MÄLARDALEN UNIVERSITY
SCHOOL OF INNOVATION, DESIGN AND ENGINEERING
VÄSTERÅS, SWEDEN

Thesis for the Degree of Bachelor in Computer Science – 15.0 hp

NSGA-II DESIGN FOR FEATURE SELECTION IN EEG CLASSIFICATION RELATED TO MOTOR IMAGERY

Robin Johansson
rjn16002@student.mdh.se

Examiner: Ning Xiong
Mälardalen University, Västerås, Sweden

Supervisor: Miguel León Ortiz
Mälardalen University, Västerås, Sweden

2020-01-15

Abstract

Feature selection is an important step regarding Electroencephalogram (EEG) classification, for a Brain-Computer Interface (BCI) systems, related to Motor Imagery (MI), due to large amount of features, and few samples. This makes the classification process computationally expensive, and limits the BCI systems real-time applicability. One solution to this problem, is to introduce a feature selection step, to reduce the number of features before classification. The problem that needs to be solved, is that by reducing the number of features, the classification accuracy suffers. Many studies propose Genetic Algorithms (GA), as solutions for feature selection problems, with Non-Dominated Sorting Genetic Algorithm II (NSGA-II) being one of the most widely used GAs in this regard. There are many different configurations applicable to GAs, specifically different combinations of individual representations, breeding operators, and objective functions. This study evaluates different combinations of representations, selection, and crossover operators, to see how different combinations perform regarding accuracy, and feature reduction, for EEG classification relating to MI. In total, 24 NSGA-II combinations were evaluated, combined with three different objective functions, on six subjects. Results shows that the breeding operators have little impact on both the average accuracy, and feature reduction. However, the individual representation, and objective function does, with a hierarchical, and an integer-based representation, achieved the most promising results regarding representations, while Pearson's Correlation Feature Selection, combined with k-Nearest Neighbors, or Feature Reduction, obtained the most significant results regarding objective functions. These combinations were evaluated with five classifiers, where Linear Discriminant Analysis, Support Vector Machine (linear kernel), and Artificial Neural Network produced the highest, and most consistent accuracies. These results can help future studies develop their GAs, and selecting classifiers, regarding feature selection, in EEG-based MI classification, for BCI systems.

Table of Contents

1. Introduction

Brain Computer Interface (BCI) is a communication interface between a brain and an external device [1]. Since brain activity produces electrical signals that are measurable from the outside of a person's head, we can feed the signals to a BCI, that translates the electrical signals and converts them to commands for the external device. A person can then interact said electrical device with their mind. It is commonly used in medicine as aid, or treatment for different disabilities and disorders. An example being NeuroFeedback (NF) [2], where a BCI system is used in combination with *Motor Imagery* (MI) to measure, and display brain activity in real-time on a screen, or as audio feedback. MI, being defined as a mental simulation of a motor action, has proven to share similar neural activity to an actual motor action [3], and when recorded, can be recognized by the BCI, and visualized on a screen in real-time for a patient to view [2]. This can be used to train your brain and gain control over involuntary activities. In paper [2], they highlight some experiments done to try NF as treatment for Attention Deficit Hyperactivity Disorder (ADHD), and to help with rehabilitation after stroke. For people with ADHD it has been observed that they have higher amplitudes of low frequency (theta and delta waves) in their resting state, which are associated with relaxation. NF can be used to try and gain control, and lower these frequencies by looking at a screen with real-time feedback, and try to alter them. Thus, NF relies heavily on giving feedback in real-time, which is proven in [4], where they explain that delay between the neural activity, and the feedback from NF decreases the learning and rehabilitation capabilities.

The most common technology to measure, and record brain signals is *Electroencephalography* (EEG) [5].
This is because the equipment is not to expensive compared to other methods. Multiple electrodes are placed directly on the scalp of a person, unlike other methods where they have to be inserted by surgery. EEG is therefore seen as the most non-invasive method. The electrodes record the electrical activity over a period of time, which can then be stored in computer memory [5]. A drawback off EEG is that a sample recorded with this method often consist of a large number of features from multiple recording channels, and due to fatigue of the user, only a limited number of samples can be recorded [6], [5], [7]. This combination is not good when used as input for classification due to over-fitting problems (i.e when a model has "memorized" the noise and will perform well on its training data, but very poorly on novel data). EEG also record a lot of noise due to a problem called volume conduction [8]. In [7], and [9], they use Support Vector Machine (SVM) as classifier, because it works well with lots of features and few samples. Unfortunately, it takes a long time to compute, and as explained earlier, applications like NF need to have the data in real-time. Therefore, a *feature selection* step is required, which is a method of reducing the number of features, and only use a small subset that provide relevant and non-redundant information to the classification, and as a result, reduces the computational time [6], [7]. Feature selection is not an easy task since you need to find the features that are important to the classification, and distinguish them from the noise, otherwise the prediction accuracy will decrease [6], [7].

To find the important features, other studies have found that *Evolutionary Algorithms (EA)* are a good choice, and the most common algorithm used, is *Genetic Algorithm* (GA) [6]. This is a metaheuristic optimization algorithm based on biological evolution, with breeding operators such as selection, crossover, and mutation. It modifies a population of individual solutions over generations, with the goal of evolving the solutions over time, towards an optimal solution. The problem with a regular GA, or any other regular EA, is that it only works well for one objective, and feature reduction in this case have multiple objectives, reduce features, and not lower the accuracy. These two objectives are conflicting in the sense that if we reduce the features too much, the accuracy decreases, and if we focus only on the solutions with high accuracy, then the number of required features will rise. Therefore, a special version of EA must be used, a *multi-objective evolutionary algorithm* (MOEA), which can optimize multiple conflicting objectives simultaneously.

In [10], the research team did a comparison between a MOEA, and two single-objective algorithms, for a feature reduction problem related to EEG classification. The MOEA used was *Non-Dominated Sorting Genetic Algorithm II* (NSGA-II), and the single-objective GAs were, Steady Sate GA (SGA), and a regular GA. They showed that NSGA-II both had higher accuracy, and feature reduction, in comparison to the other two, with a slight drawback of being much slower to converge compared to SGA. In [6], another team created a novel version of NSGA-II, to solve a similar feature selection problem. The algorithm was compared to two other proven methods, and did produce a very promising result. Christoffer Parkkila continued the work from [7] in his bachelor thesis [9], where they had used a novel single-objective GA called *Hierarchical Genetic Algorithm* (HGA) for feature selection. He implemented a MOEA to be used instead of the HGA used previously. The MOEA he implemented was NSGA-II [9], and it was able to improve the result of the previous study. Many studies have used NSGA-II for similar

problems [6], [9], [10], [11], showing that the algorithm is a good choice for a feature selection problem related to EEG classification. However, there is still room for improvement, as explained earlier, to make NF efficient we need to keep the computational time as low as possible, since NF relies on giving feedback in real-time [2], [4], [7]. A low computational time requires a fast classification system, which requires good feature selection.

Since NSGA-II have proven to be a great solution to this problem, the purpose of this study is to improve the algorithm even further. NSGA-II has not yet been evaluated with different combinations of *individual representations* and *different breeding operators*, which this study aims to evaluate, in hopes of improving the prediction accuracy, for a smaller number of used features, which in turn will decrease the computational time. Three different individual representations, three selection, and three crossover operators will be tested for the original NSGA-II, and compared to the modified NSGA-II from [6], which will also be tested with the implemented selection operators. All combinations will be tested with three different objective-function pairs, *Pearson* with *Feature Reduction (Pearson-FR)* from [7], *Pearson* with *k-nearest neighbor* (Pearson-kNN) from [9], and *Training-validation percentage with kNN*, from [6]. To evaluate the results, five different classifiers will be used.

The paper is organized as follows: In Section 2, all necessary theory is presented, that is needed to understand this study. In Section 3, some of the related works in this field are presented, and discussed. Section 4 describes the problem more thoroughly, and the research questions are defined. In Section 5, the methods of research are provided, and in Section 6, the ethical considerations are presented. In Section 7, the system design, and implementation, are described, with the experiment settings being provided in Section 8. In Section 9, the experiment results are provided, and in Section 10, the results are discussed. Finally, Section 11 concludes the work, and future work is provided in Section 11.

2. Background

In this section, all the necessary theory, regarding the study are presented, and it is organized as follows: In Section 2.1, supervised learning, and related algorithms are described. In Section 2.2, the concept of feature reduction is explained, and in Section 2.3, the concept of evolutionary algorithms are described. In Section 2.4, a specific type of evolutionary algorithms is explained, namely, Genetic Algorithm. Finally, in Section 2.5, Multi-Objective problems are described, with NSGA, and NSGA-II being highlighted as solutions.

2.1. Supervised Learning

Supervised Learning is a subcategory of *Machine Learning*, commonly used for prediction problems [12]. The algorithms are building a mathematical model, by learning from known, labeled data. A supervised learning algorithm can either be classified as *Parametric*, or *Non-Parametric* [13]. Parametric algorithms try to simplify the structure of the problem according to a set of finite parameters, no matter how much data you input, the parameters will always be of the same amount. Usually, these parameters involve the mean, and standard deviation of normal distribution. A non-parametric algorithm does not have a fixed number of parameters. The complexity of the mathematical model depends on the amount of input data, by increasing the amount of input data, the amount of parameters required also increases. For parametric models, you only need to know the parameters of the mathematical model to classify new data, while non-parametric methods instead use the training data directly in the classification, by finding similar patterns between the input data, and the labeled training data. This means that a non-parametric algorithm always needs to keep the training data, while a parametric algorithm doesn't [13].

A supervised learning algorithm learn by classifying training data, which consists of sets of input features, and the correct outputs (labels) for each set [12]. By classifying the training data, they then have the ability to look at the correct label, and use that information to tune the underlying mathematical model. The goal of the training process is to improve the mathematical model so they later can classify novel data (i.e data they have not seen before, without correct labels). A set of features can also be called a data point, instance, or sample, which can be defined as $X = \{f_1, f_2, \dots, f_n\}$, where f_i is a feature belonging to sample *X*, and *n* is the total number of features [12].

To calculate an algorithms prediction accuracy for novel data, cross-validation is commonly used [12]. The input data sets are divided into groups, where one group is used for training, and the other is used for validation [12]. Usually, this is done in multiple rounds, called a *fold*, where the dataset is divided in groups differently each time.

Some common supervised learning algorithms are *k-Nearest Neighbor* (KNN), *Support Vector Machines* (SVM), Linear Discriminant Analysis (LDA), and Artificial Neural Network (ANN) [12].

2.1.1. K-Nearest Neighbor

K-nearest neighbor (KNN) is a non-parametric, supervised learning algorithm [12]. A non-parametric algorithm uses the training sets directly to classify novel data. The algorithm considers a sample to be point in a *n* dimensional space, where the number of dimensions (*n*) corresponds to the number of features that constructs the sample. This algorithm is based on the assumption that similar data points exist close to each other. When assigning a class to an input sample *x*, it finds the $k \in \mathbb{N}$ closest points to *x* in the training data sets, and assign *x* the class that is most common between the *k* closest points.

There are multiple ways to calculate the distance between two samples, with the most common method being the *Euclidean Distance* (ED) [12]. It is defined as follows:

$$ED(X,Y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \quad (1)$$

Where X, and Y, are two samples, n is the number of features in each set, x_i is a feature from X, and y_i is a feature from Y. Figure 1 shows an activity diagram over the algorithm:

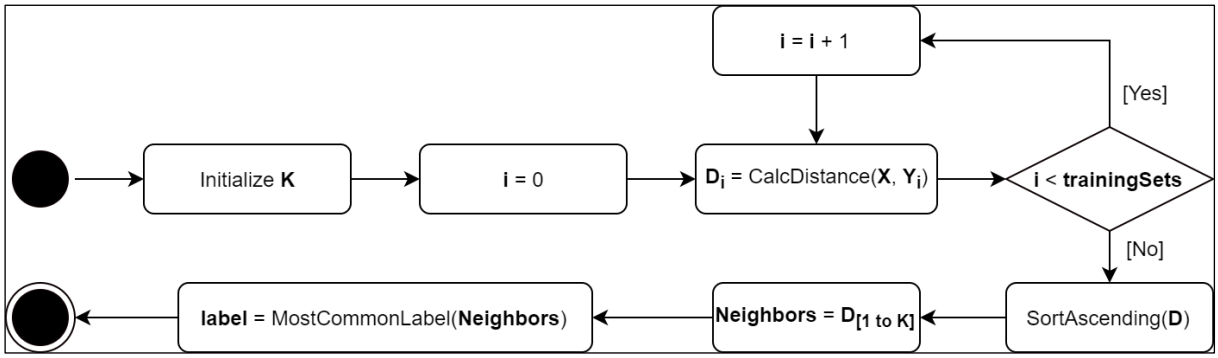

Figure 1. KNN activity diagram.

The algorithm starts by initializing k, the number of neighbors to use [12]. Then it goes through all training samples Y_i, and calculates the distances between them, and the input sample X. When all distances have been calculated, they get sorted in an ascending order. Then the algorithm takes the first k samples, and look at what their labels are. The most common label will get assigned to X. Figure 2 illustrates how it can look with $k = 5$, and two features. The red circle looks at the 5 closest instances, and since four of them are blue, it will be assigned class 1.

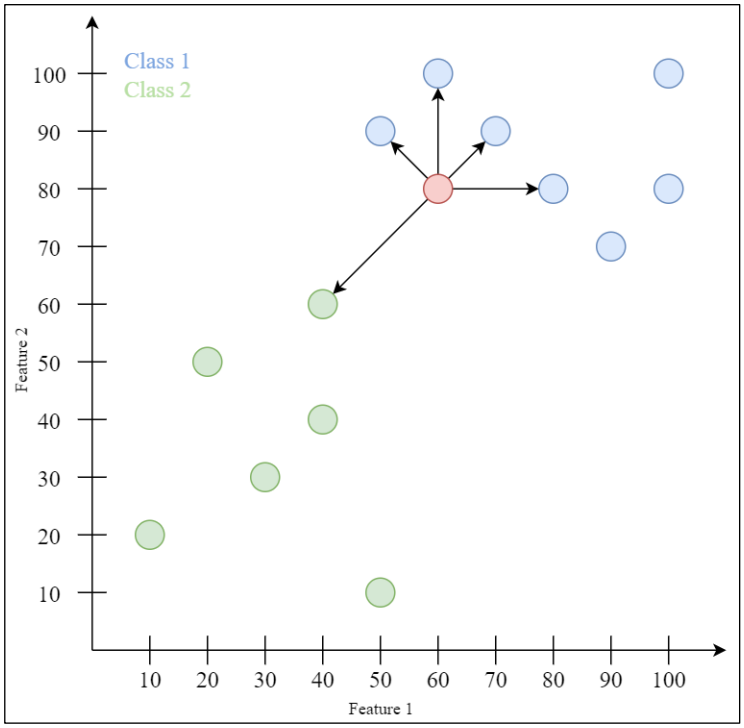

Figure 2: Example of KNN with k=5, and 2 features.

2.1.2. Support Vector Machine

Support Vector Machine (SVM) is a non-parametric, supervised learning algorithm [12]. This algorithm is able to classify between two different classes, making it a *binary classifier*. It works by dividing the training samples with a hyperplane, which is placed in the middle of two classes, with as much margin between the classes, and the hyperplane as possible [13]. This is calculated by using *support vectors*, which are vectors that defines the maximum space between the two classes. The support vectors are essentially the samples from each class, that are closest to the opposing class. With these vectors, the hyperplane can be calculated, and placed in the middle of the

resulting area, which will give the maximum margin to both classes. The algorithm can then use this plane to assign new samples a class, depending on which side of the hyperplane the point lands on. The dimension (n) of the hyperplane depends on the number of features (k) in a sample, where $n = k$. In a 2-dimensional space the hyperplane becomes a line, this is illustrated by Figure 3.

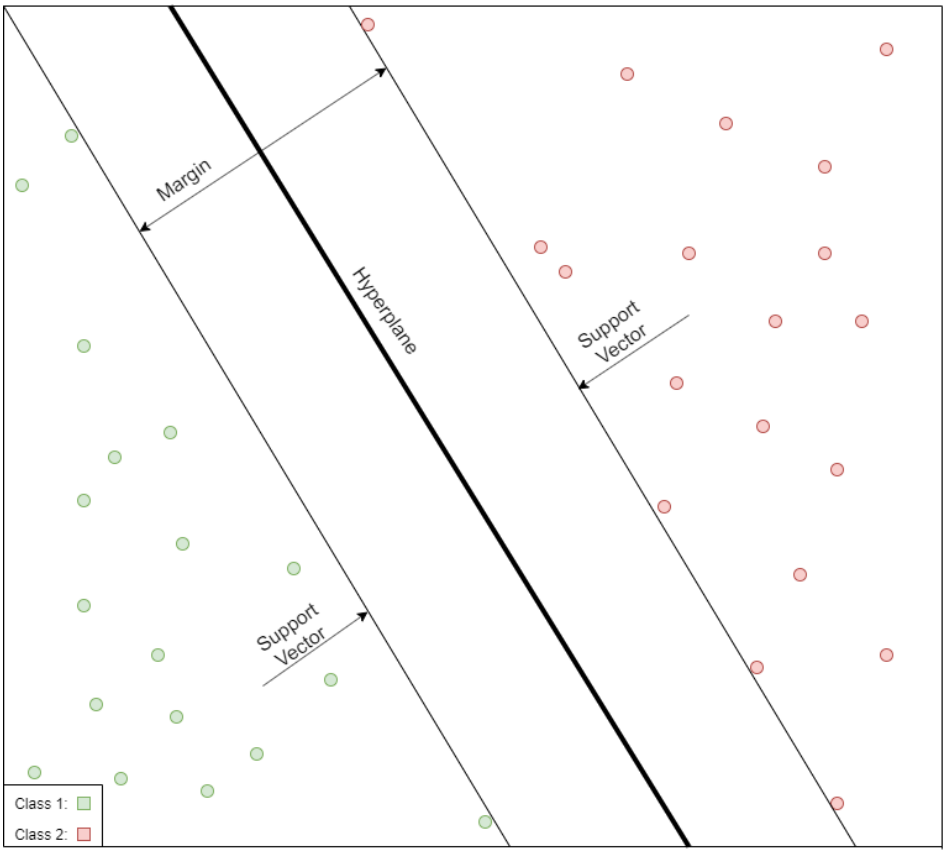

Figure 3. Simple illustration of a 2-dimensional hyperplane, highlighting the margin, support vectors, and the hyperplane. The samples on the smaller lines, are the support vectors.

To then classify a new sample, the algorithm can use the following equation [13]:

$$f(X) = \omega_0 + \omega_1 x_1 + \cdots \omega_n x_n \quad \textbf{(2)}$$

Which simply is a linear equation, that works for any number of features and dimensions (n) [13]. In EQ. (2), ω_i are the coefficients of the hyperplane, and x_i represents a feature of sample X. Since it's a binary classifier, only two labels are needed, the label (y) for a sample, X_i, is therefore defined as: $y_i \in \{-1,1\}$. To know what class X belongs to, the algorithm only has to look at the sign of EQ. (2), if $f(X) > 0$, it belongs to class 2 ($y_i = 1$), and if $f(X) < 0$ it belongs to class 1 ($y_i = -1$). To determine the distance of sample X from the hyperplane, it can calculate $|f(X)|$, which can be used as a confidence measure, where the algorithm is more confident the further away from the hyperplane X resides. However, this approach only works if the boundaries between the two classes are linear.

As previously mentioned, the hyperplane should be placed in a way that it is in the perfect middle of the two classes, with as much margin between them as possible [13]. To find this plane, we have to solve an optimization problem, where we want to maximize the margin width (M). M then relates to the hyperplane vector $\vec{\omega}$, where it is placed, and in what direction it points towards, which means that to maximize M, the algorithm needs to find the optimal hyperplane coefficients ω_i. At the same time, we want to keep the constraints of $f(X) > 0$, and $f(X) < 0$, which are used to differentiate between the two classes. This gives the optimization problem:

$$\underset{\omega_0, \omega_i, \dots, \omega_n, \mu_1, \mu_2, \dots, \mu_p}{maximize\ M} \quad (3)$$

$$subject\ to \sum_{j=1}^{n} \omega_j^2 = 1, \quad (4)$$

$$y_i \left(\omega_0 + \sum_{j=1}^{n} \omega_j x_j \right) \geq M(1 - \mu_i), \quad (5)$$

$$\mu_1 \geq 0, \sum_{i=1}^{p} \mu_i \leq C \quad (6)$$

In this optimization problem, EQ. (3) tells us what we are optimizing, which is the margin (*M*) [13]. *M* is optimized by tuning the parameters ω_i, and μ_i, where w_i are the hyperplane coefficients, and μ_i are *slack variables*, which corresponds to the distance of sample X_i relative to the margin (*M*). EQ. (4) together with EQ. (5) ensures that most of the training samples will be on the correct side of the hyperplane, with a minimum distance of *M*. In EQ. (5) we can see the linear equation from EQ. (2) inside the parentheses. Variable y_i will be the resulting class label, $y_i \in \{-1, 1\}$, and from $M(1 - \mu_i)$, we can see that some solutions will be allowed on the wrong side of the margin (if the corresponding slack variable $\mu_i > 0$). To find the optimal plane, all samples have to pass the criteria of EQ. (5), which means that a sample with label $y_i = -1$ need to have a negative distance from the plane, where $\omega_0 + \sum_{j=1}^{n} \omega_j x_j$ produces a negative number . This means that if a solution is on the wrong side of the hyperplane, the equation criteria will not be met, and the position of the plane have to be changed [13]. The slack term in EQ. (5) ensures that some terms will be allowed inside of the margin area, since μ_i will be > 1, and $M(1 - \mu_i)$ will be smaller than the margin itself (*M*), and because of this, samples closer to the plane will be allowed since the right side of EQ. (5) will be smaller. EQ. (6) is the penalty of misclassification, $\mu_1 \geq 0$ ensures that the slack variable will be positive, and $\sum_{i=1}^{p} \mu_i \leq C$ is the amount of penalty, which is the sum of all slack variables. It also limits the amount of slack to *C*, which is a tuning parameter, commonly chosen by cross-validation. If a sample is on the correct side of the margin, the slack term $\mu_i = 0$. If $\mu_i > 0$ the sample is on the wrong side of the margin, and if $\mu_i > 1$, then it is on the wrong side of the hyperplane, which will penalize even more [13]. With *C* you can therefore decide how much classification error the algorithm should allow, and a bigger *C* will result in a bigger margin, with higher bias. A small value for *C* will result in a narrow margin, with lower bias probability, but higher variance. EQ. (3)-(6) can be written in a more compact equation, similar to a linear regression problem:

$$\underset{\omega_0, \omega_i, \dots, \omega_n}{minimize} \left\{ \sum_{i=1}^{p} max[0, 1 - y_i f(X_i)] + C \sum_{j=1}^{n} \omega_j^2 \right\} \quad (7)$$

$$f(X_i) = \omega_0 + \sum_{k=1}^{n} \omega_k x_k \quad (8)$$

Where $max[0,1 - y_i f(X_i)]$ is called a *hinge loss* function, which is 0 if the sample is on the correct side of the margin, otherwise the value is related to the distance from the margin [13]. The second term $C \sum_{j=1}^{n} \omega_j^2$ corresponds to the penalty, where a large C results in smaller ω_k, with more misclassifications allowed. EQ. (8) is the linear equation from EQ. (2), which is used both for the optimization of the hyperplane, and when we want to classify new samples. The above optimization is called a *Quadratic Programming* (QP) problem. This is a complex area in itself, and won't be described in this paper. One can solve such optimization problem with for example Platt's *Sequential Minimal Optimization* algorithm (SMO). For a detailed explanation of this algorithm, and how to solve these problems, I refer you to [14].

For a non-linear boundary, where samples are mixed in a way where no straight line can divide the two classes, the method gets more complicated [13]. In a case such as this, the algorithm uses *kernels*. A kernel is a method of transforming the feature space to a higher dimension, where the two classes now can be divided by a hyperplane of a higher dimension. The hyperplane itself is of dimension n-1, which in a 2D feature space is a line (1D), and in a 3D feature space it is a 2D plane. When transforming back to the original feature space, the hyperplane will

look non-linear, and may be curved in different ways. To solve this, we need to change the linear equation, and give it the ability to enlarge the feature space, using higher-order functions. This is done by using *kernels*.

By looking at the behavior of EQ. (7)(8) , it has been found that only the inner products between the samples are needed, and only from the samples that act as support vectors. Therefore, the linear classifier from EQ. (8) can be rewritten as:

$$f(X_i) = \omega_0 + \sum_{k=1}^{n} \alpha_k \langle X, Y_k \rangle \quad \textbf{(9)}$$

Where α_k are parameters that will be found, and optimized when inserting EQ. (9) into the optimization equation EQ. (7). These parameters will be 0 for all solutions on the right side of the hyperplane, which indicates that only the support vectors affect the optimization, both for the hyperplane, and when classifying new samples. Lastly, $\langle X, Y_k \rangle$ is the inner product between samples, X, and Y, defined as:

Linear Kernel $$\langle X, Y_k \rangle = \sum_{i=1}^{p} x_i y_i \quad \textbf{(10)}$$

Putting EQ. (9) and (10) together gives us the same linear classifier as EQ. (8). A kernel is actually the inner product part, and EQ. (10) is the *linear kernel*. A kernel *K(X, Y)*, is in fact:

$$K(X, Y) = \langle X, Y \rangle \quad \textbf{(11)}$$

The equation used for inner product calculation is therefore known as the kernel. Two other popular kernels are the *Polynomial Kernel* (PK), and the *Radial Basis Function Kernel* (RBF):

Polynomial Kernel $$K(X, Y_i) = \left(1 + \sum_{j=1}^{n} x_j \, y_j\right)^d \quad \textbf{(12)}$$

Radial Basis Function Kernel $$K(X, Y_i) = exp\left(-\gamma \sum_{j=1}^{n} \left(x_j y_j\right)^2\right) \quad \textbf{(13)}$$

The polynomial kernel is almost identical to a linear kernel, in fact, if parameter *d* is equal to 1, it is a linear kernel. Otherwise, using $d > 1$ transforms the feature space to a higher dimension, and involves polynomials of degree *d*. The optimal value for this parameter is usually found by cross-validation.

Radial basis function kernel calculates the relationship between two samples in a higher dimension, without actually transforming it. This process is also referred to as *Kernel Trick*, which avoids the math of actually transforming, and saving some computations needed. In this kernel, γ is the parameter, which is a positive constant. An interesting behaviour of this kernel, is that it works somewhat similar to KNN, where only the closest samples have an effect on the classification process, when classifying a new sample *X*.

In summary, EQ. (7) is used to find the optimal hyperplane, and support vectors, where the function $f(X)$ defines the hyperplane, and is used when classifying new samples. That function is defined as:

$$f(X_i) = \omega_0 + \sum_{k=1}^{n} \alpha_k K(X, Y_k) \quad \textbf{(14)}$$

Where $K(X, Y_k)$ can be either of EQ. (10), (12), or (13).

2.1.3. Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) is a parametric, supervised learning algorithm [12]. This method tries to find the linear combination of features that best separates multiple classes. LDA works both as a binary classifier, and

a multiclass classifier. The algorithm makes the assumptions that all samples within a class are drawn from a multivariate normal distribution, and that each sample in a given class use the same mean vector [13]. It also assumes that all classes share the same covariance matrix (i.e a matrix giving the joint variability between each pair of elements). The algorithm looks for patterns in feature space between the different samples, and classifies new samples based on similar training samples.

LDA is based on *Bayes' Theorem*, and to understand it you need to know the terms used in the equations [13]. First you need to know how to calculate the *prior probability* (π) of a given class (k). This is calculated by dividing the number of training samples of class k, with the total number of training samples:

$$\pi = \frac{n_k}{n} \quad \textbf{(15)}$$

Then LDA use mean vectors of each class [13], which simply is the mean of all samples belonging to class (k), where p is the number of features, n_k is the number of samples in class k, and x_i is feature i of each sample:

$$\mu_k = \frac{1}{n_k} \sum_{i:y_i=k}^{p} x_i \quad \textbf{(16)}$$

Lastly, LDA calculates a covariance matrix, which can be seen as a weighted average of sample variances for each of the K classes [13]. In the equation, K is the total number of classes, p is the number of features, and x_i is feature i of each sample:

$$\Sigma = \frac{1}{n-K} \sum_{k=1}^{K} \sum_{i:y_i=k}^{p} (x_i - \mu_k)^2 \quad \textbf{(17)}$$

These prior variables all have to be estimated before use in the following equations [13]. *Bayes' Theorem* is then defined as:

$$f_k(X) = \frac{\pi_k g_k(X)}{\sum_{i=1}^{K} \pi_i g_i(X)} \quad \textbf{(18)}$$

Where X is the sample to be classified, π is the prior probability of class k, and $g(X)$ is the density function [13]. Since, the algorithm assumes multivariate normal distribution, the density function is defined as:

$$g(X) = \frac{1}{(2\pi)^{\frac{p}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(X-\mu)^T \Sigma^{-1} (X-\mu)\right) \quad \textbf{(19)}$$

Where p is the number of dimensions (features in sample X), Σ is the covariance matrix, and μ is a mean vector of all training samples belonging to class k [13]. The final LDA classifier is then the combination of EQ. (19), and (18). After some simplification of the equation, the resulting equation will look like his:

$$f_k(X) = X^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log(\pi_k) \quad \textbf{(20)}$$

Which is a linear function, and as a result, will compute the linear combination of the features [13]. However, before the algorithm can be calculated, the algorithm has to estimate the variables π, μ, and Σ.

The algorithm then classifies each sample according to EQ. (20), where it calculates $f_k(X)$ for each class k, and the class that returns the highest value will be assigned to sample X [13]. This LDA is only the binary classification version, for classifying more than two classes, I refer you to [13].

2.1.4. Artificial Neural Network

Artificial Neural Networks (ANN), is a parametric, supervised learning algorithm [15]. This algorithm works very well for both linear, and non-linear data. It is based on a very simple biological brain, mimicking the human

nervous system. It consists of a network of artificial neurons, where a neuron simply is a container with a single number between 0 and 1, called a neurons *activation*. The simplest form of a neural network is called a *Perceptron*.

2.1.4.1. Perceptron

The *perceptron* is a binary classifier, only capable of classifying linear separable problems [15]. It consists of multiple input signals, connected to a single neuron. Each connection between an input signal (x_i), and the neuron (y), is called a *weight* (w_i), which represents how important that particular input is to the neuron. The weight is also a number, where a bigger value corresponds to a higher importance of the connected input. The neuron also has a single output signal, which is based on the weighted sum of input signals, and their associated weights. Calculating the output of a neuron is done by an *activation function*. Since the perceptron is a linear (binary) classifier, the activation function also has to be linear, usually the inner product between the input signals, and the associated input weights. If that sum is bigger than a certain threshold, the neuron will *fire* (i.e activated), and output a 1, otherwise it will output a 0 [15]. The linear activation function can then be defined as:

$$y = \begin{cases} 1, & if \sum_{i=1}^{n} w_i x_i \geq b \\ 0, & if \sum_{i=1}^{n} w_i x_i < b \end{cases} \quad (21)$$

Where n is the number of input signals, and b is the *bias*. The bias is the activation threshold, also connected to the neuron [15].

2.1.4.2. Multilayered Perceptron

A *multilayered perceptron* (MLP) is, as its name suggests, a network of multiple perceptrons, divided into layers [15]. This is what is called a neural network, and the algorithm is a non-binary classifier, which is able to classify non-linearly separable problems.

The structure of a neural net consists of layers (L) of neurons, where you have an input layer ($L = 0$) of (n_{input}) neurons, multiple hidden layers of (n_{hidden}) neurons [15]. They are called hidden because they are in between the input, and output layer, where it's output, or input, is hidden from the user. Lastly, an output layer of (n_{output}) neurons. Each neuron in the input layer is connected to the first hidden layer. The neurons in the first hidden layer is then connected to all neurons in the next hidden layer, and the last hidden layer is all connected to all output neurons. This structure with multiple layers of neurons is called a *Multilayered Perceptron*. Figure 4 illustrates a simple multilayered perceptron, with three input neurons, one hidden layer of four neurons, and two output neurons:

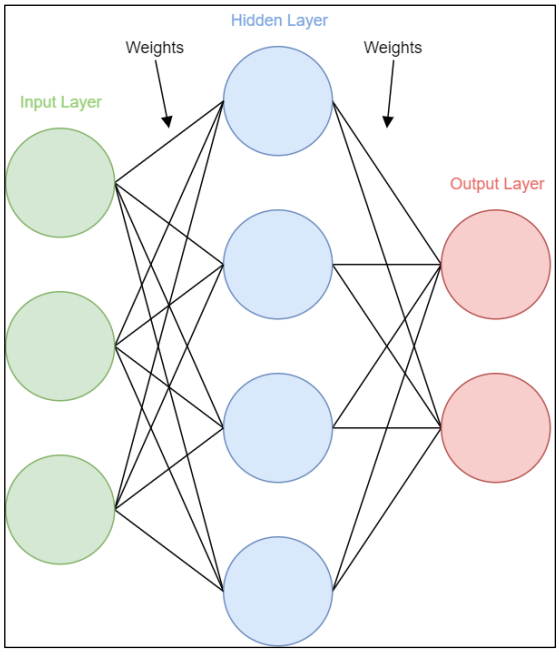

Figure 4. Illustration of a simple, multi-layered perceptron.

In a structure like this, all neurons (except those in the input layer) have their own associated bias, and each layer can have their own activation function [15]. MLP is slightly more complex than a normal perceptron, and is usually divided into two parts, *Forward Step*, and *Backpropagation*.

To understand forward step, and backpropagation, you first need to know about the available activation functions. There are a number of activation functions that can be used [15]. Some of the most common are *Sigmoid*, *Softmax*, and *Rectified Linear Unit* (ReLU):

Sigmoid $$\sigma_j(x) = \frac{1}{1 + e^{-x}} \quad (22)$$

Softmax $$\sigma_j(x_i) = \frac{e^{x_i}}{\sum_{k \in L} e^{x_k}} \quad (23)$$

ReLU $$\sigma_j(x) = \begin{cases} x & if\ x > 0 \\ 0 & if\ x \leq 0 \end{cases} \quad (24)$$

In these equations, x is a vector of all the input signals, while x_i is one of them.

- **Forward Step** is where the algorithm calculates the output from the input data, and making a prediction [15]. Essentially, it calculates the activations for each neuron, one layer at a time, until it has calculated the activation for the output neurons. Since MLP are able to classify non-linear problems, the activation function from EQ. (21) is not enough, since it only outputs either a 0 or a 1. Instead, it takes the weighted sum, which is the inner product between the input signals, and the associated weights, added with the bias. Then it feed that result through a non-linear activation function, that compresses the result into a number between 0 and 1. The resulting equation to calculate the output from neuron *j* is therefore defined as EQ. (25), where σ is an activation function, and L is the layer where neuron *j* belongs to.

$$y_j = \sigma\left(\sum_{i \in L-1} w_i x_i + b\right) \quad (25)$$

- **Backpropagation** is how the network learns [15]. This is done by updating all the weights and biases relative to the observed cost of the prediction, to minimize the cost. This is done using a cost function, the correct label, and gradient descent. The cost function (C) calculates the cost, relative to the output, and the desired output (correct label of the training data), then by using gradient descent the algorithm calculates how much each weight or bias should change to minimize the cost. This process starts from the output layer (L_{output}), and moves backwards in the network to update the weights in each layer.

A number of different cost functions exists [15], with three of the most common methods of calculating the cost for neuron *j* are the *Squared Error*, *Mean Squared Error*, and the *Euclidean Distance*:

Squared Error $$\varphi_j = \sum_{k \in L_{output}} (y_k - o_k)^2 \quad (26)$$

Mean Squared Error $$\varphi_j = \frac{1}{n_{output}} \sum_{k \in L_{output}} (y_k - o_k)^2 \quad (27)$$

Euclidean Distance $$\varphi_j = \sum_{k \in L_{output}} \frac{1}{2}(y_k - o_k)^2 \quad (28)$$

The error term (δ_j), is for each neuron in the output layer calculated by using the derivatives of the cost, and activation function [15]. For a neuron (j) in the hidden layer, it also uses the derivative of the activation function, but instead of the cost function, it uses the weights, and the error term of the neurons it is connected to in the layer

in front of itself ($L + 1$). EQ. (29) is the resulting equation, of calculating the error term for a neuron, where t_i is the desired output of neuron *j*, and y_j is the actual output.

$$\delta_j = \begin{cases} \varphi'(t_j - y_j)\sigma'(y_j) & if\ j\ is\ an\ output\ neuron \\ \left(\sum_{\ell \in L+1} w_{j\ell}\delta_\ell\right)\sigma'(y_j) & if\ j\ is\ a\ hidden\ neuron \end{cases} \quad \textbf{(29)}$$

The error term is then used to update the weights, and biases [15]. It is updated according to the following rule, which use a learning rate (η), an error term (δ), and (x_{ji}), which is the output of neuron (i), and input to neuron (j). The rule is defined as:

$$\Delta w_{ji} = \eta \delta_j x_{ji} \quad \textbf{(30)}$$

The result Δw_{ji}, is the new weight between neuron (j) and neuron (i) [15]. The learning rate (η) is controlled by the user, and is a positive number that controls the step size of the weight tuning. This is how a basic neural network works. In summary, first initialize the weights to random, small values. Then, for each training example $\{x_1, x_2, \dots, x_n\}$, and the corresponding labels $\{t_1, t_2, \dots, t_n\}$, do the following:

i) Use the training example as input signals to the network
ii) Compute the outputs for each layer (Forward Step)
iii) Compute the error terms in the output layer (Backpropagation)
iv) Update the weights between the last hidden layer, and the output layer
v) Compute the error terms in the hidden layers (From the layer closest to the output, and backwards)
vi) Update the weights of the hidden layers
vii) Repeat until the cost function returns a low value of overall error

2.2. Feature Reduction

Feature Reduction is used in the pre-processing stage of a classifier, for reducing the number of features in a large dataset [16]. This is to make the classifier more efficient regarding performance. Unfortunately, features often depends on each other, thus the prediction accuracy of the classifier may suffer if wrong features are removed. Therefore, feature reduction depends on removing only the *irrelevant* and *redundant* features, also called *noise*, and keep only those relevant to the classification. That is to increase the speed of the classifier, and to reduce *overfitting*, which is a problem where a classifier contains more parameters with a relationship to noise, thus making decisions based on said irrelevant, and redundant features. Two of the main methods of solving feature reduction problems are *filters*, and *wrappers* [16].

- **Filters** make use of statistical applications, which are independent of the final classifier. They often rank the features with correlation coefficients, with two examples being the *Pearson Correlation Coefficient* (PCC) [6], [16], and *Spearman Correlation Coefficient* (SCC) [6]. This makes this method computationally fast, but it usually has a lower prediction accuracy compared to other methods.

- **Wrappers** depends on the classifier, which is used to evaluate a subset of features, that have been selected by a search algorithm, like an *Evolutionary Algorithm* (EA), or a *Particle Swarm Optimization* (PSO) based algorithm [6], [16]. The classifier needs to go through the whole process of cross-validation on each of the feature subsets, making this method more accurate, but require more time, compared to a filter method.

2.3. EA

Evolutionary algorithms (EA) are a set of algorithms in the area of Artificial Intelligence [17]. They are *metaheuristic optimization algorithms*, based on biological evolution, first explained in the book "*On the Origin of Species*" by Charles Darwin, 1859. The theory commonly known as *Darwin's theory of evolution*, or *Darwinism*, explains the natural selection, and evolution of species. Sometimes described as "Survival of the fittest", where only the organisms who are best adapted to the environment and its conditions, survive long enough to reproduce [17]. Fitness in this case refers to the organism's ability to survive the environments conditions. The diversity of

all living organisms is also explained by natural selection, where only the strongest genetics gets carried over during reproduction, with small variations. These variations are called *mutation*, and it's where some of the original genetics have a small chance of being randomly changed [17], [18].

Evolutionary algorithms mostly work the same way, with reproduction, selection, recombination and mutation all based on biological evolution [17], modifying a population of individual solutions over generations, with the goal of evolving the solutions over time, towards an optimal solution. The difference being the environment, and the conditions surrounding it. Fitness refers to the quality of a solution, how good a solutions perform at a given problem [17], [18]. The selection operator, then considers each solutions fitness, when selecting individuals to reproduce [13]. There are a many different types of evolutionary algorithms, and each type is good for different kinds of problems, and different complexities [18]. Many mathematical problems can be solved by EAs, they will however need to be tailored to that specific problem [19]. Some common EAs are *Genetic Algorithm* (GA) [19], *Differential Evolution (DE)* [20], and *Genetic Programming (GP)* [21]. Off them, the most popular EA is GA, which also is most relevant to this paper, and is therefore described in the next section [18].

2.4. GA

Genetic Algorithms (GA) are a subcategory of the EA family [19], first introduced by John Holland in 1975, and extended further in 1989 by one of his students, David E. Goldberg [19]. GA make use of natural selection to evolve a population of individuals during iterations, creating new improved populations each generation, and the solutions represented by the individuals moves towards a global optimum. This section describes the key parts of a general GA.

In GAs, an individual is commonly called a *chromosome*, and its data points are called *genes* [19]. They are a representation of a solution to the given problem. A chromosome can be represented in many different ways, most commonly binary strings, real number array, binary trees, character strings, parse trees, or directed graphs [19], the most common for a classic GA being a binary string [19]. Usually, the algorithm consists of six different key components: *Initialization*, *Selection*, *Crossover*, *Mutation*, *Replacement*, and *Termination* [19].

2.4.1. Initialization

Usually, the algorithm starts by initializing a population of randomly generated chromosomes [19]. In a binary encoded chromosome this would be a string with randomly placed zeroes and ones. Both the number of chromosomes (k) in a population, and the length (L) of a chromosome, is predefined.

2.4.2. Selection

In the selection operator, the algorithm chooses which of the chromosomes in the population that gets to reproduce, and generate the chromosomes for the next iteration [19]. The algorithm selects the best fitted chromosomes to make sure that the next generation of chromosomes won't go down in fitness after recombination, and hopefully, the recombined chromosomes can be even better than its predecessors. But at the same time you want to select some of the less fitted chromosomes to keep the diversity, since the main goal is to find a better solution than what it already have found, and it doesn't know if some parts of the less fitted chromosomes, together with some parts of the best fitted chromosomes can result in even higher fitness. If it always were to choose the highest fitted chromosomes, not much would change when they are combined, since they may look very similar, and as a result, the solutions will stop improving. Thus, selection methods often introduce some kind of randomness to this process to stop that from happening. That randomness gives a small probability of lesser fitted individuals to get selected, which keeps the diversity in the next generation of chromosomes. There are many different operators that can be used for selection, but they all follow the same principle. Parents are randomly selected in a way that favours the fittest chromosomes [22]. Some of the most common selection operators are: *Roulette Wheel Selection* (RWS), *Stochastic Universal Sampling* (SUS), *Linear Rank Selection* (LRS), *Exponential Rank Selection* (ERS), *Tournament Selection* (TOS), and *Truncation Selection* (TRS) [23], where RWS, SUS, and TOS are described further, since they are most relevant to the study.

- ***Roulette Wheel Selection*** **(RWS)** assigns every chromosome (C_i) a probability (p_i) of being selected [23]. The probability is calculated by the following formula:

$$p_i = \frac{f_i}{\sum_{j=1}^{k} f_j} \quad \textbf{(31)}$$

Where *k* is the size of the population, and $\sum_{j=1}^{k} f_j$ is the fitness sum of all chromosomes in the population. Then it randomizes a number $r \in \{0, 1\}$. The algorithm then starts adding the fitness values (*f*) of each chromosome, until it reaches *r*. The last chromosome to be added is the one who gets selected. The time complexity of this algorithm is $O(n^2)$ because it requires *n* iterations to fill the population. With this method, a fitter solution has a much greater probability of being selected.

- ***Stochastic Universal Sampling*** **(SUS)** is a further developed version of RWS that aims to reduce the risk of premature convergence [23]. It is very similar to RWS, but instead of using one fixed point *r*, it uses as many as there are chromosomes to be selected. All chromosomes are mapped continuously on a line, where the size of an individual's segment is its assigned fitness value. Each fixed point is then evenly distributed across the line, spaced by a distance of $\bar{f}$ between them, where $\bar{f}$ represent the mean value of the population's total fitness, and it is calculated by Eq. (32). This approach makes it so that higher fitted individuals have a higher probability of being selected, since their segments are bigger. But since the fixed points are distributed across the whole line, some of the lesser fitted individuals will always be selected, giving the lesser fitted individuals a higher probability of being selected, compared to RWS.

$$\bar{f} = \frac{1}{k}\sum_{i=1}^{k} f_i \qquad (32)$$

The algorithm then generates a *uniformly distributed random number* as a starting point, $R \in [0, \bar{f}]$. It then start by assigning the initial point $r_1 = R$, and for the rest, keep adding the mean value $\bar{f}$ to the previous fixed point $r_i = r_{i-1} + \bar{f}\ where\ i\ \in \{2, 3, 4, \dots k\}$, where *k* is the number of individuals to select. This will generate a set of fixed points $P\{r_1, r_2, r_3, \dots, r_k\}$ that is equally spaced by the mean value $\bar{f}$, where the initial point $r_1 = R$. Figure 5 illustrates a simple SUS selection, with 8 individuals, where 10 of them will be selected. *Distance* in Figure 5 refers to $\bar{f}$, and pointer 1 through 10 is the set of fixed points (*P*). For each fixed point, a normal RWS is done to find the chromosomes. When all chromosomes have been selected, the set can be divided into two subsets. Then each individual of one set, can be paired with an individual from the second set. This is to reduce the chance of pairing two of the same chromosome. Another way is to shuffle the initial selected set, but the chance of pairing the same individuals is higher. With this approach, all parents will be generated in one pass, making the time complexity of this algorithm *O*(*n*), while also encouraging a more diverse selection, since less fitted chromosomes have a higher probability of being selected by the equally distributed pointers [23].

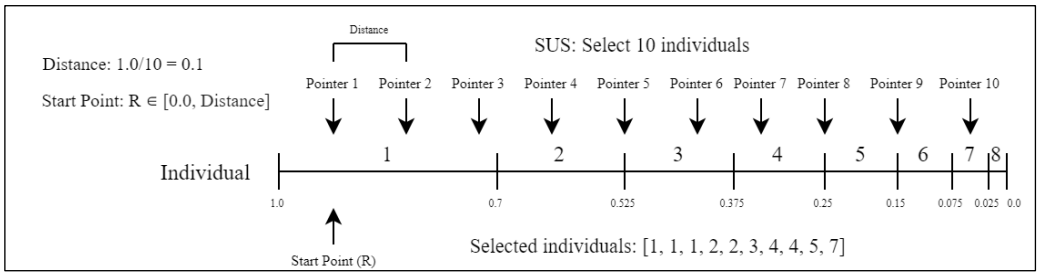

Figure 5. Illustration of a SUS selection

- ***Tournament Selection*** **(TOS)** selects *k* random chromosomes from the population, compares their fitness, and selects the best of that group as parent. This process is repeated *n* times until all parents have been selected. By choosing its competitors randomly, the algorithm puts a pressure on the selection, and the pressure can be increased by selecting a bigger number for *k*. The random selection also improves the diversity of succeeding generations. The time complexity of this operator is $O(n)$, since there are *n* iterations, and each requires a constant number of selections *k*.

2.4.3. Crossover

After the selection is done, all chosen parent chromosomes moves on to the reproduction stage [19], [22]. This is where all parents recombine in some way to create a new generation of chromosomes to be used in the next

iteration. Normally, two parents are combined to make one or two offspring. This is done by combining a randomly selected subset of the two parents. The process of combining two parent chromosomes is what is often referred to as *crossover*. The combination can be done in many different ways, and some of the most common crossover operators are: *Single-Point Crossover*, *N-Point Crossover*, and *Uniform Crossover*.

- ***Single-Point Crossover (SPC)*** [24], is done by selecting a random index (k) between 1 and the length of a chromosome (L), where k represents the crossover point, and is calculated by $k \in \{1, L\}$. Then, you take all the genes from indices 1, through k, from the first parent P_1, and add them to the offspring at the same indices. The missing indices are then taken from the second parent P_2, from indices k+1, through L. Single-Point Crossover creates two offspring this way, the second offspring is created by reversing the order of the parents, taking indices 1 through k from P_2, and the rest from P_1. This is illustrated by Figure 6, where the green genes come from P_1, and blue from P_2.

Figure 6. A simple illustration of Single-Point Crossover

- ***N-Point Crossover (MPC)*** is similar to Single-Point Crossover, the difference is that it uses N crossover points [24]. For a *Two-Point Crossover*, N=2, it selects the two random indices, $k \in \{1, L-1\}$, and $j \in \{k+1, L\}$. Then it creates the first offspring by taking the genes from indices k through j from P_1, and the rest from P_2. For the second offspring, as done in Single-Point Crossover, you reverse the order of the parents, giving it indices k through j from P_2, and the rest from P_1. This process is illustrated by Figure 7.

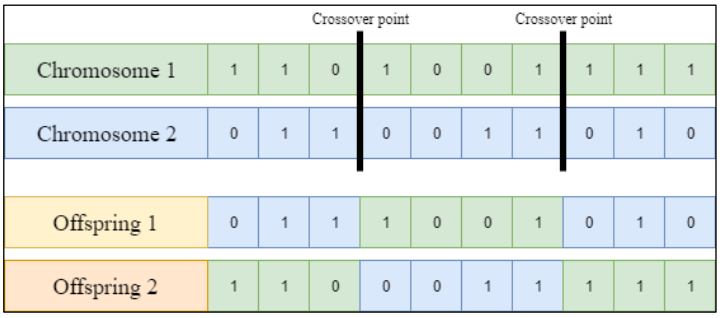

Figure 7. A simple illustration of Two-Point Crossover

- ***Uniform Crossover (UC)*** selects the genes at random from one parent, and takes the rest from the other [24]. This works by creating a string (R) of uniformly distributed random numbers, where the length of R is equal to the number of genes in a chromosome. Then, by using a predefined probability value $p \in \{0,1\}$, the algorithm goes through the generated string R, and if the value $R_i > p$, the first offspring get the gene at index i from P_1, and the second offspring get the same index from P_2. If $R_i \leq p$, the first offspring get index i from P_2, and the second offspring get index i from P_1. Each gene of the two offspring can therefore be calculated by Eq. (33), and Eq. (34), and this process is illustrated by Figure 8.

$$C_1^i = \begin{cases} P_1^i & if\ R_i > p \\ P_2^i & if\ R_i \leq p \end{cases} \quad (33)$$

$$C_2^i = \begin{cases} P_1^i & if\ R_i \leq p \\ P_2^i & if\ R_i > p \end{cases} \quad (34)$$

Chromosome 1 (C1)	1	1	0	1	0	0	1	1	1	1
Chromosome 2 (C2)	0	1	1	0	0	1	1	0	1	0
Random numbers string (R)	0.32	0.22	0.89	0.23	0.76	0.78	0.75	0.54	0.12	0.92
Probability ($p = 0.7$)	< 0.7	< 0.7	> 0.7	< 0.7	> 0.7	> 0.7	> 0.7	< 0.7	< 0.7	> 0.7
Offspring 1	0	1	0	0	0	0	1	0	1	1
Offspring 2	1	1	1	1	0	1	1	1	1	0

Figure 8. Simple illustration of Uniform Crossover

2.4.4. Mutation

The final step during recombination is *mutation* [19], [22]. Mutation exists to maintain a diversity between all chromosomes, and as a result, help preventing premature convergence. It is applied to each offspring after they are created in the crossovers step. Mutation usually have a very small probability (p_m) to occur for each individual gene of an offspring. This step also has a number of different operators to use, but for a binary encoded chromosome only one are worth mentioning, *Bit Inversion*.

- ***Bit inversion*** works by simply inverting the selected bits [19], [22]. As stated earlier, a predefined probability value (p_m) is used to select genes for mutation. A random number $r_i \in \{0,1\}$ is calculated for each gene, and if $r_i \leq p_m$ the gene at index (i) will be mutated, and the bit will be flipped from a 1 to a 0, or the other way around. Therefore, bit inversion can be represented by Eq. (35), and a visual example can be seen in Figure 9.

$$C_i = \begin{cases} mod(C_i + 1, 2) & if\ r_i \leq p_m \\ C_i & if\ r_i > p_m \end{cases} \quad (35)$$

Offspring prior to the mutation (Red genes = selected for mutation)										
Offspring	0	1	0	0	0	0	1	0	1	1
Random numbers (r_i)	0.32	0.22	0.036	0.0083	0.76	0.063	0.2	0.054	0.12	0.0036
Probability ($p_m = 0.01$)	> p_m	> p_m	> p_m	< p_m	> p_m	> p_m	> p_m	> p_m	> p_m	< p_m
Offspring after mutation										
Offspring	0	1	0	1	0	0	1	0	1	0

Figure 9. Simple illustration of Bit Inversion

2.4.5. Replacement

When the recombination stage is done, you end up with two different groups of chromosomes, the initial population, and the offspring generated during crossover and mutation [19], [22]. During replacement, the algorithm replaces the old population with the new. There are multiple ways of doing this, two being:

- ***Generational:*** A generational approach would be to replace all the chromosomes from the past generation with all offspring. All chromosomes from the past generation will therefore be discarded each iteration.

- ***Elitism:*** This approach keeps the *h* best fitted chromosomes from the previous generation (unchanged), and replace the *h* worst fitted offspring, which result in a combined population of old and new. The second approach is called *elitism* [25], and its intension is to make sure that the fitness of the population doesn't decrease over generations, and not waste time finding solutions that previously has been discarded.

2.4.6. Termination

When the new generation of chromosomes have replaced the old, the whole process repeats itself, and this process keep on going until a predefined, termination criteria is met [22]. There are different termination criteria depending on the problem the algorithm is supposed to solve, but in general, you have two options. One can be a predefined number of iterations the algorithm should run, the other, is that the algorithm stops when the population has fully converged. A population is said to have converged when 95% of the population share the same genes [22]. The phrase *premature convergence* has been used in earlier sections, and it means that the population have converged too early, and the solution quality end up being lower than expected, which gives the impression that the algorithm has gotten stuck since the solutions doesn't improve anymore.

2.5. Multi-Objective Optimization

In the previous section, Evolutionary Algorithms were presented, specifically Genetic Algorithm. A problem with the canonical GA introduced by Holland, is that it only solves one objective [26]. In reality, problems typically consist of multiple objectives, that may even be conflicting [26]. When objectives are conflicting, there are no solution where all objectives can be fully optimized at the same time. An example used in [26], is a real-world problem, where you want to minimize cost, and maximize both performance, and reliability. This is called a *multi-objective optimization problem* (MOP). This would be a hard problem for Holland's GA to solve, however it is possible to modify it to solve some of the more trivial MOPs, by combining the multiple objective functions, and using a weighted sum method to calculate the fitness. The problem with this method is that it can be hard to select the proper weights, and the algorithm would return a single solution rather than multiple solutions that can be examined for trade-offs.

For a more nontrivial MOP, we want the algorithm to return a set of *Pareto optimal* solutions [26], which are a set of solutions where none of the objective functions can be improved without decreasing the value of another, it also means that the solutions in this set are not *dominated* by any other solution. A non-dominated solution must meet the following requirements: a solution s_i dominates another solution s_j when it performs better than s_j on at least one objective, and doesn't perform worse on any of the remaining objectives. The following two equations describes what is needed for solution s_i to dominate solution s_j:

$$f_k(s_i) > f_k(s_j) \; for \; at \; least \; one \; objective \; k \in \{1, 2, 3, \dots, n\} \quad (36)$$

$$f_x(s_i) \geq f_x(s_j) \; for \; all \; other \; objectives \; x \in \{1, 2, \dots, k-1, k+1, \dots, n\} \quad (37)$$

Where *n* is the number of objectives. If solution s_i is not dominated by any other solution, it is defined as *non-dominated* [26]. Figure 10 shows an example of a MOP, and the pareto optimal solutions. The blue circles, placed on the black line are the Pareto optimal solutions, and as illustrated by the image, they are non-dominated. All the red circles are solutions that are dominated by the pareto optimal solutions. This is shown by the blue lines from solution *A*, where the resulting area is showing which solutions that are dominated by *A*. The black line going through all pareto optimal solutions, and separates them from all other solutions, is called the pareto frontier, and is the set of pareto optimal solutions.

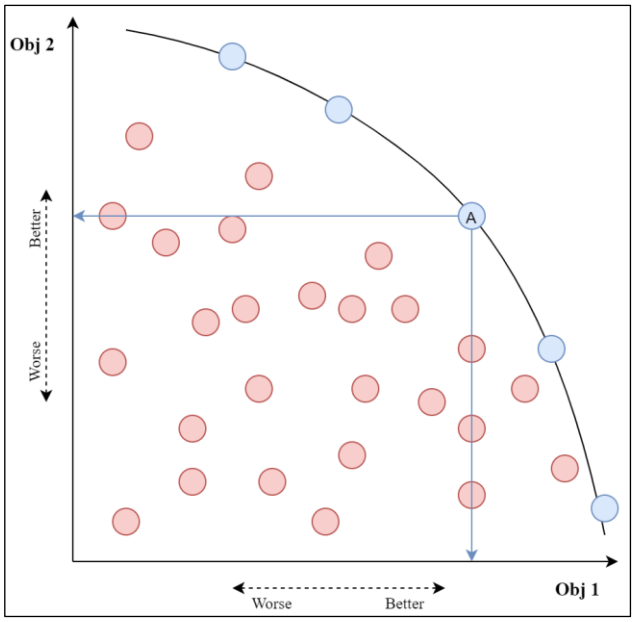

Figure 10. Illustration of the pareto front, and the pareto optimal solutions. All red circles in the area, or on the lines produced by the blue area, are solutions that are dominated by solution A.

To solve these kinds of problems, we need specialized algorithms, made for MOPs [26]. There are several versions, some of them being: *Vector Evaluated GA* (VEGA), *Multi-Objective GA* (MOGA), *Niched Pareto GA* (NPGA), *Non-dominated Sorting GA* (NSGA), *Fast Non-dominated Sorting GA* (NSGA-II). There are of course algorithms that are not based on GA, like *Strength Pareto Evolutionary Algorithm* (SPEA), and *Region-based Selection in Evolutionary Multi-Objective Optimization* (PESA-II), to name a few. The following section describes two of them, NSGA, and NSGA-II.

2.5.1. NSGA

Non-dominated Sorting Genetic Algorithm (NSGA) was introduced by Srinivas, and Deb, in [27], as a solution to MOPs. It is based on a previous Multi-Objective algorithm called Vector Evaluated GA (VEGA), by J. David Schaffer [28], and the idea of *non-dominated sorting procedure*, suggested by David E. Goldberg in [29].

The difference between Holland's GA, and NSGA, is the fitness distribution, and the selection operator [27]. Before selection, the population is being sorted based on Pareto optimality, into a hierarchy of subsets, called *fronts*, and follows the following structure:

i) Go through population and find all non-dominated solutions
ii) Move all non-dominated chromosomes to the current front (F_i)
iii) Set current front to F_{i+1}
iv) Repeat until the population is empty, and all chromosomes has been sorted into a front

The fronts then follow a hierarchical structure [27], front f_1 will contain the current Pareto optimal solutions, and F_2 the second best etc. Following this, a fitness value will be assigned to each front. The solutions in the best front F_1 will get a very large fitness value, and all solutions in the same front get the same value, to give each solution an equal probability of getting selected for reproduction. Each solution will also get a *shared* fitness value, which is calculated by dividing initial fitness value it got, by their *niche count*. Niche count is a value indicating how many solutions a solution has in the area surrounding it, within a maximum distance of σ_{share}. If solution C_j is in that area of solution C_i, it is said to be member of C_i's niche. The niche count for solution C_i is calculated by using Eq. (38), on every other solution in its assigned front, where j represents the index of another solution in the front, and d_{ij} is the distance between C_i, and C_j [30].

$$Sh(C_{ij}) = \begin{cases} 1 - \left(\frac{d_{ij}}{\sigma_{share}}\right)^2, & if\ C_{ij} < \sigma_{share} \\ 0, & otherwise \end{cases} \quad (38)$$

The purpose of the shared fitness value is to give a more diverse selection, and favor the less crowded areas [30]. This fitness distribution process, is repeated for every other front F_i to F_n, except that the initial fitness value all solutions in a front get, has to decrease for each front, and be smaller than the smallest fitness value in the previous front F_{i-1}, where $i \in \{2, n\}$, and *n* is the number of fronts [27].

When all chromosomes have been sorted into a front, and gotten their fitness values, the parent selection process begins. NSGA use a stochastic reminder selection, and since the chromosomes in the best front f_1 have the highest possible fitness, they will be selected more frequently, and with shared fitness, chromosomes residing in less dense areas will be favored more [27]. The recombination operators works just as Holland's GA does, as described in sections 2.4.3 and 2.4.4. Lastly, NSGA uses a generational replacement method, which is one of the differences with NSGA-II [27], [31].

2.5.2. NSGA-II

Non-dominated Sorting Genetic Algorithm – II (NSGA-II) is an improved version of NSGA, introduced by Deb [31], in 2000. Some of the biggest drawbacks of the original NSGA have been improved, and the three main points addressed are as follows:

- **High computational cost**: The original NSGA has a time complexity of $O(mN^3)$, where *m* is the number of objectives, and *N* is the population size. For NSGA-II, this has been improved to $O(mN^2)$.
- **Need of sharing parameter**: The need for a sharing parameter, σ_{share}, has been removed, and NSGA-II instead relies on a new way of calculating the density around each solution, namely *Crowding Distance*.
- **Generational replacement method**: After NSGA was introduced, scientists showed that an elitist method could be superior to a generational one, regarding performance, and as explained in section 2.4.6, elitism help making sure fitness doesn't decrease during generations, which is good for convergence. For those reasons, an elitist method has been introduced instead of the generational.

A number of things have been changed with the algorithm, to improve the criticisms stated above. An activity diagram of the algorithm can be seen in Figure 11:

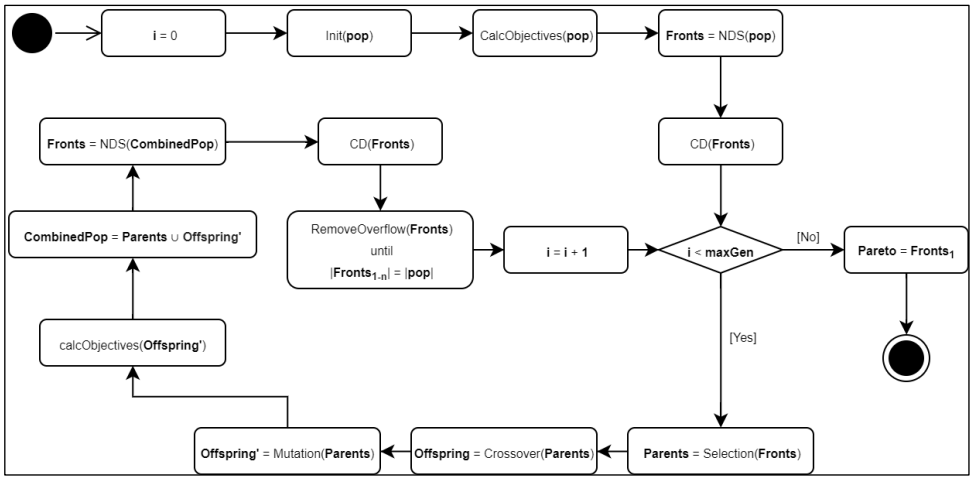

Figure 11. Activity diagram of NSGA-II

The algorithm starts by generating the initial population, as done by Holland's GA. Then it calculates the multiple objectives for every chromosome in the population. Before the algorithm enters the main loop, it sorts the population into fronts with *Fast Non-dominated Sorting* (NDS), and later calculates the *Crowding Distance* (CD) for each chromosome in the fronts. The remainder of this section is organized as follows: In Sections 2.5.2.1 and 2.5.2.2, NDS, and CD are explained, respectively. Finally, in Section 2.5.2.3, the main loop is explained.

2.5.2.1. Fast Non-dominated Sorting

The non-dominated sorting, that happens in the pre-selection stage, has been changed from the original NSGA, and relies on a completely different method [31]. The pseudo code is described by *Algorithm 1*. In the algorithm, S_x is a set belonging to chromosome *x*, where it stores all chromosomes it dominates. Variable n_x is an integer which represents how many solutions *x* is dominated by.

Algorithm 1 [31]: Fast non-dominated sorting

Input: a population $P = \{c_1, c_2, \dots, c_n\}$ where *c* is a chromosome, and *n* is the total number of chromosomes

```
for each x ∈ P
    for each y ∈ P
        if x dominates y then
            S_x = S_x ∪ {y}
        else if y dominates x then
            n_x = n_x + 1
    if n_x = 0 then
        Fronts_1 = Fronts_1 ∪ {x}
i = 1
while Fronts_i ≠ []
    H = []
    for each x ∈ Fronts_i
        for each y ∈ S_x
            n_y = n_y − 1
            if n_y == 0 then
                H = H ∪ {y}
    i = i + 1
    Fronts_i = H
```

The algorithm saves time by only having to do the domination calculation once [31]. By using the set *S*, and integer *n*, it always knows which chromosomes dominates which, and how many solutions it is dominated by. When the current non-dominated chromosomes get moved to a front, the algorithm looks at the non-dominated chromosomes' *S* sets, to find all chromosomes that will be evaluated for next front. Then it decreases the *n* variable for chromosomes in *S*, and look for the solutions which are not dominated anymore ($n = 0$). In comparison the original NSGA have to exclude the current non-dominated solutions, and then recalculate which chromosomes that dominates which each iteration.

When this is done, it returns all fronts [31], where F_1 contains the non-dominated chromosomes, and F_2 the second least dominated chromosomes, etc. Each chromosome will be assigned a rank value, which is the number of the front it's assigned to, better solutions will therefore have a lower rank value, with 1 being the best.

2.5.2.2. Crowding Distance

Crowding distance is a part of NSGA-II where it calculates how many chromosomes there are in a particular chromosome's surrounding area [31]. It is calculated for each chromosome, by finding the two closest solutions in its assigned front, and calculating the average distance to them along the objective axis. This is illustrated by the pseudo-code in *Algorithm 2*.

Algorithm 2 [31]: Crowding Distance

Input: a front $F = \{c_1, c_2, \dots, c_n\}$ where *c* is a chromosome, and *n* is the number of chromosomes in that front

```
for each c ∈ F
    c_cd = 0
for each objective o ∈ O
    F = sort(F, o)                  sort F according to objective o
    F[1]_cd = ∞
    F[end]_cd = ∞
    for i = 2 to (|F| - 1)
        F[i]_cd = F[i]_cd + (F[i+1]_o − F[i−1]_o) / (F[end]_o − F[1]_o)
```

The area calculated by Crowding Distance is shown in the following figure:

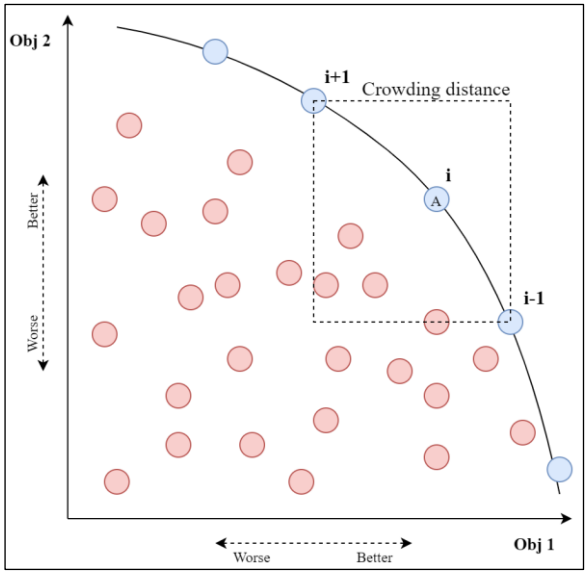

Figure 12. Crowding distance for chromosome *i*, where the blue circles are assigned to the same font. The dotted area is the calculated bounds of the crowding distance.

2.5.2.3. Main loop

As seen in Figure 11, when the initial NDS and CD have been calculated, the algorithm, enters the main loop. This is started by the breeding operators. For the selection process, it now uses the tournament selection operator [31], which is slightly modified to make use of both the rank value, and the crowding distance. In the tournament, the solution with lower rank gets selected, and if two solutions share the same rank, the crowding distance will determine the selection, where the algorithm favors a larger crowding distance (less dense area). That is, to make the new generation converge towards the lower ranked chromosomes, with the bigger crowding distance (less dense area).

After all parent chromosomes are selected, the recombination works just as NSGA and Holland's GA, with the crossover, and mutation operators [31]. The last difference is the recombination part, which now uses elitism instead of the generational replacement method ,which means that the top chromosomes from last generation will be carried over to the next one [31]. This can be seen in Figure 11, where the parents are combined with the offspring, then after NDS and CD have been calculated, the lowest ranked chromosomes gets removed until the number of chromosomes is the same as initially. When the algorithm terminates, the current Pareto frontier, front F_1, will be returned [31].

3. Related Work

Feature selection is an important step regarding the speed of classification. But searching for combinations with smaller amount of selected features, directly conflicts with the objective of keeping a high accuracy. Multiple studies have been done to solve this problem, to find a search algorithm that can balance the two objectives efficiently. This section describes some of the related studies on the subject, and how they relate to this study.

The following section is organized as follows: In Section 3.1, the related studies, and the used algorithms are introduced. In section 3.2, the different individual representations used, are presented. In Sections 3.3-3.4, the breeding operators, and objectives used in the related studies, are presented, respectively. Finally, in Section 3.5, the classifiers used in the related studies are presented.

3.1. Related Studies

In [7], they conducted an experiment where they classified EEG readings between an MI activity, and resting state. For feature reduction, they created a novel representation of an individual for a single-objective GA, called *Hierarchical Genetic Algorithm* (HGA). They represent the channels, frequencies and their corresponding time steps in a layered structure, <channel, frequency, time step, power-value>. A feature is the power-value recorded from one of the channels, at one of its frequencies, at a given time step. The EEG data were recorded from six healthy subjects. For each subject, 42 channels, 30 frequencies, and 40 time steps were used in the classification, in total, 50400 features. The team proposed a method of reducing the number of time steps, by averaging consecutive time steps, into time windows. Instead of using all 40 time steps for each frequency, they could have a window size of 40, which would average all 40 time steps into only one time window, and result in only 1260 features. This method was tested with different window sizes, 40, 20, 10, and 5. Then they applied HGA on that number of features to reduce the number even further. They found that by using HGA-SVM they could lower the computational time by 98.92% compared to SVM without feature reduction, with a feature reduction of between 77%, and 91%. This, while still maintaining a high prediction accuracy, between 69.03%, and 76.04%, compared to the 79.45% without feature reduction.

A continued study of [7] was done in [9], where the author implemented a MOEA, instead of the single-objective HGA, as feature selection algorithm. The MOEA chosen was NSGA-II, which he modified to fit the individual representation from [7]. More objective functions were also added, and the study compared their performance regarding feature reduction, and accuracy. The MOEA, and the implemented objectives, were all tested with the same dataset as in [7], and the results found was compared to their results. The author found that NSGA-II, regarding features selected, used considerably less features, which is expected since a MOEA is more efficient than a single-objective algorithm at optimizing multiple objectives [26]. However, the average accuracy was lower than what was reported in [7].

In [11] the authors did a comparison between a MOEA and two *Single-Objective Evolutionary Algorithms* (SOEA), for feature reduction related to EEG classification. They studied the relationship between the algorithms, and the proposed objective functions, with a focus on the benefit of using a MOEA. The experiment was done on data recorded from 20 undergraduate students, where they classified between resting state, and three memory tasks. They used NSGA-II as MOEA, and classic GA as SOEA, with a vectorized individual representation. The results they found were that both SOAEs used a similar number of features, but many more than NSGA-II. NSGA-II also preserved most of the accuracy with its selected features, which the SOEAs didn't. They found that the SOEAs seemed to not balance the objectives efficiently, especially when compared to the result of NSGA-II, most likely because NSGA-II is able to optimize the two objectives simultaneously, without combining them with linear combination [26].

Another study, [6], also used NSGA-II for feature selection. However, they proposed a different individual representation compared to the other studies. Because of the representation, they had to make a lot of changes to the breeding operators, which is discussed in Section 3.3. In the experiment they were classifying between three MI activities of right hand, left hand, and left foot, movements. The EEG data was recorded from 12 healthy subjects. The result they found was that their wrapper, compared to a baseline given by 240 LDA classifiers (no feature selection), and a filter method called FOPT* [32], outperformed them both in execution time, accuracy, and feature reduction.

Although the average accuracy was lower with NSGA-II in [9], compared to [7], the algorithm seems promising for a feature selection problem when considering the results in [6], and [11]. Regarding features used, [9] still shows promising results since NSGA-II used less features than HGA.

3.2. Individual Representation

Three different individual representations have been used in these previous studies, two binary encoded representations, and one encoded with integers. One representation is the vectorized version from [11], where all features were represented in a binary array, where a 1 is a selected feature, and a 0 is a non-selected feature. A solution can therefore be expressed as: $C = \{x_1, x_2, \dots x_M\}$, where C is an individual, x_i is a binary number representing whether a feature is selected or not, and M is the number of features.

Both [7] and [9] make use of the layered individual representation, originally created for HGA by the authors in [7]. It consists of three layers, channels, frequencies, and time windows (which contains the averaged power-value). These layers are each represented in binary. The first layer is all the channels, and are represented as $X = \{x_1, x_2, \dots x_n\}$ where n is the number of channels, and $x_i \in \{0, 1\}$, which corresponds to whether a channel is selected or not. Each channel then has a set of frequencies, represented as $Y = \{y_1, y_2, \dots y_m\}$, where m is the number of frequencies for each channel, and $y_j \in \{0, 1\}$. All frequencies have an additional layer of time windows, which are represented as $Z = \{z_1, z_2, \dots z_t\}$, where t is the number of time windows for each frequency, and $z_k \in \{0, 1\}$.

A drawback with the above representations is that they have to store all features, selected or not. Since the classification usually only need a small subset of features, most of the data won't be selected, and will be represented by zeroes. This will take up a lot of unnecessary memory. This problem is addressed in [6], where they have created a novel individual representation, and used it together with a modified version of NSGA-II. Instead of representing all features as binary numbers, the team in [6], only store the indices of selected features, with integers pointing to a specific index of a feature, instead of binary numbers. They have also added a limit to the features, where an individual never can have more selected features than the predefined limit. This is to both reduce the memory usage even further, and to speed up the evaluation since only a limited number of features will be processed. An individual can then be described as $I \subset \{x \in [0, n) \cap \mathbb{N}\}, \quad |I| \leq s$, where n is the number of features, and s is the maximum number of features an individual can have.

The vectorized representation from [11], and the hierarchical version from [7] and [9], have both been used for SOEAs and MOEAs. The authors from [11], tested their representation for both NSGA-II, and Holland's GA, while the authors from [7] proposed the hierarchical representation for their SOEA, HGA. Lastly, the author from [9], continued the hierarchical solution by implementing it for a MOEA, NSGA-II.

3.3. Breeding Operators

The breeding operators are used to create the new population for next iteration. It is a process that consists of three different parts, selection, crossover, and mutation. There are different algorithms that can be used for each part, and each of the previous studies talked about in this paper, used a different combination of breeding operators.

- **Selection:** The authors of [7] used RWS to select the parents in their HGA, while the team in [11] used SUS for their SOEAs. What's interesting is that all of the MOEAs have been using the same selection method [6], [9], [11], which is Tournament Selection, and it is the selection method used in the original NSGA-II algorithm in [31].

- **Crossover:** For the SOEAs, Uniform Crossover was used in [7], and Multi-Point Crossover was used in [11]. The MOEAs in [9], and [11], both used Uniform Crossover, while the MOEA in [6], used a novel method. Their novel crossover operator uses a pair of two parents to generate two offspring. This works by taking all common features of the two parents, and copy the full subset to both offspring. Then they take all the non-common features from both parents, and distributes them randomly to each offspring, so that $|p_1| = |o_1|$ and $|p_2| = |o_2|$ where p represents the parents, and o the offspring.

- **Mutation:** Since all but one study uses a binary encoding for their representations, they all use the same mutation operator, Bit-Inversion. The only exception is in [6], where they created their own novel operator. The mutation operator works by selecting a small subset of features, as you do normally with a slight probability of selecting a feature. Then they divide this subset into two smaller subsets, where one subset of features will get removed from the individual, and the others are replaced with new random features that currently aren't selected by this individual. In addition to that, they've added a small

probability of appending an additional random, non-selected feature. This is to give each individual an opportunity to increase their selected features.

All the different breeding operators were of course modified slightly to work with their different representations. The authors in [6] proposed new crossover, and mutation operators, because their representation doesn't work for the more commonly used techniques. As with the different representations, breeding operators are also an interesting topic, whether different combinations produce a noticeable difference in the results when used for feature selection. Some operators were only tested with one representation, and some operators like RWS, SUS, and Multi-Point crossover haven't been evaluated for a MOEA.

3.4. Objectives

To guide the selection algorithms towards small sets of relevant features, many different objective functions have been studied. Most noticeable in [9] where the author evaluated 6 combinations of objectives for the proposed hierarchical NSGA-II.

The authors in [7], proposed a fitness function which combines percentage of feature reduction (FR), with a correlation between the selected features (CFS), using *Pearson's Correlation Coefficient* (PCFS). Since HGA is a single-objective algorithm, the two objectives had to be combined using linear combination. That resulting combination was then called PCFS + FR.

Two other objectives were proposed in [11], *Most Relevant Features* (MR), and *Number of Features* (NF). They are calculated by the following two equations:

$$MR(C_i) = \frac{N - CC(C_i)}{N} \quad \textbf{(39)}$$

$$NF(C_i) = \sum_{j=1}^{M} x_i \quad \textbf{(40)}$$

Where N is the number of samples, and CC(C_i) is the number of correct classified patterns with an SVM classifier with RBF kernel, and one vs one mechanisms [33], and M is the number of features in subset C_i. EQ. (39) guides the solutions towards higher accuracy, while EQ. (40) guides them towards smaller numbers of features. For the two SOEAs used, one only optimizes EQ. (39), and was called SOO-ER, the other was named SOO-AG, and it worked with the linear combination between EQ. (39), and EQ. (40).

All three objectives above have also been evaluated with NSGA-II. In [11], they used ER, and SF, without linear combination, since NSGA-II is a MOEA, and can optimize both EQ. (39), and EQ. (40), simultaneously. Regarding PCFS + FR from [7], it was evaluated for NSGA-II in [9], where the author also added more objective functions to the algorithm to compare against the previous PCFS + FR combination. Since he also changed from SOEA to MOEA, he could change PCFS + FR, and evaluate the two objectives respectively, without linearly combining them. He added another CFS, with *Spearman's Correlation Coefficient* (SCFS), and he added wrapper objectives using SVM as classifier. Since two objectives are used, he could combine all the objectives, to six different pairs, PCFS + FR, PCFS + SVM, PCFS + SCFS, SCFS + FR, SCFS + SVM, and SVM + FR.

In [6], the authors used two objective functions for their version of NSGA-II, calculated with Cohen's Kappa [34], and they calculate the Kappa index from EQ. (41):

$$\kappa = \frac{p_o(\ell, D) - p_e(\ell, D)}{1 - p_e(\ell, D)} \quad \textbf{(41)}$$

Where $p_o(\ell, D)$ is the classification accuracy between the classifier ℓ, and the labeled data in D, and $p_e(\ell, D)$ is the probability that a correct classification happened by chance [6]. The Kappa index is then used to calculate the two objectives. One of the objectives (o_1), guides the solutions towards the global optimum, and the second (o_2), prevents over-fitting. The original set (D) is randomly divided into two subsets, training (D_{train}), and validation (D_{valid}), where the size of D_{valid} is a predefined percentage (p_{val}) of D. The classifier (ℓ) is then trained on the data from D_{train}, before calculating the two objectives with EQ. (42), and EQ. (43):

$$o_1 = \kappa(\ell, D_{test}) \quad (42)$$

$$o_2 = \kappa(\ell, D_{valid}) \quad (43)$$

All objective functions have been evaluated for NSGA-II, but not compared with different individual representations, and breeding operators. Much like [9], this study aims to evaluate multiple objectives. The author from [9] found that the highest average accuracy came from using PCFS – SVM, a combination of a filter, and a wrapper objective. Regarding feature reduction, SVM – FR achieved the highest average reduction with 94%, with a window size of 5. But at higher windows sizes (40, and 20), PCFS – FR, and SCFS – FR achieved the highest reduction with 91%. With the Kappa index from [6], they found that KNN, LDA – KNN, and LDA - NBC achieved similar results regarding accuracy. All tested objectives were able to find feasible solutions with sub 30 features selected.

3.5. Classifiers

After feature selection the selected subset of features gets classified with a ML algorithm using Supervised Learning. Different feature selection algorithms, and objective functions can produce different results for different classifiers. Because of this, multiple classifiers have to be considered, and evaluated.

The most common classifier used in these studies were SVM [7], [9], [11], because it works well with noisy data, and it doesn't require any parameters when used with linear kernel as in [7], [9]. However, in [11], they used a different kernel, namely RBF. They also used a second classifier, *Random Forest* (RF), which were used together with SVM for their objective function. They reported that the mix produced a very high accuracy, compared to using SVM post-feature selection, and for objective functions.

In [6], they tested their wrapper with *Linear Discriminant Analysis* (LDA), *k-nearest Neighbors* (KNN), *Naive Bayesian Classifier* (NBC), LDA + KNN, and LDA + NBC. As stated in Section 3.4, they found that KNN, LDA – KNN, and LDA - NBC achieved similar results regarding accuracy. When looking at the stability scores, KNN achieved the highest, and was very close to perfect stability. However, KNN had the worst performance regarding execution time. They further concluded that KNN is a good choice for MI classification [6].

4. Problem Formulation

Feature selection is an important step for classifying EEG data [6]. We want the classification system to be able to classify the input data as fast as possible, but we also want the accuracy of the classification to be as high as possible. This is especially important for real-time applications, where the speed of the classification is of big importance. To solve the speed problem, we want to feed the classifier only a small sub-portion of the original input data. By doing that, the classification system will have less data to process, and will therefore classify it quicker. The problem is that the right data have to be selected, in order to not make the prediction accuracy suffer when classified. We want to remove all data that is not relevant to the current classification.

NSGA-II has shown to be a promising EA for feature selection [6], [7], [9]. But as it belongs to the GA family there are a number of different ways it can be implemented. The representation of an individual, parent selection method, and breeding operators can all be interchanged to a number of different combinations. So far, there hasn't been a comparison between the different combinations, and how they affect the different classifiers.

The purpose of this study is to find what representation, parent selection method, breeding operator, and classifier is the best combination for NSGA-II, when it comes to least number of features used, and best prediction accuracy. The study considers *prediction accuracy*, and *number of features used*, as measure of performance, where we want to maximize the prediction accuracy, with as small of a feature-set as possible. That is because the computational time we want to minimize depends on number of features used [6], [7]. We also need to test all these combinations with the different objective functions that have been used in the previous studies [7], [9], since new representations, and breeding operators have not previously been examined for those functions. For that we need to investigate, (1) how each combination of *individual representation*, *breeding operators*, and *objective functions*, affect the prediction accuracy and feature reduction, (2) how do the different combinations compare to each other, (3) what combination of *objective functions* are best suited for each individual representation, regarding prediction accuracy and feature reduction, (4) how does each classifier perform, for each of the combinations. The research questions are therefore defined as:

- *How will the different combinations of individual representation, breeding operators, and objective functions, perform regarding prediction accuracy and number of used features as feature selection in classifying EEG oscillations in MI?*

- *How will the different classifiers affect the performance for each of the NSGA-II combinations when classifying EEG oscillations in MI?*

4.1. Limitations

This study will only research the algorithm NSGA-II for feature reduction, and together with the EEG data itself, they will both put limitations on how an individual can be represented. Also, each individual representation won't work with every breeding operator, the representation from [5] will for example need two special breeding operators to work correctly. This will limit the amount of combinations that can be evaluated. The same goes for all the objective functions, and classifiers. Previous studies in [35], explains that the EEG samples are subject-specific, and we only have samples from six people which can limit the conclusion.

5. Method

This is mostly an empirical study. Initially, it will be a literature study to find information regarding NSGA-II, the different individual representations, and what breeding operator techniques that can work for each representation. The González paper [6], will especially need to be studied since it outlines one of the individual representations, with its corresponding breeding operators that will be implemented, and tested.

After the literature study, the implementation will start, and all chosen individual representations, breeding operators, and selection methods will be implemented. Then, the process off testing all the combinations of individual representations, breeding operators, and selection methods will start. Each of those combinations are going to be tested for each of the objective functions, and classifiers. The objective functions will be:

- Pearson – FR (PCFS-FR)
- Pearson – KNN (PCFS-KNN)
- Training-validation percentage with KNN (KNN_tra-KNN_val)

And the evaluated classifiers are:

- Linear Discriminant Analysis (LDA)
- Support Vector Machine, with linear kernel (SVML)
- Support Vector Machine, with polynomial kernel (SVMP)
- Artificial Neural Network (ANN)
- K-Nearest Neighbors (KNN)

Each combination will be tested on six data sets, provided by the authors in [7], which are recorded EEG data from six different subjects. This will also add to the complexity of the testing phase, since every combination have to be run on all of the data sets. It is a lot of combinations to work through, which will take a lot of time to test since each individual test is estimated to run for a long time. Each test will go through the classification process, and generate prediction accuracy, number of features used, and computational time, which will be compared to each other, and compared with the results from both reference papers [7], [9]. The result of the data comparisons will answer both research questions. To ensure validity of the test results a 10-fold cross validation will be used.

6. Ethical and Societal Considerations

The EEG data used are recorded from real people and are provided by [7]. The samples are completely anonymous, there are no information that can be tied to the subjects, names are replaced with “Subject A-F” before I get the data. No other confidential data will be dealt with.

7. Design and Implementation

In this section, a description of the design, and implementation is presented. Each subsection will describe one part of the algorithm. It is organized as follows: In section 7.1, the design regarding the four representations are presented, including the pseudo-code for the algorithms. In section 7.2, a description of the used breeding operators, with corresponding pseudo-codes, are presented. In section 7.3, the tested objective functions are presented, and finally, in section 7.4, the used classifiers, are motivated, and presented.

The data used in this study was recorded, and processed by the team in [7]. It is recorded from six healthy subjects, and each sample consists of 42 channels, $42 \cdot 30$ frequencies, and $42 \cdot 30 \cdot 40$ time steps, in total, 50400 features. As described in section 3.1, they use a method of reducing the number of time steps, by averaging consecutive time steps, into time windows. Since this study is a continuation of their study, I've also used that processing step, with a window size of 40. This window size will average all 40 time steps into only one time window, and result in only 1260 total features.

In this study, the classifiers are classifying between two classes, which is the same two classes as the reference studies, [7], and [9]. The two classes are:

- **MI Activity:** The MI activity consists of either opening, or closing their hand.
- **Resting State:** The state before, and between the MI activities, where the subject didn't perform any MI activities.

7.1. Representations

In this section, the four different individual representations are presented, with their corre. In section 7.1.1, the Hierarchical Representation (HR), and its corresponding pseudo-code is described. In section 7.1.2, the Non-Hierarchical Representation (NHR), is described, and motivated, and finally, in section 7.1.3, a description of the design, and pseudo-code, regarding the two González representations (G30, & G600) are presented.

7.1.1. Hierarchical Representation (HR)

The hierarchical representation is proposed in [7]. Here the channels, frequencies, and time windows are placed in different layers. The first layer represents the channels, and are implemented as a vector. The frequencies are represented in a matrix, where each row contains a specific channel's frequencies. The same goes for all the time windows, which is also represented by a matrix, where each row is a specific frequency's, associated time windows. *Figure 13* illustrates this representation, and the corresponding layers:

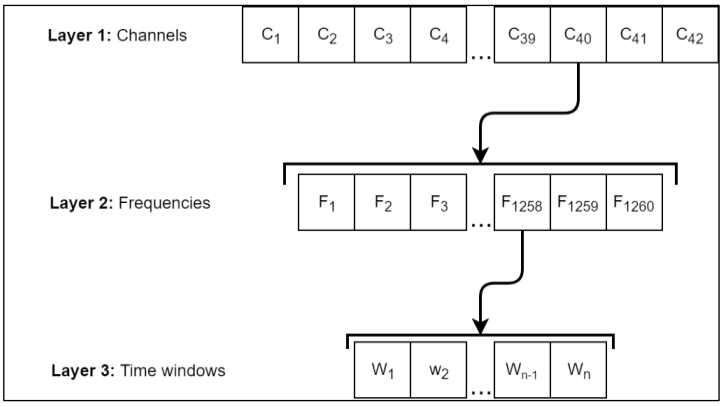

Figure 13. Illustration of the hierarchical representation

As explained in section 3.1, the hierarchical can be illustrated in vectorized form as:

Layer 1 $$X = \{x_1, x_2, \dots x_n\} \; where \; x \in \{0, 1\}, and \; n = 42$$ **(44)**

Layer 2 $$Y_i = \{y_1, y_2, \dots y_m\} \; where \; y \in \{0, 1\}, and \; m = 1260$$

Layer 3 $$Z_{ij} = \{z_1, z_2, \dots z_t\} \; where \; z \in \{0, 1\}, and \; t = 1260$$

Where X is the channel vector, Y_i is the frequency vector, for the ith channel, and lastly Z_{ij} is the time window's vector, for the ith channel, and jth frequency. In summary, this representation requires a vector of 42 channels, and two matrices of 1260 frequencies, and 1260 time windows. In total, $42 + 1260 + 1260 = 2\,562$ binary values need to be stored, for each individual.

7.1.2. Non-hierarchical Representation (NHR)

In the non-hierarchical representation, all features are represented in a single, continuous space. A feature in this representation is essentially a frequency, represented by a binary number, that corresponds to whether it is selected, or not. Since I use a window size of 40, and all frequencies have one power value, the number of features that this representation needs to store for each individual, is only the number of frequencies, 1260. The Non-Hierarcical representation is therefore implemented using a single, binary vector, which stores all the features for an individual. A non-hierarchical individual (I_i) can therefore be defined as EQ. (45).

$$I_i = \{x_1, x_2, \dots x_n\} \; where \; x \in \{0, 1\}, and \; n = 1260 \quad \textbf{(45)}$$

7.1.3. Gonzalez Representation (G30 & G600)

In [6], the representation only stores the indices of selected features, with integers pointing to a specific index of a feature, instead of a binary number for every single feature. Just like the non-hierarchical one, it only stores the frequencies, which can be represented by a vector of integers in this case. But since this vector only stores the selected indices, it doesn't require as much memory as the other representations, since it doesn't have to store any data about non-selected features. Similar to the non-hierarchical representation, it can be defined as EQ. (46).

$$I_i = \{x_1, x_2, \dots x_m\} \; where \; x \in \{1, n\} \quad \textbf{(46)}$$

In EQ. (46) n is the total number of features, which is 1260, and m is the current number of selected features for individual I_i.

Two versions of the González representation have been designed for this study, the difference is how many features one individual can select as a maximum. The original algorithm from [6], uses a maximum of 30 features (G30), while the other version in this study is using a maximum of 600 selected features (G600).

7.2. Breeding Operators

In this section, the three different breeding operators are presented. Section 7.2.1 presents the selection operators, section 7.2.2 the crossover methods, and finally, section 7.2.3 describes the mutation operators.

7.2.1. Selection

To select individuals, three selection methods were implemented, and evaluated. Those were RWS, SUS, and Tournament Selection. The theory required for each of them is explained in section 2.4.2. In this section I will present the pseudo code, as they are implemented.

Tournament selection is implemented according to the original NSGA-II implementation from [31], and is used in both [7], and [9]. Therefore, I refer the reader to [31] for an in-depth read on the implementation of this algorithm.

Since NSGA-II normally is rank-based, both RWS and SUS has to be modified to make use of the front rankings. For this reason, RWS has two roulette wheels, one to select the front, and one to select an individual from the selected front. A fitness value can be calculated from the rank of a front, or individual, by calculating:

$$2\frac{rank}{N(N+1)} \quad \textbf{(47)}$$

The modified RWS algorithm initially calculates the fitness of each front, from their respective rank. Then it goes in to the main loop, where it does an RWS to select a front, based on the initial fitness values of each front. When a front is selected, it calculates a fitness value for each individual in the selected front, with the same equation. Then another RWS selection is done, to select one of the individuals from the previously selected front, and it later adds the selected parent to the parent vector. Since each couple of parents, will create one offspring, I had to select $2 \cdot population$ parents. The modified RWS selection, and the pseudo-code is presented in *Algorithm 3*:

Algorithm 3: Roulette Wheel Selection

Input: all fronts $F = \{f_1, f_2, \dots, f_n\}$ *where* $f_i = \{c_1, c_2, \dots, c_n\}$ *and* c_i *is an individual*

for each $f \in F$
  $frontFitness_i = \frac{f_{rank} \cdot 2}{numFronts \cdot (numFronts + 1)}$

for k = 1 to $(2 \cdot population)$
  $number = rand(0, 1)$
  for each $ff \in frontFitness$
    if $ff_i < number$ and $ff_{i+1} > number$
      $currentFront = F_i$

  for each $c \in currentFront$
    $solFitness_i = \frac{i \cdot 2}{numSolutions \cdot (numSolutions + 1)}$

  $number = rand(0, 1)$
  For each $sf \in solFitness$
    if $sf_i < number$ and $sf_{i+1} > number$
      $parents_k = currentFont_i$

return $parents$

For SUS, I assigned each individual a personal rank, where the best individual at the top of the first front, has rank 1, and the last individual in the last front gets rank 15 (since I use a population of 15). Then calculate the rank fitness with EQ. (47), and the cumulative sum of those results. The algorithm calculates a pointer for each individual to be selected, then does an RWS selection on all pointers, to select every parent in one pass. The pseudo-code for this SUS version is described in *Algorithm 4*:

Algorithm 4: Stochastic Universal Selection

Input: all fronts $F = \{f_1, f_2, \dots, f_n\}$ *where* $f_i = \{c_1, c_2, \dots, c_n\}$ *and* c_i *is an individual*

for $i = 1$ to $popSize$

$$popFitness_i = \frac{i \cdot 2}{popSize \cdot (popSize + 1)}$$

$$cumulativeSum = cumsum(popFitness)$$

$$pointerDistance = \frac{sum(cumulativeSum)}{2 \cdot popSize}$$

$$startPosition = pointerDistance \cdot rand(0, pointerDistance)$$

for $i = 1$ to $(popSize \cdot 2)$

$$pointers_i = startPosition + (pointerDistance \cdot i)$$

for each $p \in pointers$

for each $c \in cumulativeSum$

if $c < p$

$$parents_i = population_j$$

return $suffle(parents)$

7.2.2. Crossover

The four crossover methods used are Uniform (UC), Single-Point (SP), Multi-Point (MP), and the special case for the González representation (GC). The theory behind the first three is described in section 2.4.3, while the Gonzalez one is described in section 3.3. In this section I present the pseudo code to the algorithms.

The UC algorithm generates two offspring from a single set of parents. A random number is generated for each feature index *i*. If the number is bigger, or equal, to the crossover probability, the feature at index *i* of *offspring1* will be inherited from parent *A*, and *offspring2* will inherit feature *i* from parent *B*. If the random number is less than the crossover probability, *offspring1* will inherit *i* from parent *B*, and *offspring2* from parent *A*. The pseudo-code for UC is presented in *Algorithm 5*:

Algorithm 5: Uniform Crossover

Input: two parents, A and B

for i = 1 to $NumberOfFeatures$

if rand() >= $CrossoverProbability$

$offspring1_i = A_i$

$offspring2_i = B_i$

else

$offspring1_i = B_i$

$offspring2_i = A_i$

The SP algorithm initially chooses a point, to decide what part of each parent to give to each offspring. The point is chosen between index 2, and the number of features. *Offspring1* then gets all features between indices 1, and the chosen point - 1 from parent *A*, and *offspring2* gets the same indices from parent *B*. The rest of the features are then inherited from the second respective parent, where *offspring1* inherits them from parent *B*, and *offspring2* from *A*. The pseudo-code for SP is presented in *Algorithm 6*:

Algorithm 6: Single-Point Crossover

Input: two parents, A and B

point = rand(2, $NumberOfFeatures$)

for i = 1 to *point-1*
    $offspring1_i = A_i$
    $offspring2_i = B_i$

for i = *point* to $NumberOfFeatures$
    $offspring1_i = B_i$
    $offspring2_i = A_i$

MP crossover is similar to SP, and is simply using more points to divide the parents. In this study I used two points, where two parts of an offspring will be inherited from one parent, and one part from the second parent. Otherwise it is identical to the single-point implementation. The pseudo-code is presented in *Algorithm 7*:

Algorithm 7: Multi-Point Crossover

Input: two parents, A and B

point1 = rand$\left(2, \ \frac{NumberOfFeatures}{2}\right)$
point2 = rand$\left(\frac{NumberOfFeatures}{2}, \ NumberOfFeatures - 1\right)$

for i = 1 to *point1-1*
    $offspring1_i = A_i$
    $offspring2_i = B_i$

for i = *point1* to *point2*
    $offspring1_i = B_i$
    $offspring2_i = A_i$

for i = *point2+1* to $NumberOfFeatures$
    $offspring1_i = A_i$
    $offspring2_i = B_i$

Algorithm 5-7 can run directly for the non-hierarchical representation. However, for the hierarchical representation, each crossover algorithm has to run multiple times for each layer, once for the channel vector, and once for each row in the frequency, and window matrices.

Lastly, the González representation need a special crossover, due to it using integers to store selected features. It distributes the selected features that are common in both *A*, and *B*, to both offspring. It then distributes the features they didn't have in common randomly to both offspring, so that *offspring1* ends up with the same length as parent *A*, and *offspring2* with the same length as parent *B*. The pseudo-code is presented in *Algorithm 8*:

Algorithm 8: Gonzalez Crossover

Input: two parents, A and B

$C = A \cap B$	% Get the selected features they have in common
$R = (A \setminus C) \cup (B \setminus C)$	% Get the selected features they don't have in common
$o1\_remaining = \|A\| - \|C\|$	% How many more features to reach the same length as *A*
$R1$ = shuffle(R)	% Shuffle *R* to get a random order of the non-common features
$offspring1 = C \cup R1_{1-o1\_remaining}$	% Add the common, and some of the non-common features % to reach the same length as parent *A*
$o2\_remaining = \|B\| - \|C\|$	% Do the same for the second offspring, but with parent *B*
$R2$ = shuffle(R)	
$offspring2 = C \cup R2_{1-o2\_remaining}$	

7.2.3. Mutation

Two mutation methods are evaluated in this paper, Bit-Inversion (BM), and the one proposed in [6], which is the special case for the González representation (GM). In section 2.4.4, the pseudo-code for BM is presented, and in section 3.3, a description, and pseudo-code of GM is presented.

The BM algorithm simply flips the bits, if a random number for the current index is less than the mutation probability. However, as the crossover methods, it has to run multiple times for the hierarchical representation, once for the channel vector, and once for each row in the frequency, and window matrices. The difference between each run is the mutation probability, which has a different value for each layer. The settings used in this study can be found in section 8. The pseudo-code for BM is presented in *Algorithm 9*:

Algorithm 9: Bit-Inversion Mutation

Input: One offspring, O

```
for i = 1 to length(O)
    if rand() < mutationProbability
        O_i = mod(O_i + 1, 2)          % Flip the bit from 0 to 1, or from 1 to 0
```

The GM algorithm selects some features to mutate according to a mutation probability. These features are saved in a set (M), which is then divided into two new sets, those that will be replaced with other non-selected features (MS), and those that will be removed without any replacements (MR). The set (NS), is simply the non-selected features that will substitute those in MS. The last part of the algorithm makes it so that an individual has a chance of growing the number of selected features. At most one feature can be added each iteration, as long as it has less selected individuals than the maximum size, which is defined by the user (30 & 600 in this study). The pseudo-code for GM is presented in *Algorithm 10*:

Algorithm 10: Gonzalez Mutation

Input: One offspring, O

```
for i = 1 to |O|                       % Select which features to mutate
    if rand() ≤ mutationProbability
        M = M ∪ O_i

for i = 1 to |M|                       % Select which of those features to substitute
    if rand() ≤ 0.5
        MS = MS ∪ M_i

MR = M \ MS                            % The rest of the mutated features will be removed

notSelected = [1, n] \ O               % Get all features that aren't selected
notSelected = shuffle(notSelected)     % Shuffle the order
for i = 1 to |MS|
    NS = NS ∪ notSelected_i            % Save indices to substitute those in MS

O' = (O \ M) ∪ NS                      % Remove all features in M from O, and add the substitutes

if rand() ≤ 0.5 and |O'| < maxSize     % Chance of adding 1 additional feature
    notSelected = [1, n] \ O'
    index = rand(1, |notSelected|)
    O' = O' ∪ notSelected_index
```

7.3. Objective Functions

The objective functions evaluated in this study are PCFS (Pearson), FR, KNN, and training-validation percentage, and all objectives are described in this section.

- **FR:** Feature Reduction (FR) is an objective, proposed in both [7], and [9], which reduces the amount of features selected. It is calculated by the following equation:

$$FR = 1 - \frac{Features\ in\ subset}{Total\ number\ of\ features} \quad (48)$$

  $Features\ in\ subset$ are the features that have been selected by NSGA-II during the feature selection process, and $Total\ number\ of\ features$ is equal to the maximum number of features.

- **KNN:** K-nearest neighbor (KNN) is a supervised learning algorithm, making this a wrapper objective. The algorithm is fully explained in section 2.1.1. The reason this objective was chosen is because of the good results it produced together with Cohen's Kappa in [6].

- **Training-validation-percentage:** Training-validation percentage is the method used in [6], where they used Cohen's Kappa, together with LDA, NBC, and KNN. A full explanation of the equations used to implement this objective is done in section 3.4. Together with KNN it produced a good result across all subjects in [6], it is included to do a comparison between it, and the top performing objectives from the other studies.

- **Pearson:** Pearson correlation of selected features (PCFS) [36], measures the linear correlation between two variables *x*, and *y*, in the range of [-1, 1]. When both variables increase in value simultaneously, there is a correlation, and if one increases, and one decreases, no correlation is present. A perfect linear correlation results in a value of 1, while a 0 represents no correlation at all. It is calculated by the following formula:

$$\frac{k \cdot \overline{\tau_{c,f}}}{\sqrt{k + k \cdot (k-1) \cdot \overline{\tau_{a,f}}}} \quad (49)$$

  Where k is the subset of selected features, $\overline{\tau_{s,f}}$ is the average value of the pearson's correlation coefficient, regarding the subset of features, and the classes. Lastly, $\overline{\tau_{a,f}}$ is the average value of the Pearson's correlation coefficient, regarding all selected features.

Since I use a multi-objective algorithm, I can optimize two objectives at once, therefore I've selected the most promising objectives from the related studies, to compare with the different combinations of NSGA-II. The combinations are as follows:

- Pearson – FR
- Pearson – KNN
- Training-validation-percentage, with KNN as classifier

The combination of Pearson – FR is included because it performed well in both [7], and [9]. The reason for using KNN as a wrapper objective, is because it always performed good in [6], with the training-validation-percentage. A prior test was done that where KNN was combined with the objectives from [9], and it was found that it performed best together with Pearson, which is why I also included that combination.

7.4. Classifiers

In [7], and [9], they used SVM with a linear kernel to classify. In this study I expanded the classifier options, to find out how different combinations of NSGA-II, and different classifiers work together. For SVM, the linear kernel was included, to be able to compare results with the two reference studies, [7], and [9]. I also included the Polynomial kernel, described in EQ. (12), in section 2.1.2. The reason being that it is a good, well known kernel.

Both [7], and [9], used SVM due to it not requiring many samples to be efficient. In [6], they proposed LDA, and KNN as classifiers, measuring good results. Therefore, I chose to include them in this study, to compare them against the SVM solutions. Lastly, I included ANN, since it's also a very well known, good performing classifier.

In summary, the evaluated classifiers were:

- SVML
- SVMP
- LDA
- KNN
- ANN

All of which, are implemented as described in section 2.2.

8. Experiment Settings

In this section, the settings used in the study, regarding NSGA-II, and the classifiers, are presented. I used the same parameters as [7], and [9], to be able to compare the result to their studies. The parameters are slightly different between the representations. In section 8.1, the settings between the different representations, regarding NSGA-II, are presented, and in section 8.2, the settings regarding the classifiers are presented.

8.1. NSGA-II Settings

In this section, all settings regarding NSGA-II, and the different representations, are presented.

8.1.1. Hierarchical settings:

- Population size: 15
- Maximum generations: 3000
- Crossover probability: 0.5
- Mutation probability 1: 0.05 (Layer 1)
- Mutation probability 2: 0.01 (Layer 2)
- Mutation probability 3: 0.1 (Layer 3)
- Window size: 40

8.1.2. Non-Hierarchical settings:

- Population size: 15
- Maximum generations: 3000
- Crossover probability: 0.5
- Mutation probability: 0.05
- Window size: 40

8.1.3. González settings:

Two versions were tested, where the only difference were the maximum selected features. This was because I wanted to test both the one they used in [6], which was a maximum size of 30, and compare it to a larger size of 600, to see how that would affect the end result.

- Population: 15
- Maximum generations: 3000
- Mutation probability: 0.01
- Maximum selected features: 30 & 600
- Window size: 40

8.2. Classifier Settings

In this section, the settings for all five classifiers are presented. All classifiers have been implemented using Matlab.

8.2.1. Support Vector Machine settings:

Two SVM classifiers were implemented, with two different kernels. Both were implemented using Matlab's *fitcsvm* function. Below are the settings used for the kernels.

- Linear Kernel (SVML)**:** Default values. No other settings required.
- Polynomial Kernel (SVMP)**:** Default values, where the order variable $d = 3$.

8.2.2. Linear Discriminant Analysis settings:

LDA was implemented using Matlab's *fitcdiscr* function, with default parameters, except for an additional automatic optimization of both delta, and gamma, using Bayesian optimization.

8.2.3. Artificial Neural Network:

ANN was implemented using the Matlab toolbox Adam (adaptive moment estimation) optimizer. The configuration was set as follows:

- Learning rate: 0.01
- Minimum epochs: 300
- Input layer number of neurons: Number of selected features
- Hidden layers: 1
- Hidden layers number of neurons: $0.2 * Number\ of\ input\ neurons$
- Hidden layers activation function: ReLU
- Output layer number of neurons: 2 (two classes to classify)
- Output layer activation function: Softmax

8.2.4. K-Nearest Neighbour Settings:

KNN was implemented as explained in section 2.1.1, with the only parameter *k* set equal to 5.

9. Results

In this section, the results found in the study are presented. The remainder of this section is organized as follows: In section 9.1, a comparison between the best performing combinations are presented. In section 9.2, the correlation between the objectives, and the classifiers, for the three best combinations of each representation, are presented. In section 9.3, a statistical validation between the objectives, and classifiers are presented, and finally, in section 9.4, a comparison with the results found by this study, and the reference studies, [7], and [9], are studied.

To better format the tables, the following abbreviations are used:

- **HR:** Hierarchical Representation
- **NHR:** Non-Hierarchical Representation
- **G30:** González Representation, with a maximum limit of 30 selected features
- **G600:** González Representation, with a maximum limit of 600 selected features
- **GC:** González Crossover
- **UC:** Uniform Crossover
- **SPC:** Single-Point Crossover
- **MPC:** Multi-Point Crossover
- **GM:** González Mutation
- **BM:** Bit-Inversion Mutation
- **RWS:** Roulette Wheel Selection
- **TOS:** Tournament Selection
- **SUS:** Stochastic Universal Selection
- **KNN_tra - KNN_val:** Training-validation-percentage with KNN

Each NSGA-II combination is listed in the following format:

- Representation-Selection-Crossover-Mutation-[Objective 1-Objective 2]

9.1. Comparison between representations & objectives

In this section, the different combinations, and objective pairs are presented, regarding accuracy and feature reduction. The following data includes the mean accuracy, and mean feature reduction percentage, of all subjects. In Table 1, the three best combinations from each representation are presented. In Table 2, the three best combinations from each objective are presented. Finally, in Table 3, the single best combination from each representation are presented.

In Table 1, the Hierarchical representation, together with stochastic selection, uniform crossover, bit-inversion mutation, and PCFS-KNN objective, resulted in the highest average accuracy, across all subjects, and classifiers. It was able to classify MI-activity with 73.5% accuracy on average, with an average feature reduction of 97%. Best average feature reduction came from all three González_30 representations, most likely due to them having a maximum limit of 30 features selected per individual. However, the average accuracy isn't far from the best hierarchical representation, only about 1-2% off. Similarly, the best HR representation with highest average accuracy, only has 1% lower feature reduction than the González_30 combinations. Taking all three combinations into account, for each representation, we can see that the HR representation has a consistent average accuracy, with all three predicting around 73% of the input data correctly, while still maintaining a high feature reduction. The Non-Hierarchical representation seems to perform the worst, with the three best combinations only achieving an average of 50% feature reduction, considerably less than the others. Even though they use more features, they still achieve less accuracy than all the other representations. The González representation with a limit of 600 features, performs worse than the other with a limit of 30 features, on both accuracy, and feature reduction. Lastly, 10/12 of the best performing combinations, came from using PCFS-KNN as objective pair.

Table 1. Three best performing NSGA-II combinations, from each representation.

Combination	Classifier					Accuracy	Feature reduction
	LDA	SVML	SVMP	ANN	KNN	Average	Average
G30-RWS-GC-GM-[PCFS-KNN]	0.747	0.738	0.71	0.74	0.675	0.722	**0.977**
G30-SUS-GC-GM-[PCFS-KNN]	0.745	0.737	0.7	0.734	0.676	0.718	**0.977**
G30-TOS-GC-GM-[PCFS-KNN]	0.751	0.737	0.702	0.738	0.662	0.718	**0.977**
G600-TOS-GC-GM-[PCFS-FR]	0.763	0.714	0.718	0.743	0.583	0.704	0.935
G600-RWS-GC-GM-[PCFS-FR]	0.762	0.722	0.703	0.746	0.578	0.702	0.893
G600-TOS-GC-GM-[PCFS-KNN]	0.778	0.749	0.581	**0.769**	0.628	0.701	0.667
HR-SUS-UC-BM-[PCFS-KNN]	0.755	0.752	**0.722**	0.754	0.693	**0.735**	0.969
HR-RWS-MPC-BM-[PCFS-KNN]	0.755	0.744	0.719	0.75	0.695	0.733	0.968
HR-TOS-UC-BM-[PCFS-KNN]	0.757	0.744	0.706	0.745	**0.698**	0.73	0.971
NHR-TOS-SPC-BM-[PCFS-KNN]	0.776	0.752	0.567	0.768	0.621	0.697	0.556
NHR-TOS-UC-BM-[PCFS-KNN]	**0.779**	0.747	0.569	0.77	0.618	0.697	0.573
NHR-SUS-SPC-BM-[PCFS-KNN]	0.778	**0.753**	0.561	0.767	0.614	0.695	0.555

In Table 2, we can see that PCFS-KNN achieves the highest average accuracies, with a high, consistent feature reduction. However, the highest average feature reduction comes from using PCFS-FR, with the Gonzáles_30 representation. It seems like PCFS-FR works best as objective function for both González representations, since they occupy all the spots for that objective pair. Both PCFS-KNN, and KNN_tra-KNN_val only consists of the hierarchical representation, while no non-hierarchical representation seems to dominate any objective pair. We can also see that the KNN_tra-KNN_val objective pair perform well on both feature reduction, and accuracy. However, it still performs worse than PCFS-KNN for all three combinations, regarding both accuracy, and feature reduction.

Table 2. Three best performing NSGA-II combinations, of each objective pair.

Combination	Classifier					Accuracy	Feature reduction
	LDA	SVML	SVMP	ANN	KNN	Average	Average
G600-TOS-GC-GM-[PCFS-FR]	**0.763**	0.714	0.718	0.743	0.583	0.704	0.935
G30-SUS-GC-GM-[PCFS-FR]	0.746	**0.747**	0.69	0.742	0.595	0.704	**0.988**
G600-RWS-GC-GM-[PCFS-FR]	0.762	0.722	0.703	0.746	0.578	0.702	0.893
HR-SUS-UC-BM-[PCFS-KNN]	0.755	0.752	**0.722**	**0.754**	0.693	**0.735**	0.969
HR-RWS-MPC-BM-[PCFS-KNN]	0.755	0.744	0.719	0.75	0.695	0.733	0.968
HR-TOS-UC-BM-[PCFS-KNN]	0.757	0.744	0.706	0.745	**0.698**	0.73	0.971
HR-TOS-UC-BM-[KNN_tra-KNN_val]	0.76	0.725	0.693	0.747	0.577	0.7	0.908
HR-SUS-SPC-BM-[KNN_tra-KNN_val]	0.76	0.73	0.681	0.746	0.582	0.7	0.902
HR-RWS-UC-BM-[KNN_tra-KNN_val]	0.76	0.724	0.686	0.748	0.572	0.698	0.903

Table 3 summarizes the other two tables, by displaying the top performing combinations from each representation. We can once again, see that the hierarchical representation achieves the best, average accuracy, while maintaining the second highest feature reduction (1% lower). At the same time, González_30 achieves similar accuracy (1.3% lower), with slightly more feature reduction, with PCFS-KNN as objective pair. Once again, it is clear that the non-hierarchical representation suffers in regard to feature reduction. However, it still performs better than the other combinations, with LDA, SVML, and ANN as classifiers, possibly due to it using more features. Although, it still achieves less accuracy in average, across all the classifiers, than every other representation.

Table 3. The single, best performing NSGA-II combinations, from each representation.

Combination	Classifier					Accuracy	Feature reduction
	LDA	SVML	SVMP	ANN	KNN	Average	Average
G30-RWS-GC-GM-[PCFS-KNN]	0.747	0.738	0.71	0.74	0.675	0.722	**0.977**
G600-TOS-GC-GM-[PCFS-FR]	0.763	0.714	0.718	0.743	0.583	0.704	0.935
HR-SUS-UC-BM-[PCFS-KNN]	0.755	**0.752**	**0.722**	0.754	**0.693**	**0.735**	0.969
NHR-TOS-SPC-BM-[PCFS-KNN]	**0.776**	**0.752**	0.567	**0.768**	0.621	0.697	0.556

9.2. Correlation between objectives & classifiers

In this section, the correlation between the objectives, and the classifier algorithms, are studied. Only the most significant combinations are studied, which are the three best combinations from each representation, found in Table 1. The correlation model used is Pearson's correlation coefficient. The data tables referred to in this section, can be found in Appendix A, section A.2.

The three combinations using the HR representation, HR-TOS-UC-BM, HR-RWS-MPC-BM, and HR-SUS-UC-BM (Table A5-Table A7), all have similar p-values, even though they are using different combinations of breeding operators. All three combinations have little, to no correlation between objective FR, and the two classifiers SVMP, and KNN. However, there are weak, negative correlations between the objective, and the other classifiers, LDA, SVML, and ANN, for all three combinations. None of the combinations have any correlation between the two objectives in the pair KNN_tra-KNN_val. However, they do have positive correlations between the objectives, and the classifiers. The objective PCFS-KNN can be observed to have moderate, positive correlation between the two objectives, and all the classifiers, possibly correlating with the higher average accuracy, and feature reduction across all classifiers, as seen in Table 1-3.

Regarding the three NHR combinations, NHR-TOS-UC-BM, NHR-TOS-SPC-BM, and NHR-SUS-SPC-BM (Table A9-Table A11), the *p*-values are similar to HR. FR has slightly weaker correlations between the two objectives, FR, and PCFS. The correlations observed, are also weaker between the objective pair, and the classifiers, with FR having little, to no correlation between it, and the classifiers. However, PCFS has identical correlations to HR, between the objective and the classifiers. Another similarity with the HR combinations, is that PCFS-KNN have moderate correlations between the two objectives, and the classifiers. The last objective pair KNN_tra-KNN_val, also have similar correlations as HR, only slightly higher between the two individual objectives. But the difference is too small to ensure a correlation.

The three combinations using G30, G30-RWS-GC-GM, G30-SUS-GC-GM, and G30-TOS-GC-GM (Table A12-Table A14), has similar correlations to NHR regarding PCFS-FR, where the correlations are weak, between the two objectives, except for G30-TOS-GC-GM, where the correlation is slightly higher, and more similar to HR. The correlations between PCFS-FR, and the classifiers are also identical to NHR, where the correlations are either very weak, or none existent in the case of G30-RWS-GC-GM. Regarding PCFS-KNN, the correlations look identical to both HR, and NHR, with moderate correlations between PCFS, and KNN, moderate correlations between KNN and the classifiers, and strong correlation between PCFS and the classifiers. Regarding KNN_tra-KNN_val, G30 has slightly higher correlation between the two objectives compared to both HR, and NHR, while also having the correlation being negative (opposed to the positive correlation the other representations had). For the G30-RWS-GC-GM, and G30-SUS-GC-GM combinations, a weak correlation between KNN_tra, and

KNN_val can be concluded. KNN_tra has weaker correlations to the classifiers, while KNN_val has similar correlations to HR, and NHR.

Regarding the three G600 combinations, G600-TOS-GC-GM, G600-RWS-GC-GM, and G600-TOS-GC-GM (Table A15-Table A17), similarly to most NHR, and G30 combinations, it can be observed that they have little, to no correlation between PCFS and FR, while having moderate to strong correlations between PCFS, and the classifiers. FR has similarly very weak, to no correlation between the classifiers. As all other observed combinations, KNN has positive correlations to PCFS, although slightly weaker than the others. Simultaneously, it has moderate correlation to all classifiers, except to the classifier KNN, which is strong.

The most consistent objective pair is PCFS-KNN, which for all representations, have positive correlations between the two objectives in the pair, and between the objectives, and all the classifiers. This is the case for all tested combinations, including those not mentioned in this section. FR has no, to weak, correlations (negative) between it, and the classifiers, while also having moderate correlation to PCFS for HR combinations, and between weak, and no correlation at all, to PCFS for the other representations. KNN_tra-KNN_val always has positive correlations in regards to the classifiers, with moderate strength. In the same regard, PCFS and KNN both have positive correlations to all classifiers, while PCFS's being strong, and KNN's being moderate.

Analyzing the results, it seems like breeding operators doesn't have a big impact on the correlations. The differences are too small between combinations using the same representation, while one operator doesn't produce the same small differences every time it appears in a combination. This makes it impossible to conclude whether one operator has any impact on the correlations. However, big differences can be seen between the objectives, with PCFS having a constant moderate, or strong correlations between it, and the classifiers. FR has weak, to no correlations to all classifiers, and KNN have moderate correlations to PCFS, and the classifiers. Lastly, the correlation between the objectives KNN_tra, and KNN_val, are weak, to none existent, while having moderate correlations to the classifiers.

9.3. Statistical validation

In this section, a statistical validation between the objectives, and between the classifiers are presented. Wilcoxon's signed rank test was used to evaluate whether two groups has the same continuous distribution. The statistical validation between the three pairs of objectives, finds whether one objective pair is statistically better than another pair of objectives, for that specific combination. The same is true for the statistical validation between the classifiers, whether one classifier is statistically better than another. The statistics analyzed in this section corresponds to the most significant combinations, which are the three best performing combinations from each representation, presented in Table 1. The data tables analyzed in this section, can be found in Appendix A, section A.3.

The data in the analyzed tables, are interpreted as follows:

- $p \geq 0.05$: No statistical significance.
- $p < 0.05$: Weak statistical significance.
- $p < 0.01$: Moderate statistical significance.
- $p < 0.001$: Strong statistical significance.

Meaning that the null-hypotheses of one algorithm (*alg1*) being better, or equal to another (*alg2*), can be rejected, at a statistical significance of 0.1% ($p < 0.001$), 1% ($p < 0.01$), and 5% ($p < 0.05$). A smaller *p*-value results in one pair of objectives, or one classifier, being statistically better than another pair of objectives, or classifier. If the *p*-value ≥ 0.05, the null-hypotheses is true, and *alg1* is worse, or equal to *alg2*.

The HR representations seem to always have PCFS-KNN dominate the other objective pairs ($p < 0.001$), with no other pair having any statistical significance. Regarding classifiers, LDA is clearly statistically better then all others, for every combination. SVML and ANN is statistically similar, and is better than both SVMP, and KNN. It could indicate that LDA is the most viable choice for this combination, with ANN, and SVML being second. KNN is dominated by all other classifiers, for all combinations, indicating that it could be the worst choice regarding the HR representation.

Regarding NHR, it is no longer a clear choice between the objective pairs. PCFS-KNN is statistically better than KNN_tra-KNN_val, for NHR-TOS-UC-BM, but only with a p-value < 0.05, while no statistical difference can be observed between PCFS-KNN, and PCFS-FR. For NHR-TOS-SPC-BM, PCFS-KNN is slightly better than PCFS-FR ($p < 0.05$). For the last combination, NHR-SUS-SPC-BM, no objective pair is statistically different than another. Regarding the classifiers, the results are similar to those found with HR. The only difference is that ANN was observed to be better than SVML, which it wasn't for any HR combination.

Taking Table 2 into account, it seems like G30, and G600, works well with PCFS-FR as objective. Together they achieve the three best results using that objective pair. G30's statistical validations, are observed to be the only representation to have a strong statistical significance regarding PCFS-FR, possibly indicating that PCFS-FR could work very well paired with that representation. However, PCFS-KNN has a strong significance over both PCFS-FR, and KNN_tra-KNN_val, when using TOS, or RWS as selection operator. Using SUS as selection, seems to result in no statistical difference between PCFS-FR, and PCFS-KNN. Regarding classifiers, this representation has identical results to NHR, and HR. The G600 representation, is observed to have no statistical differences between the objectives, while also having identical classifier results as the other three representations.

The statistical significance regarding objectives, seems to not differ much between combinations that are using the same representation, possibly indicating that the representation has a bigger impact regarding which objectives to use. All representations have similar results regarding classifiers, where LDA always dominate all other classifiers, and KNN being the worst one in every test.

9.4. Comparison to reference studies

In this section, a comparison between the results in this study, and the two reference studies are made. To be able to do the comparison, the same parameters as the reference studies was used, with a window size of 40, and SVML as classifier. The best performing combination, regarding average accuracy, and feature reduction was chosen from the reference papers, while the best of each representation is used from this study, presented in Table 3. The results are the average result, after a 10-fold evaluation, also averaged from all subjects. In Table 4, the four best combinations from this study, is compared to the combinations used in the reference papers [7], [9].

Since this study use a window size of 40, the results with that windows size are only considered, and compared. In [7], they used HGA, explained in section 3, with SVML as classifier, and PCFS-FR as objectives. In [9], they used NSGA-II, with TOS, UC, and BM as breeding operators, and HR as representation (giving the combination HR-TOS-UC-BM), while also using SVML as classifier. As for objective functions, they found that for a window size of 40, PCFS-FR gave the best feature reduction, and accuracy.

In Table 4, it can be observed that some of the combinations from this study outperformed both reference algorithms. HR-SUS-UC-BM have the best result of all, with the highest average accuracy (same as NHR-TOS-SPC-BM) of 0.752, and the highest feature reduction of 0.995. It can also be observed that both G30, and G600 have higher feature reduction than the reference algorithms. However, only G30 have a higher accuracy, but not by much. It is likely that the good classification accuracy, for the combinations used in this study, comes from using PCFS-KNN as objective pair.

Table 4. Comparison with reference studies, for a window size of 40, using SVML as classifier. The best combination is taken from the reference papers, while the best of each representation is used from this study. The results are the average result, after a 10-fold evaluation, where the result from all subjects are averaged.

Combination	Classifier	Feature reduction
	SVML	Average
HGA-[PCFS-FR] [7]	0.69	0.783
HR-TOS-UC-[PCFS-FR] [9]	0.729	0.909
G30-RWS-GC-GM-[PCFS-KNN]	0.738	0.977
G600-TOS-GC-GM-[PCFS-FR]	0.714	0.935
HR-SUS-UC-BM-[PCFS-KNN]	**0.752**	**0.995**
NHR-TOS-SPC-BM-[PCFS-KNN]	**0.752**	0.667

10. Discussion

In this section, the results from the previous section are discussed, and the most significant results are highlighted. The section is divided into five subsections, and is organized as follows: In section 10.1, the results found regarding the tested individual representations are discussed. In section 10.2, the breeding operators, and their results, are discussed. In section 10.3, the three pairs of objective functions used, and how the results relate to the representations, and classifiers, are discussed. Section 10.4 goes more into detail regarding the classifiers, and how their results relate to the other components. Finally, in section 10.5, the contributions made by this paper, and the most significant results, are presented.

10.1. Representations

From Table 1, we can see that most representations performed relatively well, except for NHR since it only obtained between 55%-57% feature reduction. It is likely that the high accuracy for LDA, SVML, and ANN comes from using considerably more features than the others. This amount of feature reduction can be observed for all of the NHR combinations in Appendix A. The two most effective representations are HR, and G30, where the resulting feature reductions, and accuracies are similar. G30 obtains slightly more feature reduction, likely due to having the hard limit of 30 selected features per individual. However, it doesn't suffer much in accuracy, when comparing it to HR, which obtained the highest average accuracy. The difference is only about 1-2%, for both accuracies regarding individual classifiers, and the average accuracy. When comparing HR's feature reduction to G30, we can see that it only has about 1% less, where HR doesn't have any max limit on features selected. Therefore, I believe the two representations to be very similar in strength, regarding this type of problem. G600 is not too far behind, but it seems like the González representation gets worse when we increase the maximum features used, since it clearly performs worse on both average accuracy, and feature reduction, as low as 67% for the combination G600-TS-GC-GM-[PCFS-KNN]. Both HR, and G30 seems very consistent, G30 has 7% between its worst accuracy, and its best, while the difference is only 2% regarding feature reduction. HR has a difference of 7% between its worst, and its best feature reduction, and a 5% difference for average accuracy. With these results, I conclude that HR, and G30 are two good choices for NSGA-II, regarding a feature reduction problem.

10.2. Breeding Operators

From Tables 1-3, it can't be concluded whether one breeding operator is better than another. In Table 1, TOS appears 6 times, while SUS, and RWS appears 3 times. In Table 2, each selection method appears three times. The results don't differ much between the different selection operators either. While SUS results in the best observed accuracy (for combination HR-SUS-UC-BM-[PCFS-KNN]), it is not clear whether it is by chance, since it doesn't produce the best accuracy for any other representation. When looking at the results from all combinations (Appendix A, section A.1), all results look similar when focusing on the different breeding operators. The only noticeable difference between results, is when the objective functions, and representations change. Looking at the correlation, and validation tables from sections A.2, and A.3 respectively, it can be observed that they don't change much, if at all, when comparing different breeding operators, within each representation. In conclusion, it seems like the breeding operators have too small of an impact to be noticeable by the accuracy, and feature reduction.

10.3. Objectives

Regarding objective functions, it's interesting to see in Table 1, that all but two combinations, have PCFS-KNN as objectives for their best results. It appears that this pair obtains the best results regarding accuracy, and it doesn't suffer much loss regarding feature reduction. In Table 2, we can see that PCFS-KNN once again achieves the highest average accuracies, with a high, consistent feature reduction. This coincide with the statistical validation in section 9.3, where for most combinations, PCFS-KNN was found to be statistically better than the other two objectives. In Table 2, it is observed that all top results, with KNN being present in the objective pair, is occupied by HR combinations, also coinciding with section 9.3, where all HR combinations had PCFS-KNN as statistically better than the other two. Taking the correlations into account, all tested combinations have a correlation between the two objectives in the pair PCFS-KNN, while simultaneously having correlations to all classifiers. This is the case for all tested combinations. The results show that PCFS-KNN is a very good objective pair, combined with NSGA-II, for feature reduction problem. The objective KNN_tra-KNN_val seem to always result in slightly worse average accuracy, and feature reduction than the other two (Tables 1-3), while always being statistically worse than the other pairs.

In Appendix A, section A.1, all combinations are listed, by looking at those tables, we can see that the combinations with G30 as representation, and PCFS-FR as objectives, never go below 98.7% feature reduction, while HR combined with PCFS-FR always has a reduction of 99%. However, they suffer a bit regarding accuracy, with both representation unable to predict accurately above 70%. But looking at Table A1, we can see that the difference between PCFS-FR, and PCFS-KNN regarding feature reduction is about 3%, which is also the difference in accuracy, where PCFS-KNN has the highest average. From Table 2, the best combinations using PCFS-FR, were from the González representations (G30, and G600). Observing their tables in Appendix A, section A3, we can see that for G30, PCFS-FR has also become statistically better than the third objective, KNN_tra-KNN_val, possibly indicating that PCFS-FR is more viable for that representation. However, for G600, no objective is statistically better than another. FR seems to have no correlations to the classifiers. However, it doesn't make the objective bad, since it still produces very good results together with PCFS. Further, PCFS produce good results, regardless whether the correlations between it, and its paired objective (FR, and KNN) is positive, or negative. However, negative correlations between the objectives (PCFS-FR) seem to relate to higher feature reduction, while positive correlations (PCFS-KNN) could relate to higher accuracy.

In conclusion, PCFS-FR is the best objective regarding feature reduction, which coincide with what was reported in [9]. PCFS-KNN instead produced slightly better accuracy, while still performing well regarding feature selection. It can be concluded that both PCFS-KNN, and PCFS-FR are two viable options regarding feature selection, where PCFS-FR has higher feature reduction (3%), and PCFS-KNN has higher accuracy (3%).

10.4. Classifiers

In Table 1, we can see that the best performing classifiers are LDA, SVML, and ANN. LDA have slightly better results overall, somewhere between 1-5%, compared to SVML, and ANN. This also coincide with the statistical validations, where LDA always dominated the other classifiers, while both SVML, and ANN dominated SVMP, and KNN. In addition to that, ANN can be observed to be statistically better than SVML for some representations, namely NHR, G30, and G600, while achieving 1-2% better accuracy than SVML. For all combinations, all classifiers have strong, positive correlations with PCFS, and moderate, positive correlation with the three KNN objectives. The only objective having little, to no correlation to the classifiers, are FR, which is true for all combinations.

Combining those results, it can be concluded that KNN, and SVMP doesn't perform as well as the other classifiers. Instead, I conclude that LDA, ANN and SVML are all fitting classifiers for feature selection, in regards to accuracy, and feature reduction, with LDA producing the highest accuracies. However, ANN seems more viable than SVML for NHR, and G30 representations, due to it being statistically better for those representations.

10.5. Contributions

The purpose of this study, was to evaluate NSGA-II, together with different combinations of individual representations, breeding operators, objective functions, and classifiers. The focus was placed on prediction accuracy, and feature reduction, and how the different combinations perform in those regards. The results are presented in section 9, and Appendix A, which answers both research questions asked in section 4. With the results found, one could decide what combination they deem to fit their needs, regarding feature reduction. G30, and HR showed very promising results, regarding representations, while both PCFS-KNN, and PCFS-FR showed promising results regarding objective functions. However, different breeding operators seemed to not have a big impact on the performance, which makes objective functions, and classifiers more important. Finally, LDA, ANN, and SVML produced the best results of all tested classifiers, with LDA being the best.

11. Conclusions

This study evaluated multiple combinations of individual representations, breeding operators, objective functions, and classifiers, for feature reduction in EEG classification, regarding MI activity. The algorithm used for feature reduction was NSGA-II, for which the operators were implemented, and tested. The purpose was to evaluate how these different combinations would affect the feature reduction, and classification. Initially, all combinations were compared to each other, and between the results in the reference papers. Then, using Pearson's correlation coefficient, correlations between objectives, and classifiers, and were calculated, for each NSGA-II combination. Later, using Wilcoxon signed-rank test, a statistical validation was done, also between objectives, and classifiers, for each combination. Three individual representations, three selection operators, four crossover methods, two mutation operators, and three pairs of objective functions were evaluated. The classification was then done with five different classifiers, to also analyse how they would affect the different combinations. The data was recorded from 6 subjects, using 42 channels, with 30 frequencies each.

It was found that the Hierarchical (HR), and one of the two González (G30 & G600) representations resulted in the highest accuracies, with the lowest amount of features used. One of the most promising objective functions were a combination of correlation feature selection, with Pearson's correlation coefficient (PCFS), combined with k-Nearest Neighbour (KNN), with $k = 5$, resulting in the objective PCFS-KNN. When pairing that with either HR, or G30 the best results were produced, regarding accuracy, and feature reduction. When looking at the correlations between the objectives, and the classifiers, it was found that PCFS-KNN had positive correlations to all classifiers, through all combinations. Similar results were found by the statistical validation, where PCFS-KNN was observed to be statistically better than the other two objectives, for HR, Non-hierarchical (NHR), and G30. However, combining PCFS with Feature Reduction (FR), resulting in the objective PCFS-FR, also achieved high accuracy, and feature reduction (about 3% lower accuracy than PCFS-KNN, but 3% higher feature reduction). Both objectives are concluded to be viable options, but the statistical validation favours PCFS-KNN, since was statistically better than PCFS-FR for all combinations.

Regarding breeding operators, it was found that they didn't have any big significance on the results, instead, different individual representations, and objective functions produced the only noticeable changes, regarding the results. This significance was also confirmed by the correlations, and by the statistical validations between the objectives, and classifiers. The correlations, and validations barely changed between different breeding operators, while changing noticeably between different individual representations.

LDA, SVML, and ANN were found to be the most significant classifiers, producing the highest accuracies, when combined with either HR-[PCFS-KNN], or G30-[PCFS-KNN]. This significance was also confirmed when comparing the statistical validations between the classifiers. LDA dominated all other classifiers according to the statistical validation, for every combination tested. But ANN, and SVML are also good options, and were found to be statistically better than SVMP, and KNN. For NHR, G30, and G600 combinations, ANN was statistically better than SVML too, which could indicate it being the second-best classifier with those representations. Overall, SVML, and ANN produced similar accuracy as LDA, only a few percentages lower.

12. Future Work

In this study, multiple individual representations, and breeding operators, were studied for NSGA-II. There are many more breeding operators that can be studied, and evaluated for NSGA-II. More breeding operators can be found in the following two papers: In [37], they bring up many different selection methods, that could be interesting to evaluate for this kind of problem, and in [38], they present many different crossover operators, for both binary, and integer, representations. It could be very interesting to modify one of the integer crossover operators, to work together with González individual representation from [6]. Also, this type of problem is not limited to NSGA-II, there are many more MOEAs that could be tested, and also evaluated with multiple representations, and breeding operators. To further evaluate this study, the algorithms could be studied regarding execution time, whether some combinations of representation, breeding operators, objective functions, and classifiers, is faster when it comes to reducing the features. Even required memory could be studied, and compared. In [6], they claim to have a faster, and more memory efficient feature selection algorithm, it would be interesting to compare the memory requirement, and speed of their solution, to the combinations having Hierarchical, and Non-Hierarchical representations. The results found in this study, could also be compared to other studies, where other combinations, or objective functions are evaluated.

13. References

[1] J. R. Wolpaw, N. Birbaumer, D. J. McFarland, G. Pfurtscheller and T. M. Vaughana, "Brain–computer interfaces for communication and control," *Clinical Neurophysiology*, vol. 113, no. 6, pp. 767-791, Jun 2002.

[2] R. Sitaram, T. Ros, L. Stoeckel, S. Haller, F. Scharnowski, J. Lewis-Peacock, N. Weiskopf, M. L. Blefari, M. Rana, E. Oblak, N. Birbaumer and J. Sulzer, "Closed-loop brain training: the science of neurofeedback," *Nature Reviews Neuroscience*, vol. 18, no. 2, pp. 86-100, Feb 2017.

[3] J. Decety, "The neurophysiological basis of motor imagery," *Behavioural Brain Research*, vol. 77, no. 1-2, pp. 45-52, May 1996.

[4] O. Bertram, F. Nicola and M. Axel, "Timing matters: The impact of immediate and delayed feedback on artificial language learning," *Frontiers in Human Neuroscience*, vol. 5, p. 8, Feb 2011.

[5] M. Teplan, "Fundamentals of EEG measurement," *Measurement science review*, vol. 2, no. 2, pp. 1-11, Jan 2002.

[6] J. González, J. Ortega, M. Damas, P. Martín-Smith and J. Q. Gan, "A new multi-objective wrapper method for feature selection – Accuracy and stability analysis for BCI," *Neurocomputing*, vol. 333, pp. 407-418, Mar 2019.

[7] M. Leon, J. Ballesteros, J. Tidare, N. Xiong and E. Astrand, "Feature Selection of EEG Oscillatory Activity Related to Motor Imagery Using a Hierarchical Genetic Algorithm," in *2019 IEEE Congress on Evolutionary Computation (CEC)*, Wellington, New Zealand, 2019.

[8] J. Holsheimer and B. W. Feenstra, "Volume conduction and eeg measurements within the brain: a quantitative approach to the influence of electrical spread on the linear relationship of activity measured at different locations," *Electroencephalography and Clinical Neurophysiology*, vol. 43, no. 1, pp. 52-58, Jul 1977.

[9] C. Parkkila, "Empirical studies of multi-objective evolutionary algorithm in classifying neural oscillations to motor imagery," Bachelor Thesis, Apt. IDT, Mälardalen Univ., Västerås, 2019.

[10] C.-Y. Kee, S. Ponnambalam and C.-K. Loo, "Multi-objective genetic algorithm as channel selection method for P300 and motor imagery data set," *Neurocomputing*, vol. 161, pp. 120-131, Aug 2015.

[11] C. Cîmpanu, L. Ferariu, T. Dumitriu and F. Ungureanu, "Multi-Objective Optimization of Feature Selection procedure for EEG signals classification," in *E-Health and Bioengineering Conference (EHB)*, Sinaia, Romania, 2017.

[12] S. B. Kotsiantis, "Supervised Machine Learning: A Review of Classification," in *Frontiers in Artificial Intelligence and Applications*, Amsterdam, IOS Press, 2007, pp. 3-24.

[13] G. James, D. Witten, T. Hastie and R. Tibshirani, Support Vector Machines, New York: Springer, 2013, pp. 337-359.

[14] S. Keerthi and E. Gilbert, "Convergence of a Generalized SMO Algorithm for SVM Classifier Design," *Machine Learning*, vol. 46, no. 1-3, p. 351–360, 2002.

[15] I. .. Basheer and M. Hajmeer, "Artificial neural networks: fundamentals, computing, design, and application," *Journal of Microbiological Methods*, vol. 43, no. 1, pp. 3-31, Dec 2000.

[16] I. Guyon and A. Elisseef, "An Introduction to Variable and Feature Selection," *The Journal of Machine Learning Research*, vol. 3, pp. 1157-1182, Mar 2003.

[17] T. Bäck, Evolutionary Algorithms in Theory and Practice: Evolution Strategies Evolutionary Programming Genetic Algorithms, New York: Oxford University Press, Inc, 1996.

[18] T. Bäck and H.-P. Schwefel, "An overview of evolutionary algorithms for parameter optimization," *Evolutionary Computation*, vol. 1, no. 1, pp. 1-23, 1993.

[19] D. Whitley, "A Genetic Algorithm Tutorial," *Statistics and Computing*, vol. 4, no. 2, pp. 65-85, Jun 1994.

[20] R. Storn and K. Price, "Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341-359, 1997.

[21] W. Banzhaf, "Artificial Intelligence: Genetic Programming," in *International Encyclopedia of the Social & Behavioral Sciences*, Orlando, FL, USA, Elsevier Ltd, 2015, pp. 41-45.

[22] D. Beasley, D. R. Bull and R. R. Martin, "An overview of genetic algorithms: Part 1 fundamentals," *University Computing* , vol. 15, no. 2, pp. 56-69, 1993.

[23] K. Jebari, "Parent Selection Operators for Genetic Algorithms," *International Journal of Engineering Research & Technology,* vol. 2, no. 11, pp. 1141-1145, Nov 2013.

[24] O. Hasançebi and F. Erbatur, "Evaluation of crossover techniques in genetic algorithm based optimum structural design," *Computers & Structures,* vol. 78, no. 1-3, pp. 435-448, Nov 2000.

[25] R. Poli, N. F. McPhee and L. Vanneschi, "Elitism Reduces Bloat in Genetic Programming," in *GECCO '08 Proceedings of the 10th annual conference on Genetic and evolutionary computation*, Atlanta, GA, USA, Jul 2008.

[26] A. Konak, D. W. Coit and A. E. Smith, "Multi-objective optimization using genetic algorithms: A tutorial," *Reliability Engineering & System Safety,* vol. 91, no. 9, pp. 992-1007, Sep 2006.

[27] N. Siinivas and K. Deb, "Muilti-Objective Optimization Using Nondominated Sorting in Genetic Algorithms," *Evolutionary Computation,* vol. 2, no. 3, pp. 221-248, 1994.

[28] J. Schaffer, "Some experiments in machine learning using vector evaluated genetic algorithms," Ph.D. dissertation, Dept. Math. Comp., Vanderbilt Univ, Nashville, TN, 1985.

[29] D. E. Goldberg and J. H. Holland, "Genetic Algorithms and Machine Learning," *Machine Learning,* vol. 3, no. 2-3, pp. 95-99, Oct 1988.

[30] K. Deb and D. E. Goldberg, "An Investigation of Niche and Species Formation in Genetic Function Optimization," in *Proceedings of the 3rd International Conference on Genetic Algorithms*, San Mateo, CA, 1991.

[31] K. Deb, S. Agrawal, A. Pratap and T. Meyarivan, "A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II," in *Parallel Problem Solving from Nature PPSN VI*, Berlin, Germany, 2000.

[32] P. Martín-Smith, J. Ortega, J. Asensio-Cubero, J. Q. Gan and A. Ortiz, "A supervised filter method for multi-objective feature selection in EEG classification based on multi-resolution analysis for BCI," *Neurocomputing,* vol. 250, no. 1, pp. 45-56, Aug 2017.

[33] J.-P. Vert, K. Tsuda and B. Schölkopf, "A primer on kernel methods," in *Kernel Methods in Computational Biology*, Cambridge, United States, MIT Press, 2004, pp. 35-70.

[34] C. Jacob, "A coefficient of agreement for nominal scales," *Educational and Psychological Measurement,* vol. 20, no. 1, pp. 37-46, Apr 1960.

[35] A. Schloegl, C. Neuper and G. Pfurtscheller, "Subject specific EEG patterns during motor imaginary [sic.: for imaginary read imagery]," *Proceedings of the 19th Annual International Conference of the IEEE Engineering in Medicine and Biology Society,* vol. 4, pp. 1530-1532, Nov 1997.

[36] M. A. Hall, "Correlation-based Feature Selection for Machine Learning," *Ph.D. dissertation, Dept. Comp. Sci., Univ. of Waikato, Hamilton, NZ, 1999.*

[37] R. Sivaraj and T. Ravichandran, "A review of selection methods in genetic algorithm," *International Journal of Engineering Science and Technology,* vol. 3, no. 5, pp. 3792-3797, May 2011.

[38] A. Umbarkar and P. Sheth, "Crossover Operators in Genetic Algorithms: A Review," *ICTACT Journal on Soft Computing,* vol. 6, no. 1, pp. 1083-1092, Oct 2015.

Appendix A

This appendix includes all data analyzed in section 9, and is structured as follows: In section A.1, the results regarding average accuracy, and feature reduction, for all combinations are presented. Section A.2 covers the correlation tables, for the most significant combinations. Lastly, section A.3 includes the statistical validation tables.

A.1 Comparison between representations & objectives

This subsection of the appendix presents the average accuracies, and feature reductions, for every combination studied. In this section, the average accuracy, and feature reduction, across all subjects, and classifiers, are presented.

The section is organized by representation, where every subsection includes a table of all average results, for every combination with that specific individual representation.

A.1.1 Hierarchical representation

Table A1. Results regarding average accuracy, and feature reduction, for all combinations using the HR representation.

Representation	LDA	SVML	SVMP	ANN	KNN	Accuracy	Feature reduction
HR-TOS-UC-BM-[PCFS-FR]	0.741	0.741	0.7	0.725	0.593	0.7	0.995
HR-TOS-UC-BM-[PCFS-KNN]	0.757	0.744	0.706	0.745	**0.698**	0.73	0.971
HR-TOS-UC-BM-[KNN_tra-KNN_val]	0.76	0.725	0.693	0.747	0.577	0.7	0.908
HR-TOS-SPC-BM-[PCFS-FR]	0.735	0.739	0.697	0.724	0.595	0.698	0.995
HR-TOS-SPC-BM-[PCFS-KNN]	0.749	0.731	0.72	0.74	0.696	0.727	0.966
HR-TOS-SPC-BM-[KNN_tra-KNN_val]	0.756	0.718	0.684	0.731	0.585	0.695	0.904
HR-TOS-MPC-BM-[PCFS-FR]	0.729	0.733	0.695	0.713	0.593	0.692	0.995
HR-TOS-MPC-BM-[PCFS-KNN]	0.752	0.741	0.717	0.736	0.688	0.727	0.968
HR-TOS-MPC-BM-[KNN_tra-KNN_val]	**0.761**	0.714	0.682	0.749	0.57	0.695	0.899
HR-RWS-UC-BM-[PCFS-FR]	0.729	0.733	0.697	0.715	0.597	0.694	0.995
HR-RWS-UC-BM-[PCFS-KNN]	0.744	0.735	0.718	0.739	0.683	0.724	0.969
HR-RWS-UC-BM-[KNN_tra-KNN_val]	0.76	0.724	0.686	0.748	0.572	0.698	0.903
HR-RWS-SPC-BM-[PCFS-FR]	0.736	0.738	0.708	0.716	0.595	0.699	**0.996**
HR-RWS-SPC-BM-[PCFS-KNN]	0.754	0.736	**0.725**	0.737	0.687	0.728	0.965
HR-RWS-SPC-BM-[KNN_tra-KNN_val]	0.755	0.717	0.68	0.742	0.58	0.695	0.892
HR-RWS-MPC-BM-[PCFS-FR]	0.723	0.73	0.705	0.712	0.593	0.692	0.995
HR-RWS-MPC-BM-[PCFS-KNN]	0.755	0.744	0.719	0.75	0.695	0.733	0.968
HR-RWS-MPC-BM-[KNN_tra-KNN_val]	0.755	0.707	0.667	0.735	0.584	0.689	0.89
HR-SUS-UC-BM-[PCFS-FR]	0.731	0.736	0.7	0.717	0.591	0.695	0.995
HR-SUS-UC-BM-[PCFS-KNN]	0.755	**0.752**	0.722	**0.754**	0.693	**0.735**	0.969
HR-SUS-UC-BM-[KNN_tra-KNN_val]	0.757	0.716	0.678	0.737	0.582	0.694	0.909
HR-SUS-SPC-BM-[PCFS-FR]	0.738	0.743	0.702	0.721	0.607	0.702	0.995
HR-SUS-SPC-BM-[PCFS-KNN]	0.752	0.739	0.715	0.748	0.683	0.728	0.966
HR-SUS-SPC-BM-[KNN_tra-KNN_val]	0.76	0.73	0.681	0.746	0.582	0.7	0.902
HR-SUS-MPC-BM-[PCFS-FR]	0.737	0.737	0.705	0.72	0.598	0.7	0.995
HR-SUS-MPC-BM-[PCFS-KNN]	0.748	0.738	0.722	0.74	0.688	0.727	0.966
HR-SUS-MPC-BM-[KNN_tra-KNN_val]	0.743	0.702	0.676	0.728	0.575	0.685	0.907

A.1.2 Non-Hierarchical representation

Table A2. Results regarding average accuracy, and feature reduction, for all combinations using the NHR representation.

Representation	LDA	SVML	SVMP	ANN	KNN	Accuracy	Feature reduction
NHR-TOS-UC-BM-[PCFS-FR]	0.773	0.749	0.585	0.767	0.578	0.69	0.701
NHR-TOS-UC-BM-[PCFS-KNN]	0.779	0.747	0.569	0.77	0.618	**0.697**	0.573
NHR-TOS-UC-BM-[KNN_tra-KNN_val]	0.773	0.753	0.575	0.764	0.564	0.686	0.502
NHR-TOS-SPC-BM-[PCFS-FR]	0.778	0.749	0.565	0.766	0.574	0.686	0.667
NHR-TOS-SPC-BM-[PCFS-KNN]	0.776	0.752	0.567	0.768	0.621	**0.697**	0.556
NHR-TOS-SPC-BM-[KNN_tra-KNN_val]	0.769	0.756	0.573	0.767	0.558	0.685	0.502
NHR-TOS-MPC-BM-[PCFS-FR]	0.774	0.749	0.573	0.768	0.571	0.687	0.672
NHR-TOS-MPC-BM-[PCFS-KNN]	0.775	0.752	0.561	0.768	0.616	0.694	0.558
NHR-TOS-MPC-BM-[KNN_tra-KNN_val]	0.775	0.76	0.576	0.763	0.557	0.686	0.501
NHR-RWS-UC-BM-[PCFS-FR]	0.771	0.748	**0.586**	0.764	0.577	0.689	**0.704**
NHR-RWS-UC-BM-[PCFS-KNN]	0.775	0.741	0.562	0.766	**0.627**	0.694	0.576
NHR-RWS-UC-BM-[KNN_tra-KNN_val]	**0.78**	**0.761**	0.58	**0.773**	0.565	0.692	0.502
NHR-RWS-SPC-BM-[PCFS-FR]	0.778	0.752	0.565	0.769	0.577	0.688	0.671
NHR-RWS-SPC-BM-[PCFS-KNN]	0.772	0.747	0.563	0.764	0.619	0.693	0.558
NHR-RWS-SPC-BM-[KNN_tra-KNN_val]	0.779	0.756	0.571	0.77	0.56	0.687	0.505
NHR-RWS-MPC-BM-[PCFS-FR]	0.779	0.748	0.568	0.769	0.574	0.687	0.677
NHR-RWS-MPC-BM-[PCFS-KNN]	0.774	0.752	0.563	0.767	0.616	0.695	0.559
NHR-RWS-MPC-BM-[KNN_tra-KNN_val]	0.767	0.753	0.573	0.759	0.572	0.685	0.503
NHR-SUS-UC-BM-[PCFS-FR]	0.772	0.749	0.584	0.765	0.575	0.689	0.702
NHR-SUS-UC-BM-[PCFS-KNN]	0.77	0.749	0.564	0.762	0.619	0.693	0.574
NHR-SUS-UC-BM-[KNN_tra-KNN_val]	0.767	0.743	0.574	0.759	0.555	0.68	0.5
NHR-SUS-SPC-BM-[PCFS-FR]	0.779	0.753	0.57	0.767	0.575	0.689	0.672
NHR-SUS-SPC-BM-[PCFS-KNN]	0.778	0.753	0.561	0.767	0.614	0.695	0.555
NHR-SUS-SPC-BM-[KNN_tra-KNN_val]	0.769	0.754	0.573	0.757	0.561	0.683	0.505
NHR-SUS-MPC-BM-[PCFS-FR]	0.778	0.746	0.568	0.763	0.566	0.684	0.675
NHR-SUS-MPC-BM-[PCFS-KNN]	0.776	0.753	0.559	0.763	0.617	0.694	0.559
NHR-SUS-MPC-BM-[KNN_tra-KNN_val]	0.773	0.748	0.58	0.761	0.562	0.685	0.501

A.1.3 González_30 representation

Table A3. Results regarding average accuracy, and feature reduction, for all combinations using the G30 representation.

Representation	LDA	SVML	SVMP	ANN	KNN	Accuracy	Feature reduction
G30-TOS-GC-GM-[PCFS-FR]	0.731	0.739	0.664	0.728	0.586	0.69	**0.99**
G30-TOS-GC-GM-[PCFS-KNN]	**0.751**	0.737	0.702	0.738	0.662	0.718	0.977
G30-TOS-GC-GM-[KNN_tra-KNN_val]	0.704	0.695	0.645	0.681	0.557	0.656	0.976
G30-RWS-GC-GM-[PCFS-FR]	0.732	0.731	0.675	0.725	0.583	0.689	0.987
G30-RWS-GC-GM-[PCFS-KNN]	0.747	0.738	**0.71**	0.74	0.675	**0.722**	0.977
G30-RWS-GC-GM-[KNN_tra-KNN_val]	0.684	0.684	0.642	0.664	0.548	0.645	0.976
G30-SUS-GC-GM-[PCFS-FR]	0.746	**0.747**	0.69	**0.742**	0.595	0.704	0.988
G30-SUS-GC-GM-[PCFS-KNN]	0.745	0.737	0.7	0.734	**0.676**	0.718	0.977
G30-SUS-GC-GM-[KNN_tra-KNN_val]	0.695	0.687	0.643	0.68	0.553	0.651	0.976

A.1.4 Gonzalez_600 representation

Table A4. Results regarding average accuracy, and feature reduction, for all combinations using the G600 representation.

Representation	LDA	SVML	SVMP	ANN	KNN	Accuracy	Feature reduction
G600-TOS-GC-GM-[PCFS-FR]	0.763	0.714	**0.718**	0.743	0.583	**0.704**	**0.935**
G600-TOS-GC-GM-[PCFS-KNN]	0.778	0.749	0.581	**0.769**	**0.628**	0.701	0.667
G600-TOS-GC-GM-[KNN_tra-KNN_val]	**0.784**	**0.762**	0.565	0.763	0.556	0.686	0.54
G600-RWS-GC-GM-[PCFS-FR]	0.762	0.722	0.703	0.746	0.578	0.702	0.893
G600-RWS-GC-GM-[PCFS-KNN]	0.772	0.751	0.579	0.767	0.627	0.699	0.651
G600-RWS-GC-GM-[KNN_tra-KNN_val]	0.773	0.76	0.566	0.763	0.568	0.686	0.539
G600-SUS-GC-GM-[PCFS-FR]	0.759	0.719	0.71	0.741	0.574	0.701	0.911
G600-SUS-GC-GM-[PCFS-KNN]	0.776	0.742	0.575	0.757	**0.628**	0.696	0.657
G600-SUS-GC-GM-[KNN_tra-KNN_val]	0.772	0.754	0.567	0.76	0.567	0.684	0.538

A.2 Correlation between objectives & classifiers

In this section, all correlation tables are presented, for the most significant combinations, which are the top three performing combinations regarding average accuracy, and feature reduction. These combinations were chosen from Table 1, and are analysed in section 9.2. Values in bold represents a p-value of < 0.05, which indicates a correlation between the algorithms. Values not in bold, have a small, to no correlation at all. The section is divided into subsections, where each subsection presents the correlation table for one of the combinations.

A.2.1 HR-TOS-UC-BM

Table A5. Correlation between objectives, and classifiers, for the HR-TOS-UC-BM combination.

Objectives	Objective algorithms			Classifier algorithms				
Obj 1 - Obj 2		PCFS	FR	LDA	SVML	SVMP	ANN	KNN
PCFS - FR	PCFS	-	**-0.39**	**0.67**	**0.68**	**0.58**	**0.62**	**0.4**
	FR	**-0.39**	-	**-0.19**	**-0.2**	-0.06	**-0.24**	-0.06
Obj 1 - Obj 2		PCFS	FR	LDA	SVML	SVMP	ANN	KNN
PCFS - KNN	PCFS	-	**0.32**	**0.63**	**0.6**	**0.66**	**0.6**	**0.51**
	KNN	**0.32**	-	**0.38**	**0.36**	**0.39**	**0.35**	**0.64**
Obj 1 - Obj 2		KNN_tra	KNN_val	LDA	SVML	SVMP	ANN	KNN
KNN_tra - KNN_val	KNN_tra	-	0.04	**0.42**	**0.41**	**0.42**	**0.41**	**0.26**
	KNN_val	0.04	-	**0.5**	**0.5**	**0.47**	**0.53**	**0.32**

A.2.2 HR-RWS-MPC-BM

Table A6. Correlation between objectives, and classifiers, for the HR-RWS-MPC-BM combination.

Objectives	Objective algorithms			Classifier algorithms				
Obj 1 - Obj 2		PCFS	FR	LDA	SVML	SVMP	ANN	KNN
PCFS - FR	PCFS	-	**-0.31**	**0.66**	**0.69**	**0.58**	**0.64**	**0.41**
	FR	**-0.31**	-	**-0.12**	**-0.14**	0.01	**-0.22**	0.01
Obj 1 - Obj 2		PCFS	FR	LDA	SVML	SVMP	ANN	KNN
PCFS - KNN	PCFS	-	**0.32**	**0.64**	**0.62**	**0.61**	**0.6**	**0.55**
	KNN	**0.32**	-	**0.36**	**0.38**	**0.41**	**0.33**	**0.65**
Obj 1 - Obj 2		KNN_tra	KNN_val	LDA	SVML	SVMP	ANN	KNN
KNN_tra - KNN_val	KNN_tra	-	0.07	**0.44**	**0.39**	**0.33**	**0.43**	**0.29**
	KNN_val	0.07	-	**0.55**	**0.49**	**0.41**	**0.51**	**0.33**

A.2.3 HR-SUS-UC-BM

Table A7. Correlation between objectives, and classifiers, for the HR-SUS-UC-BM combination.

Objectives	Objective algorithms			Classifier algorithms				
Obj 1 - Obj 2		PCFS	FR	LDA	SVML	SVMP	ANN	KNN
PCFS - FR	PCFS	-	**-0.35**	**0.67**	**0.69**	**0.61**	**0.62**	**0.42**
	FR	**-0.35**	-	**-0.21**	**-0.2**	-0.06	**-0.27**	-0.03
Obj 1 - Obj 2		PCFS	FR	LDA	SVML	SVMP	ANN	KNN
PCFS - KNN	PCFS	-	**0.32**	**0.65**	**0.63**	**0.64**	**0.64**	**0.55**
	KNN	**0.32**	-	**0.41**	**0.34**	**0.39**	**0.35**	**0.68**
Obj 1 - Obj 2		KNN_tra	KNN_val	LDA	SVML	SVMP	ANN	KNN
KNN_tra - KNN_val	KNN_tra	-	0.07	**0.4**	**0.4**	**0.38**	**0.41**	**0.29**
	KNN_val	0.07	-	**0.57**	**0.51**	**0.49**	**0.53**	**0.4**

A.2.4 HR-RWS-SPC-BM

Table A8. Correlation between objectives, and classifiers, for the HR-RWS-SPC-BM combination.

Objectives	Objective algorithms			Classifier algorithms				
Obj 1 - Obj 2		PCFS	FR	LDA	SVML	SVMP	ANN	KNN
PCFS - FR	PCFS	-	**-0.38**	**0.71**	**0.73**	**0.67**	**0.65**	**0.43**
	FR	**-0.38**	-	**-0.18**	**-0.2**	**-0.11**	**-0.26**	-0.03
Obj 1 - Obj 2		PCFS	FR	LDA	SVML	SVMP	ANN	KNN
PCFS - KNN	PCFS	-	**0.35**	**0.68**	**0.66**	**0.64**	**0.64**	**0.55**
	KNN	**0.35**	-	**0.41**	**0.4**	**0.4**	**0.37**	**0.63**
Obj 1 - Obj 2		KNN_tra	KNN_val	LDA	SVML	SVMP	ANN	KNN
KNN_tra - KNN_val	KNN_tra	-	**0.08**	**0.44**	**0.38**	**0.38**	**0.43**	**0.28**
	KNN_val	**0.08**	-	**0.58**	**0.49**	**0.51**	**0.55**	**0.31**

A.2.5 NHR-TOS-UC-BM

Table A9. Correlation between objectives, and classifiers, for the NHR-TOS-UC-BM combination.

Objectives	Objective algorithms			Classifier algorithms				
Obj 1 - Obj 2		PCFS	FR	LDA	SVML	SVMP	ANN	KNN
PCFS - FR	PCFS	-	**-0.14**	**0.63**	**0.67**	**0.38**	**0.67**	**0.25**
	FR	**-0.14**	-	**-0.08**	-0.05	**0.1**	-0.06	0
Obj 1 - Obj 2		PCFS	FR	LDA	SVML	SVMP	ANN	KNN
PCFS - KNN	PCFS	-	**0.2**	**0.65**	**0.64**	**0.41**	**0.65**	**0.39**
	KNN	**0.2**	-	**0.29**	**0.37**	**0.43**	**0.26**	**0.75**
Obj 1 - Obj 2		KNN_tra	KNN_val	LDA	SVML	SVMP	ANN	KNN
KNN_tra - KNN_val	KNN_tra	-	**0.08**	**0.44**	**0.48**	**0.23**	**0.44**	**0.16**
	KNN_val	**0.08**	-	**0.55**	**0.57**	**0.37**	**0.54**	**0.31**

A.2.6 NHR-TOS-SPC-BM

Table A10. Correlation between objectives, and classifiers, for the NHR-TOS-SPC-BM combination.

Objectives	Objective algorithms			Classifier algorithms				
Obj 1 - Obj 2		PCFS	FR	LDA	SVML	SVMP	ANN	KNN
PCFS - FR	PCFS	-	**-0.12**	**0.63**	**0.65**	**0.36**	**0.67**	**0.21**
	FR	**-0.12**	-	-0.01	-0.03	**0.09**	-0.01	0.02
Obj 1 - Obj 2		PCFS	FR	LDA	SVML	SVMP	ANN	KNN
PCFS - KNN	PCFS	-	**0.2**	**0.64**	**0.67**	**0.4**	**0.68**	**0.37**
	KNN	**0.2**	-	**0.3**	**0.38**	**0.5**	**0.25**	**0.73**
Obj 1 - Obj 2		KNN_tra	KNN_val	LDA	SVML	SVMP	ANN	KNN
KNN_tra - KNN_val	KNN_tra	-	**0.13**	**0.44**	**0.44**	**0.35**	**0.46**	**0.18**
	KNN_val	**0.13**	-	**0.56**	**0.59**	**0.41**	**0.59**	**0.3**

A.2.7 NHR-SUS-SPC-BM

Table A11. Correlation between objectives, and classifiers, for the NHR-SUS-SPC-BM combination.

Objectives	Objective algorithms			Classifier algorithms				
Obj 1 - Obj 2		PCFS	FR	LDA	SVML	SVMP	ANN	KNN
PCFS - FR	PCFS	-	**-0.11**	**0.65**	**0.65**	**0.42**	**0.68**	**0.2**
	FR	**-0.11**	-	**0.09**	0.07	**0.08**	**0.1**	**0.1**
Obj 1 - Obj 2		PCFS	FR	LDA	SVML	SVMP	ANN	KNN
PCFS - KNN	PCFS	-	**0.24**	**0.62**	**0.64**	**0.42**	**0.65**	**0.41**
	KNN	**0.24**	-	**0.29**	**0.36**	**0.55**	**0.31**	**0.74**
Obj 1 - Obj 2		KNN_tra	KNN_val	LDA	SVML	SVMP	ANN	KNN
KNN_tra - KNN_val	KNN_tra	-	**0.1**	**0.4**	**0.42**	**0.28**	**0.4**	**0.2**
	KNN_val	**0.1**	-	**0.59**	**0.58**	**0.4**	**0.59**	**0.34**

A.2.8 G30-TOS-GC-GM

Table A12. Correlation between objectives, and classifiers, for the G30-TOS-GC-GM combination.

Objectives	Objective algorithms			Classifier algorithms				
Obj 1 - Obj 2		PCFS	FR	LDA	SVML	SVMP	ANN	KNN
PCFS - FR	PCFS	-	**-0.24**	**0.66**	**0.63**	**0.63**	**0.64**	**0.35**
	FR	**-0.24**	-	**-0.21**	**-0.2**	**-0.16**	**-0.23**	**-0.07**
Obj 1 - Obj 2		PCFS	FR	LDA	SVML	SVMP	ANN	KNN
PCFS - KNN	PCFS	-	**0.37**	**0.71**	**0.7**	**0.63**	**0.72**	**0.55**
	KNN	**0.37**	-	**0.39**	**0.4**	**0.41**	**0.39**	**0.66**
Obj 1 - Obj 2		KNN_tra	KNN_val	LDA	SVML	SVMP	ANN	KNN
KNN_tra - KNN_val	KNN_tra	-	**-0.14**	**0.27**	**0.25**	**0.21**	**0.25**	**0.26**
	KNN_val	**-0.14**	-	**0.49**	**0.49**	**0.46**	**0.49**	**0.32**

A.2.9 G30-RWS-GC-GM

Table A13. Correlation between objectives, and classifiers, for the G30-RWS-GC-GM combination.

Objectives	Objective algorithms			Classifier algorithms				
Obj 1 - Obj 2		PCFS	FR	LDA	SVML	SVMP	ANN	KNN
PCFS - FR	PCFS	-	-0.05	**0.64**	**0.6**	**0.57**	**0.62**	**0.31**
	FR	-0.05	-	-0.02	0	-0.02	-0.02	-0.03
Obj 1 - Obj 2		PCFS	FR	LDA	SVML	SVMP	ANN	KNN
PCFS - KNN	PCFS	-	**0.38**	**0.66**	**0.63**	**0.69**	**0.63**	**0.54**
	KNN	**0.38**	-	**0.38**	**0.32**	**0.39**	**0.36**	**0.62**
Obj 1 - Obj 2		KNN_tra	KNN_val	LDA	SVML	SVMP	ANN	KNN
KNN_tra - KNN_val	KNN_tra	-	**-0.23**	**0.24**	**0.24**	**0.29**	**0.23**	**0.27**
	KNN_val	**-0.23**	-	**0.49**	**0.43**	**0.43**	**0.49**	**0.24**

A.2.10 G30-SUS-GC-GM

Table A14. Correlation between objectives, and classifiers, for the G30-SUS-GC-GM combination.

Objectives	Objective algorithms			Classifier algorithms				
Obj 1 - Obj 2		PCFS	FR	LDA	SVML	SVMP	ANN	KNN
PCFS - FR	PCFS	-	**0.13**	**0.63**	**0.63**	**0.55**	**0.62**	**0.31**
	FR	**0.13**	-	**0.22**	**0.18**	**0.14**	**0.2**	**0.14**
Obj 1 - Obj 2		PCFS	FR	LDA	SVML	SVMP	ANN	KNN
PCFS - KNN	PCFS	-	**0.35**	**0.64**	**0.62**	**0.67**	**0.64**	**0.51**
	KNN	**0.35**	-	**0.37**	**0.37**	**0.38**	**0.35**	**0.57**
Obj 1 - Obj 2		KNN_tra	KNN_val	LDA	SVML	SVMP	ANN	KNN
KNN_tra - KNN_val	KNN_tra	-	**-0.21**	**0.23**	**0.24**	**0.26**	**0.23**	**0.27**
	KNN_val	**-0.21**	-	**0.52**	**0.5**	**0.43**	**0.5**	**0.31**

A.2.11 G600-TOS-GC-GM

Table A15. Correlation between objectives, and classifiers, for the G600-TOS-GC-GM combination.

Objectives	Objective algorithms			Classifier algorithms				
Obj 1 - Obj 2		PCFS	FR	LDA	SVML	SVMP	ANN	KNN
PCFS - FR	PCFS	-	-0.03	**0.58**	**0.53**	**0.61**	**0.57**	**0.22**
	FR	-0.03	-	**-0.08**	**-0.1**	-0.04	**-0.11**	**0.1**
Obj 1 - Obj 2		PCFS	FR	LDA	SVML	SVMP	ANN	KNN
PCFS - KNN	PCFS	-	**0.18**	**0.62**	**0.6**	**0.49**	**0.66**	**0.39**
	KNN	**0.18**	-	**0.24**	**0.32**	**0.54**	**0.17**	**0.75**
Obj 1 - Obj 2		KNN_tra	KNN_val	LDA	SVML	SVMP	ANN	KNN
KNN_tra - KNN_val	KNN_tra	-	**0.09**	**0.4**	**0.43**	**0.21**	**0.37**	**0.29**
	KNN_val	**0.09**	-	**0.58**	**0.58**	**0.37**	**0.6**	**0.33**

A.2.12 G600-RWS-GC-GM

Table A16. Correlation between objectives, and classifiers, for the G600-RWS-GC-GM combination.

Objectives	Objective algorithms			Classifier algorithms				
Obj 1 - Obj 2		PCFS	FR	LDA	SVML	SVMP	ANN	KNN
PCFS - FR	PCFS	-	**0.13**	**0.63**	**0.57**	**0.68**	**0.61**	**0.26**
	FR	**0.13**	-	-0.03	0.01	**0.12**	0.03	**-0.1**
Obj 1 - Obj 2		PCFS	FR	LDA	SVML	SVMP	ANN	KNN
PCFS - KNN	PCFS	-	**0.15**	**0.62**	**0.69**	**0.5**	**0.64**	**0.43**
	KNN	**0.15**	-	**0.21**	**0.2**	**0.54**	**0.12**	**0.76**
Obj 1 - Obj 2		KNN_tra	KNN_val	LDA	SVML	SVMP	ANN	KNN
KNN_tra - KNN_val	KNN_tra	-	0.02	**0.37**	**0.33**	**0.29**	**0.35**	**0.26**
	KNN_val	0.02	-	**0.58**	**0.53**	**0.37**	**0.56**	**0.25**

A.2.13 G600-SUS-GC-GM

Table A17. Correlation between objectives, and classifiers, for the G600-SUS-GC-GM combination.

Objectives	Objective algorithms			Classifier algorithms				
Obj 1 - Obj 2		PCFS	FR	LDA	SVML	SVMP	ANN	KNN
PCFS - FR	PCFS	-	**0.07**	**0.64**	**0.57**	**0.69**	**0.63**	**0.28**
	FR	**0.07**	-	**0.09**	**0.12**	**0.13**	0.03	**0.1**
Obj 1 - Obj 2		PCFS	FR	LDA	SVML	SVMP	ANN	KNN
PCFS - KNN	PCFS	-	**0.21**	**0.64**	**0.68**	**0.48**	**0.69**	**0.46**
	KNN	**0.21**	-	**0.23**	**0.17**	**0.5**	**0.18**	**0.75**
Obj 1 - Obj 2		KNN_tra	KNN_val	LDA	SVML	SVMP	ANN	KNN
KNN_tra - KNN_val	KNN_tra	-	0.06	**0.37**	**0.35**	**0.22**	**0.39**	**0.21**
	KNN_val	0.06	-	**0.54**	**0.56**	**0.41**	**0.55**	**0.33**

A.3 Statistical validation

This section presents a statistical validation, where Wilcoxon's signed rank test is used, to evaluate whether two groups have the same continuous distribution. The section is divided into multiple subsection, where each subsection represents one NSGA-II combination. The statistics presented in this section corresponds to the most significant combinations, which is the three best performing combinations from each representation, presented in Table 1. Each subsection contains two tables, where the first table represents a statistical validation between each pair of objectives, which objectives that is statistically better than another pair of objectives for that combination. The second table has the statistical validation between the classifiers, which classifier is statistically better than another, for the specific combination.

Each table will use the following format:

- $p \geq 0.05$: blank space
- $p < 0.05$: *
- $p < 0.01$: **
- $p < 0.001$: ***

Meaning that the null-hypotheses of one algorithm (*alg1*) being better, or equal to another (*alg2*), can be rejected, at a statistical significance of 0.1% ($p < 0.001$), 1% ($p < 0.01$), and 5% ($p < 0.05$). A smaller p-value results one pair of objectives, or one classifier, is statistically better than another pair of objectives, or classifier. If the p-value ≥ 0.05, the null-hypotheses is true, *alg1* is worse, or equal to *alg2*.

A.3.1 HR-TOS-UC-BM

Table A18. Statistical validation between the objectives, for combination HR-TOS-UC-BM.

	Obj 1-Obj 2	Obj 1-Obj 2	Obj 1-Obj 2
Obj 1-Obj 2	PCFS-FR	PCFS-KNN	KNN_tra-KNN_val
PCFS-FR	-		
PCFS-KNN	***	-	*
KNN_tra-KNN_val			-

Table A19. Statistical validation between the classifiers, for combination HR-TOS-UC-BM.

	Algorithm	Algorithm	Algorithm	Algorithm	Algorithm
Algorithm	LDA	SVML	SVMP	ANN	KNN
LDA	-	**	***	***	***
SVML		-	***		***
SVMP			-		**
ANN			***	-	***
KNN					-

A.3.2 HR-RWS-MPC-BM

Table A20. Statistical validation between the objectives, for combination HR-RWS-MPC-BM.

	Obj 1-Obj 2	Obj 1-Obj 2	Obj 1-Obj 2
Obj 1-Obj 2	PCFS-FR	PCFS-KNN	KNN_tra-KNN_val
PCFS-FR	-		
PCFS-KNN	***	-	***
KNN_tra-KNN_val			-

Table A21. Statistical validation between the classifiers, for combination HR-RWS-MPC-BM.

	Algorithm	Algorithm	Algorithm	Algorithm	Algorithm
Algorithm	LDA	SVML	SVMP	ANN	KNN
LDA	-	***	***	***	***
SVML		-	***		***
SVMP			-		***
ANN			***	-	***
KNN					-

A.3.3 HR-SUS-UC-BM

Table A22. Statistical validation between the objectives, for combination HR-SUS-UC-BM.

	Obj 1-Obj 2	Obj 1-Obj 2	Obj 1-Obj 2
Obj 1-Obj 2	PCFS-FR	PCFS-KNN	KNN_tra-KNN_val
PCFS-FR	-		
PCFS-KNN	***	-	***
KNN_tra-KNN_val			-

Table A23. Statistical validation between the classifiers, for combination HR-SUS-UC-BM.

	Algorithm	Algorithm	Algorithm	Algorithm	Algorithm
Algorithm	LDA	SVML	SVMP	ANN	KNN
LDA	-	***	***	***	***
SVML		-	***		***
SVMP			-		***
ANN			***	-	***
KNN					-

A.3.4 NHR-TOS-UC-BM

Table A24. Statistical validation between the objectives, for combination NHR-TOS-UC-BM.

	Obj 1-Obj 2	Obj 1-Obj 2	Obj 1-Obj 2
Obj 1-Obj 2	PCFS-FR	PCFS-KNN	KNN_tra-KNN_val
PCFS-FR	-		
PCFS-KNN		-	*
KNN_tra-KNN_val			-

Table A25. Statistical validation between the classifiers, for combination NHR-TOS-UC-BM.

	Algorithm	Algorithm	Algorithm	Algorithm	Algorithm
Algorithm	LDA	SVML	SVMP	ANN	KNN
LDA	-	***	***	***	***
SVML		-	***		***
SVMP			-		***
ANN		*	***	-	***
KNN					-

A.3.5 NHR-TOS-SPC-BM

Table A26. Statistical validation between the objectives, for combination NHR-TOS-SPC-BM.

	Obj 1-Obj 2	Obj 1-Obj 2	Obj 1-Obj 2
Obj 1-Obj 2	PCFS-FR	PCFS-KNN	KNN_tra-KNN_val
PCFS-FR	-		
PCFS-KNN	*	-	
KNN_tra-KNN_val			-

Table A27. Statistical validation between the classifiers, for combination NHR-TOS-SPC-BM.

	Algorithm	Algorithm	Algorithm	Algorithm	Algorithm
Algorithm	LDA	SVML	SVMP	ANN	KNN
LDA	-	***	***	***	***
SVML		-	***		***
SVMP			-		***
ANN		**	***	-	***
KNN					-

A.3.6 NHR-SUS-SPC-BM

Table A28. Statistical validation between the objectives, for combination NHR-SUS-SPC-BM.

	Obj 1-Obj 2	Obj 1-Obj 2	Obj 1-Obj 2
Obj 1-Obj 2	PCFS-FR	PCFS-KNN	KNN_tra-KNN_val
PCFS-FR	-		
PCFS-KNN		-	
KNN_tra-KNN_val			-

Table A29. Statistical validation between the classifiers, for combination NHR-SUS-SPC-BM.

	Algorithm	Algorithm	Algorithm	Algorithm	Algorithm
Algorithm	LDA	SVML	SVMP	ANN	KNN
LDA	-	***	***	***	***
SVML		-	***		***
SVMP			-		***
ANN		***	***	-	***
KNN					-

A.3.7 G30-TOS-GC-GM

Table A30. Statistical validation between the objectives, for combination G30-TOS-GC-GM.

	Obj 1-Obj 2	Obj 1-Obj 2	Obj 1-Obj 2
Obj 1-Obj 2	PCFS-FR	PCFS-KNN	KNN_tra-KNN_val
PCFS-FR	-		***
PCFS-KNN	**	-	***
KNN_tra-KNN_val			-

Table A31. Statistical validation between the classifiers, for combination G30-TOS-GC-GM.

	Algorithm	Algorithm	Algorithm	Algorithm	Algorithm
Algorithm	LDA	SVML	SVMP	ANN	KNN
LDA	-	***	***	***	***
SVML		-	***		***
SVMP			-		***
ANN		***	***	-	***
KNN					-

A.3.8 G30-RWS-GC-GM

Table A32. Statistical validation between the objectives, for combination G30-RWS-GC-GM.

	Obj 1-Obj 2	Obj 1-Obj 2	Obj 1-Obj 2
Obj 1-Obj 2	PCFS-FR	PCFS-KNN	KNN_tra-KNN_val
PCFS-FR	-		***
PCFS-KNN	***	-	***
KNN_tra-KNN_val			-

Table A33. Statistical validation between the classifiers, for combination G30-RWS-GC-GM.

	Algorithm	Algorithm	Algorithm	Algorithm	Algorithm
Algorithm	LDA	SVML	SVMP	ANN	KNN
LDA	-	***	***	***	***
SVML		-	***		***
SVMP			-		***
ANN		***	***	-	***
KNN					-

A.3.9 G30-SUS-GC-GM

Table A34. Statistical validation between the objectives, for combination G30-SUS-GC-GM.

	Obj 1-Obj 2	Obj 1-Obj 2	Obj 1-Obj 2
Obj 1-Obj 2	PCFS-FR	PCFS-KNN	KNN_tra-KNN_val
PCFS-FR	-		***
PCFS-KNN		-	***
KNN_tra-KNN_val			-

Table A35. Statistical validation between the classifiers, for combination G30-SUS-GC-GM.

	Algorithm	Algorithm	Algorithm	Algorithm	Algorithm
Algorithm	LDA	SVML	SVMP	ANN	KNN
LDA	-	***	***	***	***
SVML		-	***		***
SVMP			-		***
ANN		***	***	-	***
KNN					-

A.3.10 G600-TOS-GC-GM

Table A36. Statistical validation between the objectives, for combination G600-TOS-GC-GM.

	Obj 1-Obj 2	Obj 1-Obj 2	Obj 1-Obj 2
Obj 1-Obj 2	PCFS-FR	PCFS-KNN	KNN_tra-KNN_val
PCFS-FR	-		
PCFS-KNN		-	
KNN_tra-KNN_val			-

Table A37. Statistical validation between the classifiers, for combination G600-TOS-GC-GM.

	Algorithm	Algorithm	Algorithm	Algorithm	Algorithm
Algorithm	LDA	SVML	SVMP	ANN	KNN
LDA	-	***	***	***	***
SVML		-	***		***
SVMP			-		***
ANN		***	***	-	***
KNN					-

A.3.11 G600-RWS-GC-GM

Table A38. Statistical validation between the objectives, for combination G600-RWS-GC-GM.

	Obj 1-Obj 2	Obj 1-Obj 2	Obj 1-Obj 2
Obj 1-Obj 2	PCFS-FR	PCFS-KNN	KNN_tra-KNN_val
PCFS-FR	-		
PCFS-KNN		-	
KNN_tra-KNN_val			-

Table A39. Statistical validation between the classifiers, for combination G600-RWS-GC-GM.

	Algorithm	Algorithm	Algorithm	Algorithm	Algorithm
Algorithm	LDA	SVML	SVMP	ANN	KNN
LDA	-	***	***	***	***
SVML		-	***		***
SVMP			-		***
ANN		***	***	-	***
KNN					-

A.3.12 G600-SUS-GC-GM

Table A40. Statistical validation between the objectives, for combination G600-SUS-GC-GM.

	Obj 1-Obj 2	Obj 1-Obj 2	Obj 1-Obj 2
Obj 1-Obj 2	PCFS-FR	PCFS-KNN	KNN_tra-KNN_val
PCFS-FR	-		
PCFS-KNN		-	
KNN_tra-KNN_val			-

Table A41. Statistical validation between the classifiers, for combination G600-SUS-GC-GM.

	Algorithm	Algorithm	Algorithm	Algorithm	Algorithm
Algorithm	LDA	SVML	SVMP	ANN	KNN
LDA	-	***	***	***	***
SVML		-	***		***
SVMP			-		***
ANN		***	***	-	***
KNN					-