546 ± 0.000
mutual_info	HSIC_Lasso	0.529 ± 0.066	0.409 ± 0.132	0.293 ± 0.137	0.337 ± 0.138	0.494 ± 0.087
mutual_info	Lasso_selection	0.542 ± 0.047	0.444 ± 0.104	0.338 ± 0.105	0.381 ± 0.102	0.545 ± 0.035
mutual_info	mRMR	0.518 ± 0.071	0.416 ± 0.102	0.324 ± 0.096	0.363 ± 0.098	0.478 ± 0.053
mutual_info	Permutation	0.500 ± 0.059	0.390 ± 0.076	0.281 ± 0.074	0.321 ± 0.071	0.466 ± 0.077
mutual_info	Raw data	0.494 ± 0.048	0.402 ± 0.055	0.353 ± 0.068	0.372 ± 0.048	0.466 ± 0.054
norm_Laplacian	backward SFS	0.588 ± 0.000	0.538 ± 0.000	0.467 ± 0.000	0.500 ± 0.000	0.644 ± 0.000
norm_Laplacian	forwards SFS	0.500 ± 0.000	0.375 ± 0.000	0.200 ± 0.000	0.261 ± 0.000	0.488 ± 0.000
norm_Laplacian	HSIC_Lasso	0.518 ± 0.100	0.431 ± 0.110	0.339 ± 0.107	0.374 ± 0.105	0.494 ± 0.112
norm_Laplacian	Lasso_selection	0.469 ± 0.069	0.309 ± 0.167	0.224 ± 0.118	0.258 ± 0.136	0.427 ± 0.089
norm_Laplacian	mRMR	0.464 ± 0.044	0.351 ± 0.052	0.297 ± 0.058	0.320 ± 0.051	0.432 ± 0.056
norm_Laplacian	Permutation	0.537 ± 0.076	0.465 ± 0.112	0.424 ± 0.123	0.435 ± 0.091	0.494 ± 0.095
norm_Laplacian	Raw data	0.506 ± 0.018	0.422 ± 0.030	0.436 ± 0.080	0.427 ± 0.049	0.494 ± 0.041
partial_corr	backward SFS	0.471 ± 0.000	0.364 ± 0.000	0.267 ± 0.000	0.308 ± 0.000	0.461 ± 0.000
partial_corr	forwards SFS	0.471 ± 0.000	0.364 ± 0.000	0.267 ± 0.000	0.308 ± 0.000	0.461 ± 0.000
partial_corr	HSIC_Lasso	0.488 ± 0.045	0.372 ± 0.068	0.296 ± 0.071	0.329 ± 0.070	0.442 ± 0.038
partial_corr	Lasso_selection	0.488 ± 0.045	0.372 ± 0.068	0.296 ± 0.071	0.329 ± 0.070	0.442 ± 0.038
partial_corr	mRMR	0.488 ± 0.045	0.372 ± 0.068	0.296 ± 0.071	0.329 ± 0.070	0.442 ± 0.038
partial_corr	Permutation	0.488 ± 0.045	0.372 ± 0.068	0.296 ± 0.071	0.329 ± 0.070	0.442 ± 0.038
partial_corr	Raw data	0.488 ± 0.045	0.372 ± 0.068	0.296 ± 0.071	0.329 ± 0.070	0.442 ± 0.038

Table 29: Performance summary for classifier: LDA

Inference	Feature Selection	Accuracy	Precision	Sensitivity	F1 Score	AUROC
mutual_info	backward SFS	0.618 ± 0.000	0.571 ± 0.000	0.533 ± 0.000	0.552 ± 0.000	0.653 ± 0.000
mutual_info	forwards SFS	0.676 ± 0.000	0.667 ± 0.000	0.533 ± 0.000	0.593 ± 0.000	0.698 ± 0.000
mutual_info	HSIC_Lasso	0.512 ± 0.078	0.421 ± 0.098	0.381 ± 0.099	0.399 ± 0.098	0.496 ± 0.077
mutual_info	Lasso_selection	0.506 ± 0.060	0.393 ± 0.097	0.268 ± 0.053	0.318 ± 0.069	0.486 ± 0.068
mutual_info	mRMR	0.464 ± 0.085	0.356 ± 0.107	0.282 ± 0.064	0.313 ± 0.081	0.464 ± 0.146
mutual_info	Permutation	0.506 ± 0.071	0.440 ± 0.072	0.466 ± 0.077	0.446 ± 0.048	0.524 ± 0.051
mutual_info	Raw data	0.493 ± 0.068	0.428 ± 0.069	0.451 ± 0.076	0.432 ± 0.037	0.525 ± 0.050
norm_Laplacian	backward SFS	0.618 ± 0.000	0.600 ± 0.000	0.400 ± 0.000	0.480 ± 0.000	0.614 ± 0.000
norm_Laplacian	forwards SFS	0.618 ± 0.000	0.562 ± 0.000	0.600 ± 0.000	0.581 ± 0.000	0.663 ± 0.000
norm_Laplacian	HSIC_Lasso	0.596 ± 0.097	0.526 ± 0.117	0.505 ± 0.155	0.510 ± 0.129	0.604 ± 0.108
norm_Laplacian	Lasso_selection	0.524 ± 0.055	0.420 ± 0.088	0.294 ± 0.075	0.344 ± 0.079	0.460 ± 0.064
norm_Laplacian	mRMR	0.518 ± 0.053	0.420 ± 0.084	0.338 ± 0.083	0.373 ± 0.079	0.465 ± 0.085
norm_Laplacian	Permutation	0.602 ± 0.043	0.540 ± 0.064	0.490 ± 0.103	0.510 ± 0.069	0.593 ± 0.079
norm_Laplacian	Raw data	0.584 ± 0.035	0.513 ± 0.049	0.477 ± 0.081	0.493 ± 0.061	0.605 ± 0.071
partial_corr	backward SFS	0.559 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.526 ± 0.000
partial_corr	forwards SFS	0.559 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.526 ± 0.000
partial_corr	HSIC_Lasso	0.524 ± 0.042	0.275 ± 0.174	0.100 ± 0.073	0.140 ± 0.092	0.406 ± 0.059
partial_corr	Lasso_selection	0.524 ± 0.042	0.275 ± 0.174	0.100 ± 0.073	0.140 ± 0.092	0.406 ± 0.059
partial_corr	mRMR	0.524 ± 0.042	0.275 ± 0.174	0.100 ± 0.073	0.140 ± 0.092	0.406 ± 0.059
partial_corr	Permutation	0.524 ± 0.042	0.275 ± 0.174	0.100 ± 0.073	0.140 ± 0.092	0.406 ± 0.059
partial_corr	Raw data	0.524 ± 0.042	0.275 ± 0.174	0.100 ± 0.073	0.140 ± 0.092	0.406 ± 0.059

Table 30: Performance summary for classifier: LogR

Inference	Feature Selection	Accuracy	Precision	Sensitivity	F1 Score	AUROC
mutual_info	backward SFS	0.559 ± 0.000	0.500 ± 0.000	0.400 ± 0.000	0.444 ± 0.000	0.572 ± 0.000
mutual_info	forwards SFS	0.618 ± 0.000	0.600 ± 0.000	0.400 ± 0.000	0.480 ± 0.000	0.698 ± 0.000
mutual_info	HSIC_Lasso	0.518 ± 0.067	0.460 ± 0.150	0.310 ± 0.075	0.353 ± 0.046	0.453 ± 0.090
mutual_info	Lasso_selection	0.537 ± 0.040	0.422 ± 0.067	0.226 ± 0.055	0.292 ± 0.062	0.524 ± 0.055
mutual_info	mRMR	0.458 ± 0.039	0.321 ± 0.037	0.240 ± 0.074	0.269 ± 0.056	0.461 ± 0.105
mutual_info	Permutation	0.494 ± 0.058	0.401 ± 0.075	0.411 ± 0.143	0.401 ± 0.105	0.494 ± 0.077
mutual_info	Raw data	0.482 ± 0.085	0.376 ± 0.108	0.397 ± 0.192	0.382 ± 0.146	0.498 ± 0.097
norm_Laplacian	backward SFS	0.559 ± 0.000	0.500 ± 0.000	0.400 ± 0.000	0.444 ± 0.000	0.533 ± 0.000
norm_Laplacian	forwards SFS	0.618 ± 0.000	0.571 ± 0.000	0.533 ± 0.000	0.552 ± 0.000	0.653 ± 0.000
norm_Laplacian	HSIC_Lasso	0.548 ± 0.064	0.457 ± 0.118	0.310 ± 0.095	0.365 ± 0.104	0.524 ± 0.086
norm_Laplacian	Lasso_selection	0.506 ± 0.070	0.405 ± 0.101	0.296 ± 0.055	0.341 ± 0.071	0.453 ± 0.066
norm_Laplacian	mRMR	0.548 ± 0.061	0.457 ± 0.098	0.352 ± 0.101	0.397 ± 0.100	0.459 ± 0.085
norm_Laplacian	Permutation	0.524 ± 0.045	0.429 ± 0.067	0.351 ± 0.086	0.383 ± 0.076	0.449 ± 0.064
norm_Laplacian	Raw data	0.530 ± 0.041	0.435 ± 0.066	0.351 ± 0.086	0.386 ± 0.077	0.465 ± 0.050
partial_corr	backward SFS	0.500 ± 0.000	0.375 ± 0.000	0.200 ± 0.000	0.261 ± 0.000	0.530 ± 0.000
partial_corr	forwards SFS	0.500 ± 0.000	0.375 ± 0.000	0.200 ± 0.000	0.261 ± 0.000	0.530 ± 0.000
partial_corr	HSIC_Lasso	0.536 ± 0.044	0.424 ± 0.330	0.113 ± 0.073	0.160 ± 0.091	0.509 ± 0.054
partial_corr	Lasso_selection	0.536 ± 0.044	0.424 ± 0.330	0.113 ± 0.073	0.160 ± 0.091	0.509 ± 0.054
partial_corr	mRMR	0.536 ± 0.044	0.424 ± 0.330	0.113 ± 0.073	0.160 ± 0.091	0.509 ± 0.054
partial_corr	Permutation	0.536 ± 0.044	0.424 ± 0.330	0.113 ± 0.073	0.160 ± 0.091	0.509 ± 0.054
partial_corr	Raw data	0.536 ± 0.044	0.424 ± 0.330	0.113 ± 0.073	0.160 ± 0.091	0.509 ± 0.054

Table 31: Performance summary for classifier: RandomForest

Inference	Feature Selection	Accuracy	Precision	Sensitivity	F1 Score	AUROC
mutual_info	backward SFS	0.559 ± 0.000	0.500 ± 0.000	0.267 ± 0.000	0.348 ± 0.000	0.526 ± 0.000
mutual_info	forwards SFS	0.559 ± 0.000	0.500 ± 0.000	0.400 ± 0.000	0.444 ± 0.000	0.449 ± 0.000
mutual_info	HSIC_Lasso	0.518 ± 0.054	0.400 ± 0.093	0.226 ± 0.055	0.285 ± 0.060	0.485 ± 0.072
mutual_info	Lasso_selection	0.506 ± 0.074	0.354 ± 0.151	0.240 ± 0.148	0.282 ± 0.150	0.508 ± 0.028
mutual_info	mRMR	0.554 ± 0.065	0.458 ± 0.122	0.338 ± 0.123	0.387 ± 0.123	0.497 ± 0.089
mutual_info	Permutation	0.554 ± 0.052	0.508 ± 0.131	0.295 ± 0.081	0.356 ± 0.049	0.543 ± 0.050
mutual_info	Raw data	0.542 ± 0.036	0.432 ± 0.088	0.182 ± 0.068	0.247 ± 0.080	0.507 ± 0.047
norm_Laplacian	HSIC_Lasso	0.476 ± 0.040	0.277 ± 0.114	0.169 ± 0.106	0.207 ± 0.113	0.430 ± 0.042
norm_Laplacian	Lasso_selection	0.500 ± 0.033	0.368 ± 0.058	0.254 ± 0.087	0.297 ± 0.080	0.433 ± 0.033
norm_Laplacian	mRMR	0.470 ± 0.101	0.356 ± 0.135	0.310 ± 0.134	0.332 ± 0.134	0.429 ± 0.080
norm_Laplacian	Permutation	0.483 ± 0.072	0.321 ± 0.181	0.243 ± 0.132	0.275 ± 0.150	0.426 ± 0.072
norm_Laplacian	Raw data	0.524 ± 0.080	0.414 ± 0.153	0.211 ± 0.078	0.276 ± 0.096	0.451 ± 0.090
partial_corr	backward SFS	0.500 ± 0.000	0.375 ± 0.000	0.200 ± 0.000	0.261 ± 0.000	0.439 ± 0.000
partial_corr	forwards SFS	0.441 ± 0.000	0.300 ± 0.000	0.200 ± 0.000	0.240 ± 0.000	0.435 ± 0.000
partial_corr	HSIC_Lasso	0.512 ± 0.041	0.406 ± 0.073	0.367 ± 0.139	0.379 ± 0.109	0.470 ± 0.035
partial_corr	Lasso_selection	0.518 ± 0.047	0.422 ± 0.070	0.367 ± 0.096	0.390 ± 0.083	0.470 ± 0.030
partial_corr	mRMR	0.506 ± 0.030	0.396 ± 0.076	0.367 ± 0.139	0.376 ± 0.110	0.453 ± 0.021
partial_corr	Permutation	0.512 ± 0.048	0.426 ± 0.076	0.367 ± 0.096	0.386 ± 0.065	0.440 ± 0.017
partial_corr	Raw data	0.506 ± 0.026	0.394 ± 0.064	0.339 ± 0.140	0.356 ± 0.104	0.465 ± 0.041

Table 32: Performance summary for classifier: SVM

Inference	Feature Selection	Accuracy	Precision	Sensitivity	F1 Score	AUROC
mutual_info	backward SFS	0.471 ± 0.000	0.286 ± 0.000	0.133 ± 0.000	0.182 ± 0.000	0.481 ± 0.000
mutual_info	forwards SFS	0.471 ± 0.000	0.200 ± 0.000	0.067 ± 0.000	0.100 ± 0.000	0.533 ± 0.000
mutual_info	HSIC_Lasso	0.500 ± 0.043	0.303 ± 0.099	0.139 ± 0.071	0.187 ± 0.083	0.515 ± 0.115
mutual_info	Lasso_selection	0.494 ± 0.057	0.355 ± 0.096	0.227 ± 0.085	0.273 ± 0.087	0.453 ± 0.115
mutual_info	mRMR	0.518 ± 0.043	0.373 ± 0.092	0.183 ± 0.056	0.244 ± 0.068	0.498 ± 0.068
mutual_info	Permutation	0.524 ± 0.023	0.320 ± 0.051	0.098 ± 0.033	0.147 ± 0.039	0.483 ± 0.063
mutual_info	Raw data	0.542 ± 0.051	0.413 ± 0.342	0.070 ± 0.042	0.112 ± 0.065	0.532 ± 0.054
norm_Laplacian	forwards SFS	0.588 ± 0.000	0.667 ± 0.000	0.133 ± 0.000	0.222 ± 0.000	0.488 ± 0.000
norm_Laplacian	HSIC_Lasso	0.536 ± 0.049	0.386 ± 0.340	0.085 ± 0.070	0.126 ± 0.092	0.505 ± 0.085
norm_Laplacian	Lasso_selection	0.566 ± 0.035	0.483 ± 0.072	0.282 ± 0.091	0.351 ± 0.087	0.521 ± 0.096
norm_Laplacian	mRMR	0.530 ± 0.038	0.351 ± 0.179	0.168 ± 0.093	0.226 ± 0.121	0.498 ± 0.035
norm_Laplacian	Permutation	0.542 ± 0.085	0.450 ± 0.348	0.112 ± 0.072	0.178 ± 0.118	0.547 ± 0.076
norm_Laplacian	Raw data	0.524 ± 0.056	0.359 ± 0.141	0.098 ± 0.033	0.151 ± 0.051	0.493 ± 0.077
partial_corr	backward SFS	0.529 ± 0.000	0.462 ± 0.000	0.400 ± 0.000	0.429 ± 0.000	0.463 ± 0.000
partial_corr	forwards SFS	0.529 ± 0.000	0.462 ± 0.000	0.400 ± 0.000	0.429 ± 0.000	0.463 ± 0.000
partial_corr	HSIC_Lasso	0.537 ± 0.048	0.380 ± 0.147	0.213 ± 0.129	0.268 ± 0.141	0.455 ± 0.062
partial_corr	Lasso_selection	0.537 ± 0.048	0.380 ± 0.147	0.213 ± 0.129	0.268 ± 0.141	0.455 ± 0.062
partial_corr	mRMR	0.537 ± 0.048	0.380 ± 0.147	0.213 ± 0.129	0.268 ± 0.141	0.466 ± 0.067
partial_corr	Permutation	0.537 ± 0.048	0.380 ± 0.147	0.213 ± 0.129	0.268 ± 0.141	0.487 ± 0.075
partial_corr	Raw data	0.537 ± 0.048	0.380 ± 0.147	0.213 ± 0.129	0.268 ± 0.141	0.469 ± 0.068

B.3 Laplacian NYU

Table 33: Performance summary for classifier: KNN

Feature Selection	Accuracy	Precision	Sensitivity	F1 Score	AUROC
backward SFS	0.543 ± 0.000	0.455 ± 0.000	0.333 ± 0.000	0.385 ± 0.000	0.567 ± 0.000
forwards SFS	0.657 ± 0.000	0.636 ± 0.000	0.467 ± 0.000	0.538 ± 0.000	0.608 ± 0.000
HSIC_Lasso	0.529 ± 0.068	0.444 ± 0.114	0.284 ± 0.049	0.343 ± 0.064	0.489 ± 0.038
Lasso_selection	0.552 ± 0.015	0.481 ± 0.016	0.448 ± 0.099	0.456 ± 0.057	0.512 ± 0.029
mRMR	0.547 ± 0.064	0.472 ± 0.099	0.378 ± 0.107	0.413 ± 0.090	0.549 ± 0.042
Permutation	0.535 ± 0.071	0.484 ± 0.147	0.285 ± 0.056	0.344 ± 0.045	0.499 ± 0.071

Table 34: Performance summary for classifier: LDA

Feature Selection	Accuracy	Precision	Sensitivity	F1 Score	AUROC
backward SFS	0.629 ± 0.000	0.600 ± 0.000	0.400 ± 0.000	0.480 ± 0.000	0.597 ± 0.000
forwards SFS	0.571 ± 0.000	0.500 ± 0.000	0.400 ± 0.000	0.444 ± 0.000	0.603 ± 0.000
HSIC_Lasso	0.599 ± 0.086	0.556 ± 0.139	0.486 ± 0.041	0.515 ± 0.079	0.610 ± 0.094
Lasso_selection	0.663 ± 0.045	0.663 ± 0.098	0.472 ± 0.109	0.541 ± 0.076	0.676 ± 0.071
mRMR	0.593 ± 0.069	0.564 ± 0.163	0.366 ± 0.094	0.433 ± 0.093	0.600 ± 0.062
Permutation	0.564 ± 0.076	0.485 ± 0.089	0.542 ± 0.178	0.505 ± 0.127	0.622 ± 0.070

Table 35: Performance summary for classifier: LogR

Feature Selection	Accuracy	Precision	Sensitivity	F1 Score	AUROC
backward SFS	0.600 ± 0.000	0.556 ± 0.000	0.333 ± 0.000	0.417 ± 0.000	0.607 ± 0.000
forwards SFS	0.571 ± 0.000	0.500 ± 0.000	0.400 ± 0.000	0.444 ± 0.000	0.617 ± 0.000
HSIC_Lasso	0.570 ± 0.060	0.508 ± 0.130	0.338 ± 0.074	0.402 ± 0.085	0.617 ± 0.091
Lasso_selection	0.575 ± 0.064	0.552 ± 0.135	0.406 ± 0.074	0.450 ± 0.029	0.575 ± 0.032
mRMR	0.598 ± 0.075	0.576 ± 0.170	0.366 ± 0.094	0.437 ± 0.098	0.587 ± 0.069
Permutation	0.575 ± 0.073	0.500 ± 0.111	0.489 ± 0.156	0.488 ± 0.125	0.573 ± 0.091

Table 36: Performance summary for classifier: RandomForest

Feature Selection	Accuracy	Precision	Sensitivity	F1 Score	AUROC
backward SFS	0.543 ± 0.000	0.455 ± 0.000	0.333 ± 0.000	0.385 ± 0.000	0.537 ± 0.000
forwards SFS	0.571 ± 0.000	0.500 ± 0.000	0.400 ± 0.000	0.444 ± 0.000	0.555 ± 0.000
HSIC_Lasso	0.575 ± 0.121	0.533 ± 0.193	0.363 ± 0.127	0.426 ± 0.144	0.572 ± 0.135
Lasso_selection	0.610 ± 0.117	0.574 ± 0.141	0.477 ± 0.165	0.508 ± 0.131	0.606 ± 0.124
mRMR	0.512 ± 0.050	0.427 ± 0.068	0.408 ± 0.106	0.414 ± 0.081	0.590 ± 0.070
Permutation	0.633 ± 0.115	0.607 ± 0.175	0.436 ± 0.186	0.494 ± 0.168	0.621 ± 0.122

Table 37: Performance summary for classifier: SVM

Feature Selection	Accuracy	Precision	Sensitivity	F1 Score	AUROC
backward SFS	0.457 ± 0.000	0.333 ± 0.000	0.267 ± 0.000	0.296 ± 0.000	0.573 ± 0.000
forwards SFS	0.629 ± 0.000	0.750 ± 0.000	0.200 ± 0.000	0.316 ± 0.000	0.260 ± 0.000
HSIC_Lasso	0.599 ± 0.057	0.562 ± 0.143	0.270 ± 0.095	0.362 ± 0.115	0.545 ± 0.067
Lasso_selection	0.593 ± 0.038	0.552 ± 0.100	0.377 ± 0.121	0.435 ± 0.081	0.514 ± 0.119
mRMR	0.581 ± 0.014	0.525 ± 0.047	0.297 ± 0.030	0.379 ± 0.032	0.497 ± 0.065
Permutation	0.598 ± 0.068	0.573 ± 0.119	0.379 ± 0.039	0.451 ± 0.049	0.596 ± 0.102

B.4 Laplacian multisite

Table 38: Performance summary for classifier: KNN

Feature Selection	Accuracy	Precision	Sensitivity	F1 Score	AUROC
backward SFS	0.537 ± 0.000	0.500 ± 0.000	0.444 ± 0.000	0.471 ± 0.000	0.548 ± 0.000
forwards SFS	0.531 ± 0.000	0.492 ± 0.000	0.383 ± 0.000	0.431 ± 0.000	0.540 ± 0.000
HSIC_Lasso	0.512 ± 0.046	0.469 ± 0.052	0.449 ± 0.091	0.457 ± 0.070	0.506 ± 0.046
Lasso_selection	0.512 ± 0.019	0.472 ± 0.022	0.454 ± 0.027	0.463 ± 0.023	0.524 ± 0.034
mRMR	0.548 ± 0.020	0.512 ± 0.021	0.506 ± 0.051	0.508 ± 0.030	0.564 ± 0.038
Permutation	0.504 ± 0.013	0.461 ± 0.014	0.432 ± 0.047	0.445 ± 0.028	0.498 ± 0.033

Table 39: Performance summary for classifier: LDA

Feature Selection	Accuracy	Precision	Sensitivity	F1 Score	AUROC
backward SFS	0.634 ± 0.000	0.631 ± 0.000	0.506 ± 0.000	0.562 ± 0.000	0.630 ± 0.000
forwards SFS	0.589 ± 0.000	0.563 ± 0.000	0.494 ± 0.000	0.526 ± 0.000	0.579 ± 0.000
HSIC_Lasso	0.543 ± 0.019	0.505 ± 0.023	0.452 ± 0.059	0.476 ± 0.042	0.561 ± 0.019
Lasso_selection	0.550 ± 0.035	0.519 ± 0.053	0.400 ± 0.041	0.451 ± 0.042	0.544 ± 0.046
mRMR	0.542 ± 0.020	0.506 ± 0.028	0.397 ± 0.045	0.444 ± 0.037	0.560 ± 0.030
Permutation	0.588 ± 0.037	0.552 ± 0.039	0.590 ± 0.035	0.570 ± 0.032	0.618 ± 0.029

Table 40: Performance summary for classifier: LogR

Feature Selection	Accuracy	Precision	Sensitivity	F1 Score	AUROC
backward SFS	0.634 ± 0.000	0.631 ± 0.000	0.506 ± 0.000	0.562 ± 0.000	0.630 ± 0.000
forwards SFS	0.594 ± 0.000	0.569 ± 0.000	0.506 ± 0.000	0.536 ± 0.000	0.579 ± 0.000
HSIC_Lasso	0.557 ± 0.019	0.525 ± 0.027	0.469 ± 0.031	0.495 ± 0.017	0.589 ± 0.036
Lasso_selection	0.560 ± 0.048	0.535 ± 0.073	0.407 ± 0.057	0.461 ± 0.057	0.581 ± 0.051
mRMR	0.542 ± 0.019	0.507 ± 0.027	0.400 ± 0.044	0.445 ± 0.034	0.561 ± 0.030
Permutation	0.590 ± 0.043	0.556 ± 0.045	0.566 ± 0.059	0.560 ± 0.049	0.615 ± 0.040

Table 41: Performance summary for classifier: RandomForest

Feature Selection	Accuracy	Precision	Sensitivity	F1 Score	AUROC
backward SFS	0.543 ± 0.000	0.508 ± 0.000	0.383 ± 0.000	0.437 ± 0.000	0.566 ± 0.000
forwards SFS	0.577 ± 0.000	0.552 ± 0.000	0.457 ± 0.000	0.500 ± 0.000	0.562 ± 0.000
HSIC_Lasso	0.572 ± 0.013	0.545 ± 0.024	0.471 ± 0.032	0.504 ± 0.015	0.592 ± 0.027
Lasso_selection	0.522 ± 0.024	0.483 ± 0.028	0.439 ± 0.027	0.460 ± 0.025	0.529 ± 0.018
mRMR	0.524 ± 0.025	0.481 ± 0.033	0.436 ± 0.066	0.457 ± 0.050	0.547 ± 0.023
Permutation	0.547 ± 0.038	0.514 ± 0.053	0.409 ± 0.041	0.455 ± 0.043	0.564 ± 0.035

Table 42: Performance summary for classifier: SVM

Feature Selection	Accuracy	Precision	Sensitivity	F1 Score	AUROC
backward SFS	0.600 ± 0.000	0.571 ± 0.000	0.543 ± 0.000	0.557 ± 0.000	0.601 ± 0.000
forwards SFS	0.543 ± 0.000	0.509 ± 0.000	0.346 ± 0.000	0.412 ± 0.000	0.566 ± 0.000
HSIC_Lasso	0.542 ± 0.041	0.509 ± 0.057	0.402 ± 0.043	0.448 ± 0.042	0.547 ± 0.031
Lasso_selection	0.553 ± 0.030	0.520 ± 0.039	0.412 ± 0.052	0.459 ± 0.047	0.558 ± 0.023
mRMR	0.577 ± 0.032	0.559 ± 0.049	0.424 ± 0.029	0.482 ± 0.033	0.557 ± 0.080
Permutation	0.596 ± 0.030	0.570 ± 0.030	0.506 ± 0.066	0.535 ± 0.049	0.634 ± 0.021

B.5 rspect NYU

Table 43: Performance summary for classifier: KNN

Feature Selection	Accuracy	Precision	Sensitivity	F1 Score	AUROC
backward SFS	0.571 ± 0.000	0.500 ± 0.000	0.133 ± 0.000	0.211 ± 0.000	0.542 ± 0.000
forwards SFS	0.514 ± 0.000	0.417 ± 0.000	0.333 ± 0.000	0.370 ± 0.000	0.513 ± 0.000
HSIC_Lasso	0.604 ± 0.094	0.582 ± 0.154	0.409 ± 0.164	0.459 ± 0.135	0.588 ± 0.105
Lasso_selection	0.546 ± 0.078	0.470 ± 0.116	0.310 ± 0.078	0.371 ± 0.085	0.492 ± 0.092
mRMR	0.529 ± 0.021	0.431 ± 0.045	0.337 ± 0.092	0.375 ± 0.077	0.503 ± 0.047
Permutation	0.530 ± 0.067	0.404 ± 0.158	0.271 ± 0.136	0.320 ± 0.146	0.515 ± 0.057

Table 44: Performance summary for classifier: LDA

Feature Selection	Accuracy	Precision	Sensitivity	F1 Score	AUROC
backward SFS	0.457 ± 0.000	0.333 ± 0.000	0.267 ± 0.000	0.296 ± 0.000	0.513 ± 0.000
forwards SFS	0.600 ± 0.000	0.538 ± 0.000	0.467 ± 0.000	0.500 ± 0.000	0.563 ± 0.000
HSIC_Lasso	0.604 ± 0.052	0.570 ± 0.100	0.448 ± 0.123	0.485 ± 0.082	0.662 ± 0.043
Lasso_selection	0.581 ± 0.097	0.549 ± 0.184	0.418 ± 0.127	0.459 ± 0.115	0.585 ± 0.102
mRMR	0.581 ± 0.086	0.593 ± 0.230	0.405 ± 0.144	0.443 ± 0.115	0.569 ± 0.094
Permutation	0.575 ± 0.128	0.525 ± 0.165	0.470 ± 0.147	0.488 ± 0.143	0.610 ± 0.134

Table 45: Performance summary for classifier: LogR

Feature Selection	Accuracy	Precision	Sensitivity	F1 Score	AUROC
backward SFS	0.457 ± 0.000	0.300 ± 0.000	0.200 ± 0.000	0.240 ± 0.000	0.517 ± 0.000
forwards SFS	0.629 ± 0.000	0.583 ± 0.000	0.467 ± 0.000	0.519 ± 0.000	0.567 ± 0.000
HSIC_Lasso	0.605 ± 0.082	0.578 ± 0.138	0.460 ± 0.033	0.505 ± 0.059	0.581 ± 0.142
Lasso_selection	0.575 ± 0.069	0.535 ± 0.132	0.406 ± 0.164	0.435 ± 0.120	0.555 ± 0.105
mRMR	0.581 ± 0.086	0.593 ± 0.230	0.405 ± 0.144	0.443 ± 0.115	0.573 ± 0.098
Permutation	0.518 ± 0.097	0.445 ± 0.128	0.447 ± 0.111	0.443 ± 0.114	0.522 ± 0.085

Table 46: Performance summary for classifier: RandomForest

Feature Selection	Accuracy	Precision	Sensitivity	F1 Score	AUROC
backward SFS	0.600 ± 0.000	0.545 ± 0.000	0.400 ± 0.000	0.462 ± 0.000	0.582 ± 0.000
forwards SFS	0.686 ± 0.000	0.700 ± 0.000	0.467 ± 0.000	0.560 ± 0.000	0.683 ± 0.000
HSIC_Lasso	0.564 ± 0.063	0.503 ± 0.096	0.326 ± 0.059	0.391 ± 0.061	0.550 ± 0.078
Lasso_selection	0.558 ± 0.055	0.467 ± 0.133	0.258 ± 0.111	0.323 ± 0.126	0.536 ± 0.090
mRMR	0.564 ± 0.092	0.519 ± 0.182	0.296 ± 0.106	0.365 ± 0.117	0.521 ± 0.062
Permutation	0.541 ± 0.083	0.481 ± 0.183	0.314 ± 0.138	0.363 ± 0.111	0.521 ± 0.161

Table 47: Performance summary for classifier: SVM

Feature Selection	Accuracy	Precision	Sensitivity	F1 Score	AUROC
backward SFS	0.543 ± 0.000	0.444 ± 0.000	0.267 ± 0.000	0.333 ± 0.000	0.537 ± 0.000
forwards SFS	0.600 ± 0.000	0.667 ± 0.000	0.133 ± 0.000	0.222 ± 0.000	0.310 ± 0.000
HSIC_Lasso	0.598 ± 0.052	0.579 ± 0.102	0.352 ± 0.102	0.423 ± 0.085	0.558 ± 0.119
Lasso_selection	0.586 ± 0.074	0.634 ± 0.247	0.310 ± 0.130	0.381 ± 0.116	0.488 ± 0.134
mRMR	0.598 ± 0.066	0.678 ± 0.264	0.299 ± 0.114	0.380 ± 0.106	0.554 ± 0.116
Permutation	0.621 ± 0.062	0.608 ± 0.148	0.421 ± 0.177	0.471 ± 0.127	0.618 ± 0.070

B.6 rspect multisite

Table 48: Performance summary for classifier: KNN

Feature Selection	Accuracy	Precision	Sensitivity	F1 Score	AUROC
backward SFS	0.543 ± 0.000	0.507 ± 0.000	0.457 ± 0.000	0.481 ± 0.000	0.538 ± 0.000
forwards SFS	0.560 ± 0.000	0.522 ± 0.000	0.593 ± 0.000	0.555 ± 0.000	0.542 ± 0.000
HSIC_Lasso	0.519 ± 0.036	0.480 ± 0.048	0.402 ± 0.023	0.437 ± 0.029	0.521 ± 0.037
Lasso_selection	0.520 ± 0.036	0.480 ± 0.038	0.444 ± 0.047	0.461 ± 0.039	0.517 ± 0.036
mRMR	0.519 ± 0.015	0.478 ± 0.018	0.424 ± 0.039	0.449 ± 0.025	0.523 ± 0.010
Permutation	0.521 ± 0.008	0.478 ± 0.011	0.395 ± 0.036	0.432 ± 0.025	0.532 ± 0.008

Table 49: Performance summary for classifier: LDA

Feature Selection	Accuracy	Precision	Sensitivity	F1 Score	AUROC
backward SFS	0.537 ± 0.000	0.500 ± 0.000	0.469 ± 0.000	0.484 ± 0.000	0.552 ± 0.000
forwards SFS	0.560 ± 0.000	0.525 ± 0.000	0.519 ± 0.000	0.522 ± 0.000	0.587 ± 0.000
HSIC_Lasso	0.560 ± 0.030	0.530 ± 0.036	0.420 ± 0.072	0.466 ± 0.054	0.566 ± 0.036
Lasso_selection	0.542 ± 0.016	0.506 ± 0.023	0.345 ± 0.057	0.408 ± 0.045	0.540 ± 0.040
mRMR	0.544 ± 0.014	0.510 ± 0.021	0.350 ± 0.038	0.414 ± 0.031	0.531 ± 0.028
Permutation	0.582 ± 0.028	0.551 ± 0.034	0.513 ± 0.059	0.531 ± 0.045	0.603 ± 0.040

Table 50: Performance summary for classifier: LogR

Feature Selection	Accuracy	Precision	Sensitivity	F1 Score	AUROC
backward SFS	0.520 ± 0.000	0.479 ± 0.000	0.432 ± 0.000	0.455 ± 0.000	0.552 ± 0.000
forwards SFS	0.566 ± 0.000	0.532 ± 0.000	0.519 ± 0.000	0.525 ± 0.000	0.587 ± 0.000
HSIC_Lasso	0.573 ± 0.032	0.544 ± 0.038	0.452 ± 0.066	0.493 ± 0.054	0.592 ± 0.025
Lasso_selection	0.543 ± 0.024	0.506 ± 0.035	0.360 ± 0.063	0.419 ± 0.053	0.542 ± 0.028
mRMR	0.542 ± 0.013	0.507 ± 0.019	0.345 ± 0.035	0.410 ± 0.029	0.531 ± 0.028
Permutation	0.592 ± 0.026	0.560 ± 0.028	0.548 ± 0.053	0.554 ± 0.038	0.625 ± 0.029

Table 51: Performance summary for classifier: RandomForest

Feature Selection	Accuracy	Precision	Sensitivity	F1 Score	AUROC
backward SFS	0.549 ± 0.000	0.514 ± 0.000	0.444 ± 0.000	0.477 ± 0.000	0.561 ± 0.000
forwards SFS	0.583 ± 0.000	0.551 ± 0.000	0.531 ± 0.000	0.541 ± 0.000	0.593 ± 0.000
HSIC_Lasso	0.575 ± 0.038	0.548 ± 0.051	0.492 ± 0.069	0.516 ± 0.048	0.577 ± 0.035
Lasso_selection	0.521 ± 0.040	0.480 ± 0.048	0.424 ± 0.066	0.449 ± 0.054	0.520 ± 0.022
mRMR	0.504 ± 0.030	0.464 ± 0.030	0.409 ± 0.041	0.432 ± 0.019	0.509 ± 0.015
Permutation	0.566 ± 0.019	0.536 ± 0.021	0.462 ± 0.039	0.495 ± 0.029	0.582 ± 0.037

Table 52: Performance summary for classifier: SVM

Feature Selection	Accuracy	Precision	Sensitivity	F1 Score	AUROC
backward SFS	0.531 ± 0.000	0.493 ± 0.000	0.420 ± 0.000	0.453 ± 0.000	0.538 ± 0.000
forwards SFS	0.571 ± 0.000	0.537 ± 0.000	0.531 ± 0.000	0.534 ± 0.000	0.577 ± 0.000
HSIC_Lasso	0.564 ± 0.023	0.536 ± 0.031	0.437 ± 0.094	0.476 ± 0.056	0.577 ± 0.034
Lasso_selection	0.533 ± 0.025	0.495 ± 0.033	0.374 ± 0.064	0.423 ± 0.048	0.539 ± 0.020
mRMR	0.526 ± 0.027	0.485 ± 0.037	0.347 ± 0.056	0.402 ± 0.043	0.519 ± 0.030
Permutation	0.599 ± 0.038	0.575 ± 0.049	0.519 ± 0.078	0.543 ± 0.055	0.626 ± 0.049

C Appendix C — *Python Code*

C.1 pipeline

```python

import json
import os
import time
import glob
from tqdm import tqdm
from datetime import datetime
from joblib import Parallel, delayed
from classification.src import classifiers as cl
from featureselection.src.feature_selection_methods import *
from Pipeline import load_graph, load_full_corr, train_and_evaluate, cross_validate_model, print_selected_features, failsafe_feature_selection, classify, load_dataframe
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix
from featureselection.src import cluster

# ========== CONFIGURATION ========== #

classifiers_to_run = ["SVM", "RandomForest", "LogR", "LDA", "KNN"]

feature_selection_methods = [
    ("Lasso_selection", Lasso_selection, {"alpha": 0.044984, "max_iter": 2000}, "cv"), #0.044984 for full corr
    ("HSIC_Lasso", hsiclasso, {"num_feat": 19}, "cv"), #98 for full corr
    ("mRMR", mRMR, {"num_features_to_select": 100}, "cv"),
    ("Permutation", Perm_importance, {}, "cv"),
    ("forwards␣SFS", forwards_SFS, {"n_features_to_select": 20}, "train"),
    ("backward␣SFS", backwards_SFS, {"n_features_to_select": 10}, "train")
]

inf_methods = ["partial_corr", "mutual_info", "norm_laplacian", "rlogspect"]
#("ReliefF", reliefF_, {"num_features_to_select": 200}, "cv")
# ========== SAVE RESULTS ========== #

def save_results(classifier, feature_selection_name, inf_method, results_dict):
    os.makedirs(f"results_graph_NYU_male/{classifier}/{inf_method}", exist_ok=True)

    # Generate timestamp: YYYYMMDD-HHMMSS
    timestamp = datetime.now().strftime("%Y%m%d-%H%M%S")

    filename = f"results_graph_NYU_male/{classifier}/{inf_method}/{feature_selection_name}_{timestamp}.json"

        # Helper function to convert numpy types
    def convert(o):
        if isinstance(o, np.integer):
            return int(o)
        if isinstance(o, np.floating):
            return float(o)
        if isinstance(o, np.ndarray):
            return o.tolist()
        return o

    with open(filename, "w") as f:
        json.dump(results_dict, f, indent=4, default=convert)

    print(f"   ␣Saved:␣{filename}")

# ========== MAIN PER CLASSIFIER ========== #

def main_for_classifier(classifier):
    print(f"\n\n==========␣Running␣pipeline␣for␣{classifier}␣==========\n")

    for inf_method in inf_methods:
        print(f"\n---␣Using␣inference␣method:␣{inf_method}␣---")

        # Load data with current inference method
        X, y = load_graph(sex='male', site_id='NYU', method=inf_method)

        # Run raw model just for reference
        X_train, X_test, y_train, y_test = train_and_evaluate(X, y, classifier)

        # Use clustered features for Perm / SFS
        X_clustered = cluster.cluster(X_train, y_train, t=3)
        X_mRMR = mRMR(X_train, y_train, classifier, num_features_to_select=50)

        # Add results for raw data without feature selection
        print(f"\n===␣Running␣Raw␣data␣for␣{classifier}␣===")
        result_raw = {
            "classifier": classifier,
            "feature_selection": "Raw␣data",
            "mode": "cv"
        }

        start_time = time.time()

        # Cross-validation for raw data
        selected_features, selected_feature_names, avg_metrics, fold_metrics = cross_validate_model(
            X, y, None, classifier, n_splits=5, return_metrics=True
        )
        result_raw["selected_features"] = selected_features
        result_raw["selected_feature_names"] = list(selected_feature_names) if selected_feature_names is not None else []
        result_raw["metrics"] = avg_metrics
        result_raw["fold_metrics"] = fold_metrics

        # Save raw data results
        elapsed = time.time() - start_time
        result_raw["elapsed_seconds"] = elapsed
        save_results(classifier, "Raw data", inf_method, result_raw)

        # Loop over feature selection methods with tqdm per classifier
        for fs_name, fs_func, fs_kwargs, mode in tqdm(feature_selection_methods, desc=f"{classifier} pipeline", position=0, leave=True):
            print(f"\n=== Running {fs_name} for {classifier} ===")

            result = {
                "classifier": classifier,
                "feature_selection": fs_name,
                "mode": mode
            }

            start_time = time.time()

            if mode == "cv":
                # Normal cross-validation     capture avg metrics
                selected_features, selected_feature_names, avg_metrics, fold_metrics = cross_validate_model(
                    X, y, fs_func, classifier, n_splits=5, return_metrics=True, **fs_kwargs
                )
                print_selected_features(selected_features, selected_feature_names, print_feat=False)

                result["selected_features"] = selected_features
                result["selected_feature_names"] = list(selected_feature_names) if selected_feature_names is not None else []
                result["metrics"] = avg_metrics
                result["fold_metrics"] = fold_metrics

            elif mode == "train":
                # Single run on training data      classify
                if fs_name == "Permutation":
                    select_features = X_clustered
                elif fs_name == "forwards␣SFS":
                    select_features = X_mRMR
                elif fs_name == "backwards␣SFS":
                    select_features = X_mRMR
                else:
                    select_features = None  # fallback

                selected_features = failsafe_feature_selection(
                    fs_func, X_train, y_train, min_features=20,
                       classifier=classifier, select_features=
                       select_features, **fs_kwargs
                )

                selected_feature_names = classify(
                    X_train, X_test, y_train, y_test,
                       selected_features, classifier, performance=
                       True
                )
                print_selected_features(selected_features,
                   selected_feature_names, print_feat=False)

                # Prepare data
                scaler = StandardScaler()
                X_train_scaled = scaler.fit_transform(X_train)
                X_test_scaled = scaler.transform(X_test)

                X_train_sel = X_train_scaled[:, selected_features
                   ]
                X_test_sel = X_test_scaled[:, selected_features]

                if classifier == "SVM":
                    model = cl.applySVM(X_train_sel, y_train)
                elif classifier == "RandomForest":
                    model = cl.applyRandForest(X_train_sel,
                       y_train)
                elif classifier == "LogR":
                    model = cl.applyLogR(X_train_sel, y_train)
                elif classifier == "LDA":
                    model = cl.applyLDA(X_train_sel, y_train)
                elif classifier == "KNN":
                    model = cl.applyKNN(X_train_sel, y_train)

                y_pred = model.predict(X_test_sel)
                try:
                    y_proba = model.predict_proba(X_test_sel)[:, 1]
                except:
                    y_proba = None

                acc = accuracy_score(y_test, y_pred)
                precision = precision_score(y_test, y_pred)
                recall = recall_score(y_test, y_pred)
                f1 = f1_score(y_test, y_pred)
                try:
                    auc = roc_auc_score(y_test, y_proba) if y_proba is not None else None
                except:
                    auc = None

                cm = confusion_matrix(y_test, y_pred)
                tn, fp, fn, tp = cm.ravel()
                sensitivity = tp / (tp + fn) if (tp + fn) > 0 else 0.0

                result["selected_features"] = selected_features
                result["selected_feature_names"] = list(selected_feature_names) if selected_feature_names is not None else []
                result["metrics"] = {
                    "num feat": len(selected_features),
                    "accuracy": acc,
                    "precision": precision,
                    "recall": recall,
                    "f1_score": f1,
                    "auroc": auc,
                    "sensitivity": sensitivity
                }

            # Save result after each feature selection run
            elapsed = time.time() - start_time
            result["elapsed_seconds"] = elapsed
            save_results(classifier, fs_name, inf_method, result)

        print(f"\n  Finished {classifier} pipeline!\n")

def gather_and_rank_results(result_dir="results_graph_total_multisite", metric="f1_score", top_n=5):
    results = []

    for clf in classifiers_to_run:
        for inf in inf_methods:
            path = os.path.join(result_dir, clf, inf, "*.json")
            for file in glob.glob(path):
                with open(file, "r") as f:
                    data = json.load(f)
                    metrics = data.get("metrics", {})
                    results.append({
                        "classifier": clf,
                        "inf_method": inf,
                        "fs_method": data.get("feature_selection", "Unknown"),
                        "metric_value": metrics.get(metric, 0),
                    })

    df = pd.DataFrame(results)
    top_results = df.sort_values(by="metric_value", ascending=False).head(top_n)
    print("\n ␣Top␣configurations␣based␣on", metric)
    print(top_results)

    return top_results

def print_selected_features_from_top_result(result_dir="results_graph_total_multisite", metric="f1_score"):
    top_results = gather_and_rank_results(result_dir=result_dir, metric=metric, top_n=1)
    if top_results.empty:
        print(" ␣No␣top␣result␣found.")
        return

    top = top_results.iloc[0]
    clf = top["classifier"]
    inf = top["inf_method"]
    fs = top["fs_method"]

    path = os.path.join(result_dir, clf, inf, f"{fs}_*.json")
    best_file = max(glob.glob(path), key=os.path.getctime)  # get the most recent

    with open(best_file, "r") as f:
        data = json.load(f)
        selected_feature_names = data.get("selected_feature_names", [])

    print(f"\n ␣Best␣configuration:␣{clf}␣+␣{inf}␣+␣{fs}")
    print(f" ␣Loaded␣from:␣{best_file}")
    print(f" ␣Selected␣Features␣({len(selected_feature_names)}):")
    for feat in selected_feature_names:
        print(f"␣-␣{feat}")


# ========== PARALLEL RUNNER ========== #

if __name__ == "__main__":
    print(" ␣Starting␣parallel␣pipeline...")

    # Run all classifiers in parallel
    Parallel(n_jobs=len(classifiers_to_run))(
        delayed(main_for_classifier)(clf) for clf in
           classifiers_to_run
    )

    print("\   n     ␣All␣classifiers␣completed!")

    #top_configs = gather_and_rank_results()
    print_selected_features_from_top_result()

    print("\   n     ␣Program␣finished")
```

Listing 1: parallel_main

```

from sklearn.metrics import classification_report,
   confusion_matrix
import numpy as np
import pandas as pd
import os
from scipy import stats
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, KFold
from sklearn.metrics import mean_squared_error, accuracy_score,
   recall_score, precision_score, f1_score, roc_auc_score,
   confusion_matrix, ConfusionMatrixDisplay, roc_curve
from classification.src import classifiers as cl,
   basicfeatureextraction
from featureselection.src.feature_selection_methods import *
from featureselection.src import cluster
from featureselection.src import Compute_HSIC_Lasso as hsic_lasso
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, RobustScaler
from featuredesign.graph_inference.AAL_test import multiset_feats
   , load_files, adjacency_df
import glob
import cvxpy as cp
import seaborn as sns

def load_file(sex='all', method='pearson_corr', alpha=5):
    #folder_path = r"C:\Users\guus\Python_map\AutismDetection-
       main\abide\female-cpac-filtnoglobal-aal" # Enter your local
        ABIDE dataset path
    fmri_data, subject_ids, _, _ = load_files(sex=sex, max_files
       =800, site="NYU", shuffle=True, var_filt=True, ica=True)

    print(f"Final␣data:␣{len(fmri_data)}␣subjects")
    print(f"Final␣IDs:␣{len(subject_ids)}")

    full_df = adjacency_df(fmri_data, subject_ids, method =
```

```python
        method, alpha = alpha)
    print("Merged feature+label shape:\n", full_df.shape)

    #print(full_df)

    subject_id_to_plot = '0051044'  # Change this to any valid subject ID
    #plot_adjacency_matrix(full_df, subject_id_to_plot)

    full_df = full_df.sample(frac=1, random_state=42).reset_index(drop=True)  # Shuffle the DataFrame

    X = full_df.drop(columns=['DX_GROUP', 'subject_id', 'SEX'])
    y = full_df['DX_GROUP'].map({1: 1, 2: 0}) #1 ASD, 0 ALL

    # Making sure the data is numeric
    X = X.apply(pd.to_numeric, errors='coerce')
    X = X.dropna(axis=1,how='all')
    non_nan_ratio = X.notna().mean()
    X = X.loc[:, non_nan_ratio > 0.8]  # Keep columns with more than 50% non-NaN values
    # Making sure there is no 0 var data for the hsic algorithm
    X = X.loc[:, X.var() > 1e-6]

    # NaN values are filled with the median of the column
    X= X.fillna(X.median())

    return X, y

def load_graph_csv(method, sex='all', site_id=None):
    # Load data
    if method == 'laplacian':
        data = pd.read_csv('cpac_rois-aal_nogsr_filt_norm-laplacian_direct_20ICA_alpha0.0001_thr0.25.csv', encoding='ISO-8859-1')
    elif method == 'rspect':
        data = pd.read_csv('cpac_rois-aal_nogsr_filt_rspect_direct_20ICA_alpha0.0001_thr0.10.csv', encoding='ISO-8859-1')
    else:
        print("use laplacian or rspect as method")
    data = data[data['DX_GROUP'].notna()]

    # Separate by sex
    fc_female = data[data['SEX'] == 2]
    fc_male = data[data['SEX'] == 1]

    if sex == 'female':
        fc = fc_female
    elif sex == 'male':
        fc = fc_male
    elif sex == 'all':
        fc = pd.concat([fc_female, fc_male], axis=0, ignore_index
           =True)
    else:
        print("Use␣male,␣female␣or␣all␣as␣sex")

    if site_id is not None:
        fc = fc[fc['SITE_ID'] == site_id]

    fc = fc.sample(frac=1, random_state=42).reset_index(drop=True
       )  # Shuffle the DataFrame
    fc = fc.dropna(subset=['DX_GROUP'])

    X = fc.drop(columns=['DX_GROUP', 'SEX', 'SITE_ID', '
       subject_id', 'AGE_AT_SCAN'])
    y = fc['DX_GROUP']

    # Making sure the data is numeric
    X = X.apply(pd.to_numeric, errors='coerce')
    X = X.dropna(axis=1,how='all')
    non_nan_ratio = X.notna().mean()
    X = X.loc[:, non_nan_ratio > 0.8]  # Keep columns with more
       than 50% non-NaN values
    # Making sure there is no 0 var data for the hsic algorithm
    X = X.loc[:, X.var() > 1e-4]
    # NaN values are filled with the median of the column
    X = X.fillna(X.median())

    # Remove extremely correlated features
    X = correlation_filter(X, threshold=0.9)
    print(f"After␣outlier␣removal:␣{X.shape}")
    # Remove extreme outliers
    X = remove_extreme_outliers(X, threshold=3.5)
    #print(f"After outlier removal: {X.shape}")

    # Apply feature transformations for better distributions
    X = apply_feature_transformations(X)
    #print(f"After transformation: {X.shape}")

    # Site effect correction
    if 'SITE_ID' in fc.columns:
        X = correct_site_effects(X, fc['SITE_ID'])
        #print(f"After site correction: {X.shape}")

    #print(f"X: {X}, y: {y}")

    return X, y

def load_graph(sex='all', site_id=None, method="norm_laplacian",
   cov="ledoit"):

    fmri_data_f, subject_ids_f, _, _ = load_files(sex='Female', max_files=800, site=site_id, shuffle=True, var_filt=True, ica=True)
    fmri_data_m, subject_ids_m, _, _ = load_files(sex='Male', max_files=800, site=site_id, shuffle=True, var_filt=True, ica=True)

    fc_female = multiset_feats(fmri_data_f, subject_ids_f, inf_method=method, cov_method=cov,
                   thresh=0.1, n_jobs=-1, feats="graph")
    fc_male = multiset_feats(fmri_data_m, subject_ids_m, inf_method=method, cov_method=cov,
                   thresh=0.1, n_jobs=-1, feats="graph")
    if sex == 'female':
        fc = fc_female
    elif sex == 'male':
        fc = fc_male
    elif sex == 'all':
        fc = pd.concat([fc_female, fc_male], axis=0, ignore_index=True)
    else:
        print("Use male, female or all as sex")

    if site_id is not None:
        fc = fc[fc['SITE_ID'] == site_id]

    fc = fc.sample(frac=1, random_state=42).reset_index(drop=True)  # Shuffle the DataFrame
    fc = fc.dropna(subset=['DX_GROUP'])

    X = fc.drop(columns=['DX_GROUP', 'SEX', 'SITE_ID', 'subject_id'])
    y = fc['DX_GROUP']

    if X.empty or X.shape[1] < 10:
        raise ValueError("    Not enough usable features extracted from multiset_feats")

    #X.to_csv('laplacian_prefilter_ledoit.csv', index=False)

    # Making sure the data is numeric
    X = X.apply(pd.to_numeric, errors='coerce')
    X = X.dropna(axis=1,how='all')
    non_nan_ratio = X.notna().mean()
    X = X.loc[:, non_nan_ratio > 0.8]  # Keep columns with more than 50% non-NaN values
    # Making sure there is no 0 var data for the hsic algorithm
    X = X.loc[:, X.var() > 1e-4]
    # NaN values are filled with the median of the column
    X = X.fillna(X.median())

```

```python
    # Remove extremely correlated features
    X = correlation_filter(X, threshold=0.9)
    print(f"After outlier removal: {X.shape}")
    # Remove extreme outliers
    X = remove_extreme_outliers(X, threshold=3.5)
    #print(f"After outlier removal: {X.shape}")

    # Apply feature transformations for better distributions
    X = apply_feature_transformations(X)
    #print(f"After transformation: {X.shape}")

    # Site effect correction
    if 'SITE_ID' in fc.columns:
        X = correct_site_effects(X, fc['SITE_ID'])
        #print(f"After site correction: {X.shape}")

    #X.to_csv('laplacian_ledoit.csv', index=False)

    return X, y

def load_full_corr(sex='all', site_id=None):

    fc_female = basicfeatureextraction.extract_fc_features("abide/female-cpac-filtnoglobal-aal", "abide/Phenotypic_V1_0b_preprocessed1.csv")
    fc_male = basicfeatureextraction.extract_fc_features("abide/male-cpac-filtnoglobal-aal", "abide/Phenotypic_V1_0b_preprocessed1.csv")
    if sex == 'female':
        fc = fc_female
    elif sex == 'male':
        fc = fc_male
    elif sex == 'all':
        fc = pd.concat([fc_female, fc_male], axis=0, ignore_index=True)
    else:
        print("Use male, female or all as sex")

    if site_id is not None:
        fc = fc[fc['SITE_ID'] == site_id]

    fc = fc.sample(frac=1, random_state=42).reset_index(drop=True)  # Shuffle the DataFrame
    fc = fc.dropna(subset=['DX_GROUP'])

    X = fc.drop(columns=['DX_GROUP', 'SEX', 'SITE_ID', 'subject_id', 'AGE'])
    y = fc['DX_GROUP']

    # Making sure the data is numeric
    X = X.apply(pd.to_numeric, errors='coerce')
    X = X.dropna(axis=1,how='all')
    non_nan_ratio = X.notna().mean()
    X = X.loc[:, non_nan_ratio > 0.8]  # Keep columns with more
       than 50% non-NaN values
    # Making sure there is no 0 var data for the hsic algorithm
    X = X.loc[:, X.var() > 1e-4]
    # NaN values are filled with the median of the column
    X = X.fillna(X.median())

    return X, y

def load_dataframe(path='multi'):
    if path =='uni':
        folder_path = 'Feature_Dataframes/first_run'
    if path == 'multi':
        folder_path = 'Feature_Dataframes/second_run'
        #file_name = 'cpac_rois-
           aal_nogsr_filt_LADMM_direct_20ICA_graph_thr0.3.csv'
        file_name = 'cpac_rois-
           aal_nogsr_filt_LADMM_var_20ICA_graph_thr0.3.csv'
        #file_name = 'cpac_rois-aal_nogsr_filt_norm-
           laplacian_direct_20ICA_graph_thr0.3.csv'
        #file_name = 'cpac_rois-aal_nogsr_filt_norm-
           laplacian_glasso_20ICA_graph_thr0.3.csv'
        #file_name = 'cpac_rois-aal_nogsr_filt_norm-
           laplacian_ledoit_20ICA_graph_thr0.3.csv'
        #file_name = 'cpac_rois-aal_nogsr_filt_norm-
           laplacian_var_20ICA_graph_thr0.3.csv'

    file_path = os.path.join(folder_path, file_name)
    fc = pd.read_csv(file_path)
    #fc = pd.concat([pd.read_csv(file) for file in glob.glob(os.
       path.join(folder_path, '*.csv'))], ignore_index=True)

    fc = fc.sample(frac=1, random_state=42).reset_index(drop=True
       )  # Shuffle the DataFrame
    #fc = fc.dropna(subset=['DX_GROUP'])

    X = fc.drop(columns=['DX_GROUP', 'SEX', 'SITE_ID', '
       subject_id', 'AGE_AT_SCAN'])
    y = fc['DX_GROUP']

    # Making sure the data is numeric
    X = X.apply(pd.to_numeric, errors='coerce')
    X = X.dropna(axis=1,how='all')
    non_nan_ratio = X.notna().mean()
    X = X.loc[:, non_nan_ratio > 0.8]  # Keep columns with more
       than 50% non-NaN values
    # Making sure there is no 0 var data for the hsic algorithm
    X = X.loc[:, X.var() > 1e-4]
    # NaN values are filled with the median of the column
```

```python
    X = X.fillna(X.median())
    print(f"shape dataframe: {X.shape}")

    print(f"X: {X}, y: {y}")

    return X, y

def evaluate_performance(y_true, y_pred, y_proba=None, show_plots=False, classifier_name="", fold_idx=None, verbose=True):
    # Compute basic metrics
    acc = accuracy_score(y_true, y_pred)
    prec = precision_score(y_true, y_pred)
    rec = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)
    auc = roc_auc_score(y_true, y_proba) if y_proba is not None else None

    if verbose==True:
        print(f"\nPerformance Metrics ({classifier_name}):")
        print(f"Performance Metrics ({classifier_name}):")
        print(f"  Accuracy:  {acc:.4f}")
        print(f"  Precision: {prec:.4f}")
        print(f"  Recall:    {rec:.4f}")
        print(f"  F1 Score:  {f1:.4f}")
        if auc is not None:
            print(f"  AUC:       {auc:.4f}")

    if show_plots:
        # Confusion matrix
        cm = confusion_matrix(y_true, y_pred)
        disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Class 0", "Class 1"])
        disp.plot(cmap="Blues")
        plt.title(f"Confusion Matrix - {classifier_name}")
        plt.show()

        # ROC curve (if proba is available)
        if y_proba is not None:
            fpr, tpr, _ = roc_curve(y_true, y_proba)
            plt.plot(fpr, tpr, label=f"AUC = {auc:.2f}")
            plt.plot([0, 1], [0, 1], 'k--')
            plt.xlabel("False Positive Rate")
            plt.ylabel("True Positive Rate")
            plt.title(f"ROC Curve - {classifier_name}")
            plt.legend()
            plt.grid()
            plt.show()

    return {
        "accuracy": acc,
        "precision": prec,
        "recall": rec,
        "f1": f1,
        "auc": auc
    }

def print_selected_features(selected_features, selected_feature_names, print_feat=False):
    num_feat = len(selected_features)
    print(f"Selected␣features␣({num_feat}):", selected_features)
    #if print_feat==True:
    #    print(f"\nSelected feature names({len(selected_feature_names)}):")
    #    for name in selected_feature_names:
    #        print("-", name)

def train_and_evaluate(X, y, classifier):
    #splitting the data in train and test 0.8:0.2 respecively
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

    print(f"y_train:␣{y_train},␣y_test:␣{y_test}")

    #scale the data for the classifier
    scaler = RobustScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    if classifier == "SVM":
        model_raw = cl.applySVM(X_train_scaled, y_train)
    elif classifier == "RandomForest":
        model_raw = cl.applyRandForest(X_train_scaled, y_train)
    elif classifier == "LogR":
        model_raw = cl.applyLogR(X_train_scaled, y_train)
    elif classifier == "LDA":
        model_raw = cl.applyLDA(X_train_scaled, y_train)
    elif classifier == "KNN":
        model_raw = cl.applyKNN(X_train_scaled, y_train)
    else:
        print("Classifier␣not␣supported:␣choose␣from␣SVM,␣RandomForest,␣LogR,␣LDA␣or␣KNN")
    #applying the classifier to the total data
    model_raw = cl.applySVM(X_train, y_train)
    y_pred_raw = model_raw.predict(X_test)

    try:
        y_proba_raw = model_raw.predict_proba(X_test_scaled)[:, 1]
    except:
        y_proba_raw = None

    #finding mse and accuracy
    perf_raw = evaluate_performance(y_test, y_pred_raw, y_proba_raw, classifier_name=classifier, verbose=False)
    acc_raw = perf_raw["accuracy"]
    mse_raw = mean_squared_error(y_test, y_pred_raw)
    precision_raw = perf_raw["precision"]
    recall_raw = perf_raw["recall"]
    F1_raw = perf_raw["f1"]
    AUC_raw = perf_raw["auc"]
    #print(classification_report(y_test, y_pred_raw, target_names=["Class 0", "Class 1"]))
    #print('Confusion matrix:', confusion_matrix(y_test, y_pred_raw))
    #print('Amount of features:', X_train.shape[1])

    #acc, mse, selected_feature_names = cross_validate_model(X, y, selected_features)
    print(f"Train/Test␣Accuracy␣raw:␣{acc_raw:.4f},␣MSE:␣{mse_raw:.4f},␣Precision:␣{precision_raw:.4f},␣Recall:␣{recall_raw:.4f},␣F1:␣{F1_raw:.4f},␣AUC:␣{AUC_raw:.4f}")

    return X_train, X_test, y_train, y_test

def classify(X_train, X_test, y_train, y_test, selected_features, classifier, performance=True):

    #scale the data for the classifier
    scaler = StandardScaler()
    X_train_scaled = X_train #scaler.fit_transform(X_train)
    X_test_scaled = X_test #scaler.transform(X_test)

    if isinstance(X_train_scaled, pd.DataFrame):
        # If it's a DataFrame, use `.iloc[]` for indexing
        selected_train_x = X_train_scaled.iloc[:, selected_features]
        selected_test_x = X_test_scaled.iloc[:, selected_features]
    else:
        # If it's a numpy array, use standard array indexing
        selected_train_x = X_train_scaled[:, selected_features]
        selected_test_x = X_test_scaled[:, selected_features]

    if classifier == "SVM":
        model = cl.applySVM(selected_train_x, y_train)
    elif classifier == "RandomForest":
        model = cl.applyRandForest(selected_train_x, y_train)
    elif classifier == "LogR":
        model = cl.applyLogR(selected_train_x, y_train)
    elif classifier == "LDA":
        model = cl.applyLDA(selected_train_x, y_train)
    elif classifier == "KNN":
        model = cl.applyKNN(selected_train_x, y_train)
    else:
        print("Classifier␣not␣supported:␣choose␣from␣SVM,␣RandomForest,␣LogR,␣DecisionTree␣or␣MLP")

    #applying the classifier to the selected data
    y_pred = model.predict(selected_test_x)
    #params=bestSVM_RS(X_train, X_test, y_train, y_test, svcdefault=SVC())
    #finding mse and accuracy

    # Predict probabilities if supported
    try:
        y_proba = model.predict(selected_test_x) if hasattr(model, "predict_proba") else None
        if y_proba is not None:
            y_proba = model.predict_proba(selected_test_x)[:, 1]
    except:
        y_proba = None
    if performance==True:
        evaluate_performance(y_test, y_pred, y_proba, classifier_name=classifier)
    #getting and printing the feature names
    feature_names = X_train.columns
    selected_feature_names = feature_names[selected_features]

    return selected_feature_names

def cross_validate_model(X, y, feature_selection, classifier, raw=True, return_metrics=False, n_splits=5, **feature_selection_kwargs):
    #K-Fold cross-validation evaluation.
    kf = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=42) #shuffle=True, random_state=42
    acc_scores = []
    mse_scores = []
    precision_scores = []
    recall_scores = []
    F1_scores = []
    AUC_scores = []
    acc_scores_raw = []
    mse_scores_raw = []
    precision_scores_raw = []
    recall_scores_raw = []
    F1_scores_raw = []
    AUC_scores_raw= []
    fold_metrics = []

    if classifier is not Perm_importance or backwards_SFS:
        # Convert inputs to numpy arrays once at the beginning
        if isinstance(X, pd.DataFrame):
            feature_names = X.columns
```

```python
            X = X.to_numpy()
        elif isinstance(X, pd.Series):
            feature_names = [f"feature_{i}" for i in range(len(X))]
            X = X.to_numpy()
        else:
            feature_names = [f"feature_{i}" for i in range(X.shape[1])]

        X = np.asarray(X, dtype=np.float64)

        # Inside failsafe_feature_selection
        if isinstance(y, pd.Series) or isinstance(y, pd.DataFrame):
            y = y.values
        y = np.asarray(y, dtype=np.float64).reshape(-1)

    selected_features = None
    selected_feature_names = None

    for train_idx, test_idx in kf.split(X, y):

        X_train, X_test = X[train_idx], X[test_idx]
        y_train, y_test = y[train_idx], y[test_idx]

        #Scaling the data
        scaler = RobustScaler()
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)

        if feature_selection is not None:
            selected_features = failsafe_feature_selection(feature_selection, X_train_scaled, y_train, classifier=classifier, **feature_selection_kwargs)

            # Ensure selected_features is a list of valid indices
            if not isinstance(selected_features, (list, np.ndarray)):
                selected_features = [selected_features] if selected_features is not None else []

            selected_features = [int(idx) for idx in selected_features if isinstance(idx, (int, np.integer)) and 0 <= idx < X_train.shape[1]]

            if not selected_features:
                # Fallback to all features if selection fails
                selected_features = list(range(X_train.shape[1]))

            # Select the features based on the selected indices
            X_train_sel = X_train_scaled[:, selected_features]
            X_test_sel = X_test_scaled[:, selected_features]
        else:
            X_train_sel = X_train_scaled
            X_test_sel = X_test_scaled

        #applying the classifier
        if classifier == "SVM":
            model = cl.applySVM(X_train_sel, y_train)
            model_raw = cl.applySVM(X_train_scaled, y_train)
        elif classifier == "RandomForest":
            model = cl.applyRandForest(X_train_sel, y_train)
            model_raw = cl.applyRandForest(X_train_scaled, y_train)
        elif classifier == "LogR":
            model = cl.applyLogR(X_train_sel, y_train)
            model_raw = cl.applyLogR(X_train_scaled, y_train)
        elif classifier == "DT":
            model = cl.applyDT(X_train_sel, y_train)
            model_raw = cl.applyDT(X_train_scaled, y_train)
        elif classifier == "MLP":
            model = cl.applyMLP(X_train_sel, y_train)
            model_raw = cl.applyMLP(X_train_scaled, y_train)
        elif classifier == "LDA":
            model = cl.applyLDA(X_train_sel, y_train)
            model_raw = cl.applyLDA(X_train_scaled, y_train)
        elif classifier == "KNN":
            model = cl.applyKNN(X_train_sel, y_train)
            model_raw = cl.applyKNN(X_train_scaled, y_train)

        y_pred = model.predict(X_test_sel)
        y_pred_raw = model_raw.predict(X_test_scaled)

        try:
            y_proba = model.predict_proba(X_test_sel)[:, 1]
        except:
            y_proba = None

        try:
            y_proba_raw = model.predict_proba(X_test_scaled)[:, 1]
        except:
            y_proba_raw = None

        perf = evaluate_performance(y_test, y_pred, y_proba, classifier_name=classifier, fold_idx=len(acc_scores) + 1, verbose=False)

        acc_scores.append(perf["accuracy"] if perf["accuracy"] is not None else 0.0)
        mse_scores.append(mean_squared_error(y_test, y_pred))
        precision_scores.append(perf["precision"] if perf["
precision"] is not None else 0.0)
        recall_scores.append(perf["recall"] if perf["recall"] is not None else 0.0)
        F1_scores.append(perf["f1"] if perf["f1"] is not None else 0.0)
        AUC_scores.append(perf["auc"] if perf["auc"] is not None else 0.0)

        fold_metrics.append({
            "accuracy": acc_scores,
            "precision": precision_scores,
            "recall": recall_scores,
            "f1_score": F1_scores,
            "auroc": AUC_scores
        })

        print(classifier)
        #print(classification_report(y_test, y_pred, target_names=["Class 0", "Class 1"]))
        #print('Confusion matrix:', confusion_matrix(y_test, y_pred))

        if raw==True:
            perf_raw = evaluate_performance(y_test, y_pred_raw, y_proba_raw, classifier_name=classifier, fold_idx=len(acc_scores) + 1, verbose=False)
            # Raw performance
            acc_scores_raw.append(perf_raw["accuracy"])
            mse_scores_raw.append(mean_squared_error(y_test, y_pred))
            precision_scores_raw.append(perf_raw["precision"])
            recall_scores_raw.append(perf_raw["recall"])
            F1_scores_raw.append(perf_raw["f1"])
            AUC_scores_raw.append(perf_raw["auc"])

            avg_acc_raw = np.mean(acc_scores_raw)
            avg_mse_raw = np.mean(mse_scores_raw)
            avg_precision_raw = np.mean(precision_scores_raw)
            avg_recall_raw = np.mean(recall_scores_raw)
            avg_F1_raw = np.mean(F1_scores_raw)
            avg_AUC_raw = np.mean([score for score in AUC_scores_raw if score is not None])

            print(f"Average␣accuracy␣raw:␣{avg_acc_raw}")
            print(f"Average␣mse␣raw:␣{avg_mse_raw}")
            print(f"Average␣precision␣raw:␣{avg_precision_raw}")
            print(f"Average␣recall␣raw:␣{avg_recall_raw}")
            print(f"Average␣F1␣raw:␣{avg_F1_raw}")
            print(f"Average␣AUC␣raw:␣{avg_AUC_raw}")

    # Calculate averages (only if we have results)
    if acc_scores:
        # Get feature names for the last fold's selection
        if selected_features is not None:
            selected_feature_names = [feature_names[i] for i in selected_features
                                    if i < len(feature_names)]
        else:
            selected_feature_names = list(feature_names)
    """
    selected_features = failsafe_feature_selection(feature_selection, X, y, classifier=classifier, **feature_selection_kwargs)
    X_selected = X[:, selected_features]

    model = select_model(classifier)

    acc_scores = cross_val_score(model, X_selected, y, cv=kf, scoring='accuracy')
    mse_scores = cross_val_score(model, X_selected, y, cv=kf, scoring='neg_mean_squared_error')
    precision_scores = cross_val_score(model, X_selected, y, cv=kf, scoring='precision')
    recall_scores = cross_val_score(model, X_selected, y, cv=kf, scoring='recall')
    F1_scores = cross_val_score(model, X_selected, y, cv=kf, scoring='f1')
    AUC_scores = cross_val_score(model, X_selected, y, cv=kf, scoring='roc_auc')
    """

    avg_acc = np.mean(acc_scores)
    avg_mse = np.mean(mse_scores)
    avg_precision = np.mean(precision_scores)
    avg_recall = np.mean(recall_scores)
    avg_F1 = np.mean(F1_scores)
    avg_AUC = np.mean(AUC_scores)

    avg_metrics = {
        "accuracy": avg_acc,
        "precision": avg_precision,
        "recall": avg_recall,
        "f1_score": avg_F1,
        "auroc": avg_AUC,
        "sensitivity": avg_recall  # sensitivity == recall in binary classification

    }

    print(f"\nMean performance Metrics ({classifier}), ({feature_selection}):")
    print(f"Mean performance Metrics ({classifier}), ({
       feature_selection}):")
    print(f"  Accuracy:  {avg_acc:.4f}")
    print(f"  Precision: {avg_precision:.4f}")
    print(f"  Recall:    {avg_recall:.4f}")
    print(f"  F1 Score:  {avg_F1:.4f}")
    if avg_AUC is not None:
        print(f"  AUC:       {avg_AUC:.4f}")

    if raw==True:
        acc_scores_raw = cross_val_score(model, X, y, cv=kf,
           scoring='accuracy')
        mse_scores_raw = cross_val_score(model, X, y, cv=kf,
           scoring='neg_mean_squared_error')
        precision_scores_raw = cross_val_score(model, X, y, cv=kf
           , scoring='precision')
        recall_scores_raw = cross_val_score(model, X, y, cv=kf,
           scoring='recall')
        F1_scores_raw = cross_val_score(model, X, y, cv=kf,
           scoring='f1')
        AUC_scores_raw = cross_val_score(model, X, y, cv=kf,
           scoring='roc_auc')

        avg_acc_raw = np.mean(acc_scores_raw)
        avg_mse_raw = np.mean(mse_scores_raw)
        avg_precision_raw = np.mean(precision_scores_raw)
        avg_recall_raw = np.mean(recall_scores_raw)
        avg_F1_raw = np.mean(F1_scores_raw)
        avg_AUC_raw = np.mean(AUC_scores_raw)

        print(f"\nPerformance Metrics raw ({classifier}):")
        print(f"Performance Metrics raw ({classifier}):")
        print(f"  Accuracy:  {avg_acc_raw:.4f}")
        print(f"  Precision: {avg_precision_raw:.4f}")
        print(f"  Recall:    {avg_recall_raw:.4f}")
        print(f"  F1 Score:  {avg_F1_raw:.4f}")
        if avg_AUC is not None:
            print(f"  AUC:       {avg_AUC_raw:.4f}")

    if return_metrics:
        return selected_features, selected_feature_names,
           avg_metrics, fold_metrics
    else:
        selected_features, selected_feature_names

def select_model(classifier):
    # Determine the model based on the classifier name
    if classifier == "SVM":
        model = SVC(kernel='linear')
    elif classifier == "RandomForest":
        model = RandomForestClassifier(random_state=42)
    elif classifier == "LogR":
        model = LogisticRegression(random_state=42)
    elif classifier == "DT":
        model = DecisionTreeClassifier(random_state=42)
    elif classifier == "MLP":
        model = MLPClassifier(random_state=42)
    elif classifier == "LDA":
        model = LinearDiscriminantAnalysis()
    elif classifier == "KNN":
        model = KNeighborsClassifier()
    else:
        raise ValueError("Unsupported␣classifier␣type,␣choose␣SVM,␣RandomForest,␣DT,␣MLP,␣LogR,␣LDA␣or␣KNN")

    return model
```

Listing 2: Pipeline

C.2 Feature selection methods

```

from __future__ import division
import numpy as np
import pandas as pd
from sklearn.pipeline import Pipeline
from sklearn.linear_model import Lasso, LassoLars
from sklearn.inspection import permutation_importance
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression, Lasso, LassoCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler, KBinsDiscretizer, LabelEncoder
from sklearn.metrics import mutual_info_score
from sklearn.feature_selection import RFE, SequentialFeatureSelector, VarianceThreshold, mutual_info_classif, SelectKBest, f_classif, SelectFromModel
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score, StratifiedKFold, train_test_split
from skfeature.function.information_theoretical_based import MRMR
from scipy.stats import gamma
from pyHSICLasso import HSICLasso
import time
import warnings
import inspect

```

```python
def failsafe_feature_selection(selection_func, X, y, min_features=10, fallback_method='mutual_info', **kwargs):
    """
    Failsafe wrapper for feature selection methods that ensures a minimum number of features are returned.

    Parameters:
    - selection_func: The feature selection function to call
    - X: Input feature matrix (pandas DataFrame or numpy array)
    - y: Target labels (pandas Series or numpy array)
    - min_features: Minimum number of features to return (default: 10)
    - fallback_method: Method to use if primary selection returns insufficient features
                       Options: 'mutual_info', 'f_score', 'random_forest', 'top_variance'
    - **kwargs: Additional arguments to pass to the selection function

    Returns:
    - selected_features: List of selected feature indices
    """

    # Ensure we have enough features to select from
    n_total_features = X.shape[1]
    min_features = min(min_features, n_total_features)

    selected_features = []

    try:
        # Try the primary selection method
        print(f"Attempting␣primary␣feature␣selection␣method...")
        valid_kwargs = _filter_kwargs_for_function(selection_func, kwargs)
        selected_features = selection_func(X, y, **valid_kwargs)

        # Handle different return types
        if hasattr(selected_features, '__iter__') and not isinstance(selected_features, str):
            selected_features = list(selected_features)
        else:
            selected_features = [selected_features] if selected_features is not None else []

        # Remove any invalid indices
        selected_features = [idx for idx in selected_features
                             if isinstance(idx, (int, np.integer)) and 0 <= idx < n_total_features]

        print(f"Primary␣method␣returned␣{len(selected_features)}␣features")

    except Exception as e:
        print(f"Primary␣feature␣selection␣failed:␣{str(e)}")
        selected_features = []

    # Check if we have enough features
    if len(selected_features) < min_features:
        print(f"Insufficient␣features␣from␣primary␣method␣({len(selected_features)}).␣Using␣fallback...")

        # Apply fallback feature selection
        fallback_features = _apply_fallback_selection(X, y, min_features, fallback_method)

        # Combine primary and fallback features (remove duplicates)
        all_features = list(set(selected_features + fallback_features))

        # If still not enough, add top variance features
        if len(all_features) < min_features:
            variance_features = _get_top_variance_features(X, min_features - len(all_features))
            all_features = list(set(all_features + variance_features))

        selected_features = all_features[:min_features]

    # Final safety check - ensure we have valid indices
    selected_features = [idx for idx in selected_features
                         if isinstance(idx, (int, np.integer)) and 0 <= idx < n_total_features]

    # If still empty, return first min_features indices
    if not selected_features:
        print("All␣methods␣failed.␣Returning␣first␣features␣as␣last␣resort.")
        selected_features = list(range(min(min_features, n_total_features)))

    print(f"Final␣selection:␣{len(selected_features)}␣features")
    return selected_features

def _apply_fallback_selection(X, y, min_features, method):
    """Apply fallback feature selection method."""

    try:
        if method == 'mutual_info':
            # Use mutual information
            selector = SelectKBest(score_func=mutual_info_classif, k=min_features)
            selector.fit(X, y)
            return selector.get_support(indices=True).tolist()

        elif method == 'f_score':
            # Use F-score
            selector = SelectKBest(score_func=f_classif, k=min_features)
            selector.fit(X, y)
            return selector.get_support(indices=True).tolist()

        elif method == 'random_forest':
            # Use Random Forest feature importance
            rf = RandomForestClassifier(n_estimators=100, random_state=42)
            rf.fit(X, y)
            importances = rf.feature_importances_
            indices = np.argsort(importances)[::-1]
            return indices[:min_features].tolist()

        elif method == 'top_variance':
            return _get_top_variance_features(X, min_features)

    except Exception as e:
        print(f"Fallback method {method} failed: {str(e)}")

    # If fallback fails, return top variance features
    return _get_top_variance_features(X, min_features)

def _filter_kwargs_for_function(func, kwargs):
    """Filter kwargs to only include parameters that the function accepts."""
    try:
        # Get function signature
        sig = inspect.signature(func)
        valid_params = set(sig.parameters.keys())

        # Filter kwargs to only include valid parameters
        filtered_kwargs = {k: v for k, v in kwargs.items() if k in valid_params}
        return filtered_kwargs
    except Exception:
        # If we can't inspect the function, return empty dict to be safe
        return {}

def _get_top_variance_features(X, min_features):
    """Get features with highest variance as last resort."""
    try:
        if isinstance(X, pd.DataFrame):
            variances = X.var()
        else:
                variances = np.var(X, axis=0)

            indices = np.argsort(variances)[::-1]
            return indices[:min_features].tolist()
        except:
            # Ultimate fallback - return first features
            return list(range(min(min_features, X.shape[1])))

def hsiclasso(X, y, classifier, num_feat=None, feature_range=(1,
   50), verbose=False):
    """
    Perform HSIC Lasso feature selection.
    Parameters:
    - X: Input feature matrix (numpy array or pandas DataFrame).
    - y: Target labels (numpy array or pandas Series).
    - alpha: Regularization strength.
    - max_iter: Maximum number of iterations for convergence.
    - tol: Tolerance for convergence.
    Returns:
    - Selected feature indices.
    """

    original_X = X
    # Ensure X is a numpy array for HSICLasso
    if isinstance(X, pd.DataFrame):
        X = X.values
    # Ensure y is a 1D numpy array
    if isinstance(y, (pd.Series, pd.DataFrame)):
        y = y.values.ravel()
    else:
        y = np.ravel(y)
    if verbose==True:
        print(f"Final shapes - X: {X.shape}, y: {y.shape}")

    if num_feat is not None:
        return perform_HSICLasso(X, y, num_feat, original_X)

    min_feat, max_feat = feature_range
    max_feat = min(max_feat, X.shape[1]) #Don't exceed available
        features

    best_score = -1
    best_features = None
    best_num_feat = min_feat

    model = select_model(classifier)
    cv = StratifiedKFold(n_splits=5, shuffle=True, random_state
        =42)

    print(f"Testing feature counts from {min_feat} to {max_feat
        }...")
```

```python

    for test_num_feat in range(min_feat, max_feat + 1):
        try:
            selected_features = perform_HSICLasso(X, y, test_num_feat, original_X)

            if len(selected_features) == 0:
                continue

            X_selected = X[:, selected_features]
            scores = cross_val_score(model, X_selected, y, cv=cv, scoring='accuracy')
            mean_score = np.mean(scores)

            if verbose==True:
                print(f"Features {test_num_feat}: {mean_score:.4f} +- {np.std(scores):.4f}")
                if len(selected_features) > 0:
                    if hasattr(original_X, 'columns'):
                        # Print feature names if DataFrame
                        feature_names = [original_X.columns[i] for i in selected_features]
                        print(f"  Selected features: {selected_features}")
                        print(f"  Feature names: {feature_names}")
                    else:
                        print(f"  Selected features: {selected_features}")
                else:
                    print(f"  No features selected")

            if mean_score > best_score:
                best_score = mean_score
                best_features = selected_features
                best_num_feat = test_num_feat

        except Exception as e:
            print(f"Error with {test_num_feat} features: {e}")
            continue

    print(f"\nBest: {best_num_feat} features with score { best_score:.4f}")
    if best_features is not None and len(best_features) > 0:
        if hasattr(original_X, 'columns'):
            best_feature_names = [original_X.columns[i] for i in best_features]
            print(f"Final selected feature indices: {best_features}")
            print(f"Final selected feature names: {best_feature_names}")
        else:
            print(f"Final␣selected␣feature␣indices:␣{best_features}")
    else:
        print("No␣features␣were␣successfully␣selected")

    return best_features

def perform_HSICLasso(X, y, num_feat, original_X):
    # Perform HSIC Lasso to select features
    hsic_lasso = HSICLasso()
    # Set parameters for HSIC Lasso

    # Fit the model
    hsic_lasso.input(X, y)
    hsic_lasso.classification(num_feat)

    selected_features = hsic_lasso.get_features()

         # Convert string indices to integers if necessary
    if len(selected_features) > 0 and isinstance(selected_features[0], str):
        try:
            selected_features = [int(feat) for feat in selected_features]
            print(f"Converted␣to␣integer␣indices:␣{selected_features}")
        except ValueError as e:
            print(f"Could␣not␣convert␣feature␣names␣to␣integers:␣{e}")
            # If conversion fails, try to map to column positions
            if hasattr(original_X, 'columns'):
                # If X is a DataFrame, map feature names to positions
                feature_positions = []
                for feat in selected_features:
                    try:
                        pos = list(original_X.columns).index(feat)
                        feature_positions.append(pos)
                    except ValueError:
                        print(f"Feature␣{feat}␣not␣found␣in␣columns")
                selected_features = feature_positions
                print(f"Mapped␣to␣column␣positions:␣{selected_features}")

    return selected_features

def select_model(classifier):
    # Determine the model based on the classifier name
    if classifier == "SVM":
        model = SVC(kernel='linear')
    elif classifier == "RandomForest":
        model = RandomForestClassifier(random_state=42)
    elif classifier == "LogR":
        model = LogisticRegression(random_state=42, max_iter=10000)
    elif classifier == "DT":
        model = DecisionTreeClassifier(random_state=42)
    elif classifier == "MLP":
        model = MLPClassifier(random_state=42)
    elif classifier == "LDA":
        model = LinearDiscriminantAnalysis()
    elif classifier == "KNN":
        model = KNeighborsClassifier()
    else:
        raise ValueError("Unsupported␣classifier␣type")

    return model

def mRMR(X, y, classifier, num_features_to_select=None, range=(1,150), verbose=True):

    original_X = X
    model = select_model(classifier)
    # Handle both pandas DataFrame and numpy array inputs
    if isinstance(X, pd.DataFrame):
        X_array = X.values
    else:
        X_array = np.asarray(X)

    if isinstance(y, (pd.Series, pd.DataFrame)):
        y_array = y.values.ravel()
    else:
        y_array = np.asarray(y).ravel()  # Ensure y is a 1D array

    # Ensure proper data types
    X_array = X_array.astype(np.float64)  # Ensure X is float64 for compatibility

    # Handle categorical target variable
    if y_array.dtype == 'object' or not np.issubdtype(y_array.dtype, np.number):
        le = LabelEncoder()
        y_array = le.fit_transform(y_array)

    y_array = y_array.astype(np.int32)  # MRMR often expects integer labels

    # Check for NaN values and handle them
    if np.any(np.isnan(X_array)) or np.any(np.isnan(y_array)):
        print("Warning:␣NaN␣values␣detected.␣Consider␣handling␣them␣before␣feature␣selection.")
        # Remove rows with NaN
        valid_rows = ~(np.isnan(X_array).any(axis=1) | np.isnan(y_array))
        X_array = X_array[valid_rows]
        y_array = y_array[valid_rows]

    if num_features_to_select is not None:
        mRMR_selector = MRMR.mrmr(X_array, y_array)
        selected_features = mRMR_selector[0:num_features_to_select]
        return selected_features

    cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

    min_feat, max_feat = range
    max_feat = min(max_feat, X.shape[1]) #Don't exceed available features

    for test_num_feat in range(min_feat, max_feat + 1):
        try:
            mRMR_selector = MRMR.mrmr(X_array, y_array)
            selected_features = mRMR_selector[0:num_features_to_select]

            if len(selected_features) == 0:
                continue

            X_selected = X[:, selected_features]
            scores = cross_val_score(model, X_selected, y, cv=cv, scoring='accuracy')
            mean_score = np.mean(scores)

            if verbose==True:
                print(f"Features␣{test_num_feat}:␣{mean_score:.4f}␣+-␣{np.std(scores):.4f}")
                if len(selected_features) > 0:
                    if hasattr(original_X, 'columns'):
                        # Print feature names if DataFrame
                        feature_names = [original_X.columns[i] for i in selected_features]
                        print(f"␣␣Selected␣features:␣{selected_features}")
                        print(f"␣␣Feature␣names:␣{feature_names}")
                    else:
                        print(f"␣␣Selected␣features:␣{selected_features}")
                else:
                    print(f"␣␣No␣features␣selected")

            if mean_score > best_score:
                best_score = mean_score
                best_features = selected_features
                best_num_feat = test_num_feat

        except Exception as e:
            print(f"Error␣with␣{test_num_feat}␣features:␣{e}")
            continue

    print(f"\nBest:␣{best_num_feat}␣features␣with␣score␣{␣best_score:.4f}")
    if best_features is not None and len(best_features) > 0:
        if hasattr(original_X, 'columns'):
            best_feature_names = [original_X.columns[i] for i in best_features]
            print(f"Final␣selected␣feature␣indices:␣{best_features}")
            print(f"Final␣selected␣feature␣names:␣{best_feature_names}")
        else:
            print(f"Final␣selected␣feature␣indices:␣{best_features}")
    else:
        print("No␣features␣were␣successfully␣selected")

    return best_features

def Perm_importance(X, y, classifier, min_features=10, select_features=None):

    # Determine the model based on the classifier name
    model = select_model(classifier)

    # Handle both pandas DataFrame and numpy array inputs
    if isinstance(X, pd.DataFrame):
        X_array = X.values
        original_indices = X.columns.tolist()
    else:
        X_array = np.asarray(X)
        original_indices = list(range(X_array.shape[1]))

    if isinstance(y, (pd.Series, pd.DataFrame)):
        y = y.values.ravel()
    else:
        y = np.asarray(y).ravel()  # Ensure y is a 1D array

    # Ensure proper data types
    X_array = X_array.astype(np.float64)  # Ensure X is float64 for compatibility
```

```python

    if select_features is not None:
        if isinstance(select_features, list):
            if isinstance(select_features[0], int):  # Indices-based selection
                X_array = X_array[:, select_features]  # Subset X_array using indices
                original_indices = [original_indices[i] for i in select_features]
            elif isinstance(select_features[0], str):  # Names-based selection
                feature_indices = [original_indices.index(f) for f in select_features]
                X_array = X_array[:, feature_indices]  # Subset X_array using the corresponding indices
                original_indices = select_features

    model.fit(X_array, y)

    # Calculate permutation importance
    result = permutation_importance(model, X_array, y, n_repeats=10, random_state=42, n_jobs=-1)

    # Get the importances and sort them from most to least important
    importances = result.importances_mean
    indices = np.argsort(importances)[::-1]

    # Select features based on importance (threshold: features that have positive importance)
    selected_subset_indices = [i for i in indices if importances[i] > 0]  # Select features that have positive importance

    # Fallback: ensure at least `min_features` are returned
    if len(selected_subset_indices) < min_features:
        selected_subset_indices = indices[:min_features].tolist()

    if select_features is not None:
        # Map selected features back to original indices if necessary
        selected_features = [original_indices[i] for i in selected_subset_indices]
    else:
        selected_features = selected_subset_indices

    return selected_features

def backwards_SFS(X, y, classifier, select_features=None, n_features_to_select=20):

    # Determine the model based on the classifier name
    model = select_model(classifier)
    fast_model = LogisticRegression(random_state=42, max_iter=1000)
 # Handle both pandas DataFrame and numpy array inputs
    if isinstance(X, pd.DataFrame):
        X_array = X.values
        original_indices = X.columns.tolist()
    else:
        X_array = np.asarray(X)
        original_indices = list(range(X_array.shape[1]))

    if isinstance(y, (pd.Series, pd.DataFrame)):
        y = y.values.ravel()
    else:
        y = np.asarray(y).ravel()  # Ensure y is a 1D array

    # Ensure proper data types
    X_array = X_array.astype(np.float64)  # Ensure X is float64 for compatibility

    if select_features is not None:
        if isinstance(select_features, list):
            if isinstance(select_features[0], int):  # Indices-based selection
                X_array = X_array[:, select_features]  # Subset X_array using indices
                original_indices = [original_indices[i] for i in select_features]
            elif isinstance(select_features[0], str):  # Names-based selection
                feature_indices = [original_indices.index(f) for f in select_features]
                X_array = X_array[:, feature_indices]  # Subset X_array using the corresponding indices
                original_indices = select_features

    # Normalize the data
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X_array)

    model.fit(X_scaled, y)
    start_time = time.time()
    # Initialize SequentialFeatureSelector with the base model and the desired number of features to select
    sfs = SequentialFeatureSelector(fast_model, n_features_to_select=n_features_to_select, direction='backward', n_jobs=-1)

    # Fit SFS
    sfs.fit(X, y)
    selection_time = time.time() - start_time
```

```python

    # Get the selected feature indices
    selected_features = np.where(sfs.get_support())[0]

    return selected_features

def Lasso_selection(X, y, alpha=None, max_iter=2000,
   select_features=None):

    """
    Perform Lasso to select features.

    Parameters:
    - X: Input feature matrix (numpy array or pandas DataFrame).
    - y: Target labels (numpy array or pandas Series).
    - alpha: Regularization strength.
    - max_iter: Maximum number of iterations for convergence.

    Returns:
    - Selected feature indices.
    """
 # Handle both pandas DataFrame and numpy array inputs
    if isinstance(X, pd.DataFrame):
        X_array = X.values
        original_indices = X.columns.tolist()
    else:
        X_array = np.asarray(X)
        original_indices = list(range(X_array.shape[1]))

    if isinstance(y, (pd.Series, pd.DataFrame)):
        y = y.values.ravel()
    else:
        y = np.asarray(y).ravel()  # Ensure y is a 1D array

    feature_mapping = list(range(X_array.shape[1]))  # Maps from
       subset to original indices

    if select_features is not None:
        if isinstance(select_features, list):
            if isinstance(select_features[0], int):  # Indices-
               based selection
                X_array = X_array[:, select_features]  # Subset
                   X_array using indices
                feature_mapping = select_features
            elif isinstance(select_features[0], str):  # Names-
               based selection
                feature_indices = [original_indices.index(f) for
                   f in select_features]
                X_array = X_array[:, feature_indices]  # Subset
                   X_array using the corresponding indices
                feature_mapping = feature_indices

    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X_array)

    if alpha is not None:
        model = Lasso(alpha=alpha, random_state=42)
    else:
    # Fit L1 logistic regression model
        model = LassoCV(random_state=42)
    model.fit(X_scaled, y)

    best_alpha = model.alpha_
    print(best_alpha)
    # Get the selected feature indices
    selected_mask = model.coef_ != 0
    selected_subset_indices = np.where(selected_mask)[0]

    selected_features = [feature_mapping[i] for i in selected_subset_indices]

    return selected_features

def forwards_SFS(X, y, classifier, select_features=None, n_features_to_select=20):

    # Determine the model based on the classifier name
    model = select_model(classifier)
    fast_model = LogisticRegression(random_state=42, max_iter=1000)
 # Handle both pandas DataFrame and numpy array inputs
    if isinstance(X, pd.DataFrame):
        X_array = X.values
        original_indices = X.columns.tolist()
    else:
        X_array = np.asarray(X)
        original_indices = list(range(X_array.shape[1]))

    if isinstance(y, (pd.Series, pd.DataFrame)):
        y = y.values.ravel()
    else:
        y = np.asarray(y).ravel()  # Ensure y is a 1D array

    # Ensure proper data types
    X_array = X_array.astype(np.float64)  # Ensure X is float64 for compatibility

    if select_features is not None:
        if isinstance(select_features, list):
            if isinstance(select_features[0], int):  # Indices-based selection
```

```python
                X_array = X_array[:, select_features]  # Subset X_array using indices
            elif isinstance(select_features[0], str):  # Names-based selection
                feature_indices = [original_indices.index(f) for f in select_features]
                X_array = X_array[:, feature_indices]  # Subset X_array using the corresponding indices

    # Normalize the data
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X_array)

    model.fit(X_scaled, y)
    start_time = time.time()
    # Initialize SequentialFeatureSelector with the base model and the desired number of features to select
    sfs = SequentialFeatureSelector(fast_model, n_features_to_select=n_features_to_select, direction='forward', n_jobs=-1)

    # Fit SFS
    sfs.fit(X, y)
    selection_time = time.time() - start_time

    # Get the selected feature indices
    selected_features = np.where(sfs.get_support())[0]

    return selected_features

def without_fs(X, y):
    return X, y
```

Listing 3: Feature selection methods

D Appendix D — ***ROIs***

Table 53: Smith et al. (2009) 10/20 Resting-State Networks (RSNs)

Component #	**Network Name**
1	medial visual
2	occipital pole visual
3	lateral visual
4	default mode
5	cerebellum
6	sensorimotor
7	auditory
8	executive control
9	right frontoparietal
10	left frontoparietal

E Appendix E — *Features*

E.1 Graph Features

E.1.1 Dataset 1

- **ROI 1**
  - Closeness Centrality_ROI_1
  - Clustering Coefficient_ROI_1
  - Degree Centrality_ROI_1
  - Eigenvector Centrality_ROI_1
- **ROI 2**
  - Closeness Centrality_ROI_2
  - Clustering Coefficient_ROI_2
  - Degree Centrality_ROI_2
  - Eigenvector Centrality_ROI_2
- **ROI 3**
  - Closeness Centrality_ROI_3
  - Clustering Coefficient_ROI_3
  - Degree Centrality_ROI_3
  - Eigenvector Centrality_ROI_3
- **ROI 4**
  - Closeness Centrality_ROI_4
  - Clustering Coefficient_ROI_4
  - Degree Centrality_ROI_4
  - Eigenvector Centrality_ROI_4
- **ROI 5**
  - Closeness Centrality_ROI_5
  - Clustering Coefficient_ROI_5
  - Degree Centrality_ROI_5
  - Eigenvector Centrality_ROI_5
- **ROI 6**
  - Closeness Centrality_ROI_6
  - Clustering Coefficient_ROI_6
  - Degree Centrality_ROI_6
  - Eigenvector Centrality_ROI_6

- **ROI 7**
  - Closeness Centrality_ROI_7
  - Clustering Coefficient_ROI_7
  - Degree Centrality_ROI_7
  - Eigenvector Centrality_ROI_7
- **ROI 8**
  - Closeness Centrality_ROI_8
  - Clustering Coefficient_ROI_8
  - Degree Centrality_ROI_8
  - Eigenvector Centrality_ROI_8
- **ROI 9**
  - Closeness Centrality_ROI_9
  - Clustering Coefficient_ROI_9
  - Degree Centrality_ROI_9
  - Eigenvector Centrality_ROI_9
- **ROI 10**
  - Closeness Centrality_ROI_10
  - Clustering Coefficient_ROI_10
  - Degree Centrality_ROI_10
  - Eigenvector Centrality_ROI_10
- **ROI 11**
  - Closeness Centrality_ROI_11
  - Clustering Coefficient_ROI_11
  - Degree Centrality_ROI_11
  - Eigenvector Centrality_ROI_11
- **ROI 12**
  - Closeness Centrality_ROI_12
  - Clustering Coefficient_ROI_12
  - Degree Centrality_ROI_12
  - Eigenvector Centrality_ROI_12
- **ROI 13**
  - Closeness Centrality_ROI_13
  - Clustering Coefficient_ROI_13

  - Degree Centrality_ROI_13
  - Eigenvector Centrality_ROI_13
- **ROI 14**
  - Closeness Centrality_ROI_14
  - Clustering Coefficient_ROI_14
  - Degree Centrality_ROI_14
  - Eigenvector Centrality_ROI_14
- **ROI 15**
  - Closeness Centrality_ROI_15
  - Clustering Coefficient_ROI_15
  - Degree Centrality_ROI_15
  - Eigenvector Centrality_ROI_15
- **ROI 16**
  - Closeness Centrality_ROI_16
  - Clustering Coefficient_ROI_16
  - Degree Centrality_ROI_16
  - Eigenvector Centrality_ROI_16
- **ROI 17**
  - Closeness Centrality_ROI_17
  - Clustering Coefficient_ROI_17
  - Degree Centrality_ROI_17
  - Eigenvector Centrality_ROI_17
- **ROI 18**
  - Closeness Centrality_ROI_18
  - Clustering Coefficient_ROI_18
  - Degree Centrality_ROI_18
  - Eigenvector Centrality_ROI_18
- **ROI 19**
  - Closeness Centrality_ROI_19
  - Clustering Coefficient_ROI_19
  - Degree Centrality_ROI_19
  - Eigenvector Centrality_ROI_19
- **ROI 20**

- Closeness Centrality_ROI_20
- Clustering Coefficient_ROI_20
- Degree Centrality_ROI_20
- Eigenvector Centrality_ROI_20

- **Global Features**
  - Average Clustering
  - Diameter
  - Spectral Entropy
  - Mean Laplacian Eigenvalue
  - Max Laplacian Eigenvalue
  - Frobenius Norm (Laplacian Spectrum)
  - Algebraic Connectivity (λ_2)
  - Graph Energy

E.1.2 Dataset 2 (tuned parameters)

- **Group A_0_***
  - A_0_0, A_0_1, A_0_2, A_0_3, A_0_4, A_0_5, A_0_6, A_0_7, A_0_8, A_0_9, A_0_10, A_0_11, A_0_12, A_0_13, A_0_14, A_0_15, A_0_16, A_0_17, A_0_18, A_0_19
- **Group A_1_***
  - A_1_0, A_1_1, A_1_2, A_1_3, A_1_4, A_1_5, A_1_6, A_1_7, A_1_8, A_1_9, A_1_10, A_1_11, A_1_12, A_1_13, A_1_14, A_1_15, A_1_16, A_1_17, A_1_18, A_1_19
- **Group A_2_***
  - A_2_0, A_2_1, A_2_2, A_2_3, A_2_4, A_2_5, A_2_6, A_2_7, A_2_8, A_2_9, A_2_10, A_2_11, A_2_12, A_2_13, A_2_14, A_2_15, A_2_16, A_2_17, A_2_18, A_2_19
- **Group A_3_***
  - A_3_0, A_3_1, A_3_2, A_3_3, A_3_4, A_3_5, A_3_6, A_3_7, A_3_8, A_3_9, A_3_10, A_3_11, A_3_12, A_3_13, A_3_14, A_3_15, A_3_16, A_3_17, A_3_18, A_3_19
- **Group A_4_***
  - A_4_0, A_4_1, A_4_2, A_4_3, A_4_4, A_4_5, A_4_6, A_4_7, A_4_8, A_4_9, A_4_10, A_4_11, A_4_12, A_4_13, A_4_14, A_4_15, A_4_16, A_4_17, A_4_18, A_4_19
- **Group A_5_***

  - A_5_0, A_5_1, A_5_2, A_5_3, A_5_4, A_5_5, A_5_6, A_5_7, A_5_8, A_5_9, A_5_10, A_5_11, A_5_12, A_5_13, A_5_14, A_5_15, A_5_16, A_5_17, A_5_18, A_5_19

- **Group A_6_***

  - A_6_0, A_6_1, A_6_2, A_6_3, A_6_4, A_6_5, A_6_6, A_6_7, A_6_8, A_6_9, A_6_10, A_6_11, A_6_12, A_6_13, A_6_14, A_6_15, A_6_16, A_6_17, A_6_18, A_6_19

- **Group A_7_***

  - A_7_0, A_7_1, A_7_2, A_7_3, A_7_4, A_7_5, A_7_6, A_7_7, A_7_8, A_7_9, A_7_10, A_7_11, A_7_12, A_7_13, A_7_14, A_7_15, A_7_16, A_7_17, A_7_18, A_7_19

- **Group A_8_***

  - A_8_0, A_8_1, A_8_2, A_8_3, A_8_4, A_8_5, A_8_6, A_8_7, A_8_8, A_8_9, A_8_10, A_8_11, A_8_12, A_8_13, A_8_14, A_8_15, A_8_16, A_8_17, A_8_18, A_8_19

- **Group A_9_***

  - A_9_0, A_9_1, A_9_2, A_9_3, A_9_4, A_9_5, A_9_6, A_9_7, A_9_8, A_9_9, A_9_10, A_9_11, A_9_12, A_9_13, A_9_14, A_9_15, A_9_16, A_9_17, A_9_18, A_9_19

- **Group A_10_***

  - A_10_0, A_10_1, A_10_2, A_10_3, A_10_4, A_10_5, A_10_6, A_10_7, A_10_8, A_10_9, A_10_10, A_10_11, A_10_12, A_10_13, A_10_14, A_10_15, A_10_16, A_10_17, A_10_18, A_10_19

- **Group A_11_***

  - A_11_0, A_11_1, A_11_2, A_11_3, A_11_4, A_11_5, A_11_6, A_11_7, A_11_8, A_11_9, A_11_10, A_11_11, A_11_12, A_11_13, A_11_14, A_11_15, A_11_16, A_11_17, A_11_18, A_11_19

- **Group A_12_***

  - A_12_0, A_12_1, A_12_2, A_12_3, A_12_4, A_12_5, A_12_6, A_12_7, A_12_8, A_12_9, A_12_10, A_12_11, A_12_12, A_12_13, A_12_14, A_12_15, A_12_16, A_12_17, A_12_18, A_12_19

- **Group A_13_***

  - A_13_0, A_13_1, A_13_2, A_13_3, A_13_4, A_13_5, A_13_6, A_13_7, A_13_8, A_13_9, A_13_10, A_13_11, A_13_12, A_13_13, A_13_14, A_13_15, A_13_16, A_13_17, A_13_18, A_13_19

- **Group A_14_***

  - A_14_0, A_14_1, A_14_2, A_14_3, A_14_4, A_14_5, A_14_6, A_14_7, A_14_8, A_14_9, A_14_10, A_14_11, A_14_12, A_14_13, A_14_14, A_14_15, A_14_16, A_14_17, A_14_18, A_14_19

- **Group A_15_***
  - A_15_0, A_15_1, A_15_2, A_15_3, A_15_4, A_15_5, A_15_6, A_15_7, A_15_8, A_15_9, A_15_10, A_15_11, A_15_12, A_15_13, A_15_14, A_15_15, A_15_16, A_15_17, A_15_18, A_15_19

- **Group A_16_***
  - A_16_0, A_16_1, A_16_2, A_16_3, A_16_4, A_16_5, A_16_6, A_16_7, A_16_8, A_16_9, A_16_10, A_16_11, A_16_12, A_16_13, A_16_14, A_16_15, A_16_16, A_16_17, A_16_18, A_16_19

- **Group A_17_***
  - A_17_0, A_17_1, A_17_2, A_17_3, A_17_4, A_17_5, A_17_6, A_17_7, A_17_8, A_17_9, A_17_10, A_17_11, A_17_12, A_17_13, A_17_14, A_17_15, A_17_16, A_17_17, A_17_18, A_17_19

- **Group A_18_***
  - A_18_0, A_18_1, A_18_2, A_18_3, A_18_4, A_18_5, A_18_6, A_18_7, A_18_8, A_18_9, A_18_10, A_18_11, A_18_12, A_18_13, A_18_14, A_18_15, A_18_16, A_18_17, A_18_18, A_18_19

- **Group A_19_***
  - A_19_0, A_19_1, A_19_2, A_19_3, A_19_4, A_19_5, A_19_6, A_19_7, A_19_8, A_19_9, A_19_10, A_19_11, A_19_12, A_19_13, A_19_14, A_19_15, A_19_16, A_19_17, A_19_18, A_19_19