Wrapper Methods for Multi-Objective Feature Selection

Uchechukwu F. Njoku
Universitat Politècnica de Catalunya
Barcelona, Spain
unjoku@essi.upc.edu

Alberto Abelló
Universitat Politècnica de Catalunya
Barcelona, Spain
aabello@essi.upc.edu

Besim Bilalli
Universitat Politècnica de Catalunya
Barcelona, Spain
bbilalli@essi.upc.edu

Gianluca Bontempi
Université Libre de Bruxelles
Brussels, Belgium
gianluca.bontempi@ulb.be

ABSTRACT

The ongoing data boom has democratized the use of data for improved decision-making. Beyond gathering voluminous data, preprocessing the data is crucial to ensure that their most relevant aspects are considered during the analysis. Feature Selection (FS) is one integral step in data preprocessing for reducing data dimensionality and preserving the most relevant features of the data. FS can be done by inspecting inherent associations among the features in the data (filter methods) or using the model performance of a concrete learning algorithm (wrapper methods).

In this work, we extensively evaluate a set of FS methods on 32 datasets and measure their effect on model performance, stability, scalability and memory usage. The results re-establish the superiority of wrapper methods over filter methods in model performance. We further investigate the unique role of wrapper methods in multi-objective FS with a focus on two traditional metrics - accuracy and Area Under the ROC Curve (AUC). On model performance, our experiments showed that optimizing for both metrics simultaneously, rather than using a single metric, led to improvements in the accuracy and AUC trade-off[1] up to 5% and 10%, respectively.

1 INTRODUCTION

The process of transforming raw data into meaningful insights involves various stages, from data collection, preprocessing, and storage, to analysis. One of the difficulties faced in the preprocessing stage is the presence of irrelevant or redundant features in the data, often paired with insufficient instances (e.g., in bioinformatics [30]). Datasets of this type are challenging to analyze due to poor data understanding and visualization, model overfitting, and complex model explanation [23].

To solve the challenges mentioned above, the noise that comes from irrelevant or redundant features is diminished through dimensionality reduction. This involves combining features into new fewer ones (Feature Extraction -FE) or identifying the most relevant features (Feature Selection -FS). Both approaches reduce the dimensions of the data to build simpler and more explainable models.

In this work, we focus on wrapper methods which often use the performance metric of a Machine Learning (ML) algorithm (e.g., accuracy) as the *evaluation criterion* to measure the relevance of a set of features and obtain the set that maximizes the evaluation criterion. Typically, wrapper methods are based on a single (mono-objective) evaluation criterion, however, we argue the need for multiple (multi-objective) criteria to be used as a more realistic approach in determining the relevance of features since models cannot be fully described by a single metric [16]. We also empirically demonstrate the advantage of multi-objective FS (which selects features with a better balance among multiple metrics of interest) over mono-objective.

After selecting the subset of features, further evaluation is necessary to understand the FS method's selection process and ascertain the reliability of the chosen features. We do this by answering questions such as *What is the method's performance? Is it memory efficient? and How robust are the selected features to slight changes from sampling and shuffling the data?*

In this work, we first survey existing tools and libraries for wrapper FS, detailing their specificities and comparing them based on various criteria. Next, we show that the evaluation criterion (e.g., the accuracy of a classifier) is the chief determinant in selecting features and how, even for the same classifier, a change in the evaluation criteria leads to a different subset of features. Lastly, we demonstrate the advantage of considering multiple evaluation criteria in FS. Using 32 datasets and four classification algorithms, we extensively study ten FS methods by measuring:

(1) the effect they have on the accuracy and AUC of the models after training over the subset of selected features;
(2) the stability of each method;
(3) the average execution time and memory footprint.

On the view of these results, we emphasize the need for multi-objective criteria in FS by considering accuracy and AUC metrics simultaneously. We show that compared to a single measure, a multi-objective wrapper method produces a feature subset with a better balance between the accuracy and AUC of the classifiers.

In particular, the contributions of this work can be summarized as follows:

(1) Extensive evaluation of the predictive performance and stability of existing wrapper and filter methods in Python[2] libraries. The results of our experiment show:
   - that wrapper methods outperform filter methods in terms of both accuracy and AUC;
   - filter methods are more stable than wrapper methods although the variance in accuracy and AUC caused by the instability of the latter is negligible.

[1] We define *trade-off* as the compromise in one objective as a consequence of considering a second objective.

[2] Python continues to be the most preferred language for scientific computing, data science, and machine learning [29].

(2) Empirical comparison of multi-objective and mono-objective wrapper FS by considering two well-known metrics, accuracy and AUC. Our results show that on average, the multi-objective FS method produces feature subsets with a better combined accuracy and AUC compared to using just one of these metrics.
(3) Analysis of the scalability and memory footprint of wrapper methods. The results show that wrapper methods are memory efficient but lack scalability.

2 PRELIMINARIES

There are broadly four steps in FS - (1) feature subset generation, (2) subset evaluation, (3) stopping criteria, and (4) result evaluation. In the first step, a search strategy is used to generate subsets of features which are then evaluated during the second step using an evaluation criterion. Also, FS methods are grouped into three categories; *filter*, *wrapper*, and *embedded* methods [14], depending on the evaluation criteria they use in this step. This cycle of generating and evaluating feature subsets is controlled by a stopping criterion, e.g., by setting a maximum number of evaluations or iterations. Lastly, the final selected subset is evaluated depending on the analysis task, for example, by using an ML algorithm. This section discusses the three categories of FS methods, the various search strategies, and the evaluation criteria.

2.1 Classes of FS

FS methods are classified into *filter*, *wrapper*, or *embedded*; based on their reliance on ML algorithms for feature subset evaluation - to select the relevant features.

Filter methods. Filter methods typically perform two out of the four FS steps; the subset evaluation and the result evaluation steps. These methods do not use ML algorithms in the subset evaluation step but rather rely on the inherent characteristics, and associations among the features to determine their relevance. Some filter methods measure the association between each predictive feature and the target feature using information gain, correlation, or distance measures [23]. While others, additionally consider the relationship among predictive features to reduce the likelihood of selecting redundant features. The advantages of filter methods are the generalizability of their results to any ML model, faster computation time over wrapper or embedded methods, and robustness to overfitting. However, since the selection is model agnostic, selected features could be non 'optimal' for the final ML algorithm. Also, they risk selecting redundant features when the associations among predictive features are not considered. This work evaluates six filter methods - CMIM, Gini, JMI, MRMR, Relief, and SPEC. For a detailed discussion of these methods and their performance, we refer the reader to our previous work [23].

Wrapper methods. These methods consist of all four FS steps. Particularly, the generation of subsets from the search space (i.e., all possible subsets of features) is guided by a search strategy, which could be exhaustive, population-based, or sequential (see section 2.2). Furthermore, wrapper methods typically depend on the performance metrics (e.g., accuracy or AUC) of a chosen ML algorithm to evaluate generated feature subsets. The iterative nature of these methods make them computationally intensive even with a moderate number of features. Nevertheless, their dependence on the ML algorithm ensures that selected features favor the algorithm of interest. Also, applying cross-validation (i.e., resampling) during model building reduces the risk of overfitting the ML algorithm [3]. Using four classification algorithms, and two performance metrics (accuracy and AUC), this work highlights the strengths and weaknesses of wrapper methods and their unique role in multi-objective FS.

Embedded methods. Embedded methods are ML algorithms that intrinsically do feature selection while building the model. An example is decision trees [14] which prune out less relevant features by using criteria like *gini* or *entropy*. Embedded methods also have the advantage of selecting features optimized for the ML algorithm with a better computational performance than wrapper methods. However, the results (i.e., the selected subset of features) are not generalizable because they are tailored to the ML algorithm. The outcome of embedded methods are typically the trained models and not feature subsets, for this reason, we do not include them in our analysis.

2.2 Search strategies

The search space contains all the possible feature subsets that FS methods can select, and search strategies determine how FS methods traverse this search space.

Exhaustive search. This strategy traverses the entire search space and evaluates every possible set of features. It is guaranteed to find the optimal feature subset w.r.t the evaluation criterion. However, it is computationally intensive and infeasible even for a moderate number of features [10].

Population-based search. Since it is infeasible to traverse the entire search space, this strategy applies heuristics that imitate natural population evolution to obtain near-optimal results. It starts with an initial set of feature subsets called population and iteratively updates it based on the heuristics until a stopping criterion is reached [10]. Genetic algorithms, particle swarm optimization, and bee colony algorithms are examples of population-based search strategies used in feature selection [10].

Sequential search. These simple greedy algorithms consecutively update the feature subset with features that improve or maintain the evaluation criterion. The two variants of this strategy are Sequential Forward Selection (SFS) and Sequential Backward Selection (SBS). The former begins with an empty set and at each iteration adds features that improve or preserve the evaluation criterion until a stopping criterion is satisfied (e.g., the maximum number of iterations or feature subset size). The latter conversely begins with the complete set of features and, at each iteration, drops off features that decrease the evaluation criterion [10]. In this work, we use the SFS search strategy in wrapper methods, due to it's simplicity and resilience against overfitting [14].

2.3 Evaluation criteria

The second and fourth steps of the FS process focus on the evaluation, during and after the selection respectively. We distinguish between the criteria used to score the relevance of a feature subset **during** the selection and those used **after** selection to both evaluate the final subset and understand the FS method's efficiency.

Internal evaluation criteria. They refer to the metrics used to score feature subsets during the FS process. For filter and embedded methods, they include information theory measures (such as gini or entropy), distance measures, and correlation

measures. For wrapper methods, they include model performance metrics (such as accuracy, AUC, and Kappa index), the size of the feature subset, and other user-defined metrics. These criteria influence the traversal of wrapper methods through the search space and thus determine the final feature subset. In wrapper methods, more than one criterion can be combined into a new utility criterion for multi-objective FS.

External evaluation criteria. These criteria focus on evaluating the finally selected feature subset and the overall FS process in ways that do not contribute to or affect the selection process and outcome. As examples, we can mention the method's stability which measures resistance to slight changes in the data. We can also mention execution time, resource usage such as memory footprint, and other performance metrics except the ones used during selection. These criteria give insights that facilitate the explainability of the methods.

In evaluating subsets of features using internal evaluation criteria, one criterion is usually used in a mono-objective fashion to optimize the model's predictive performance. However, it is essential to consider multiple criteria while selecting features since in ML, models cannot be fully described by a single metric[16]; this is called multi-objective feature selection .

3 MULTI-OBJECTIVE FEATURE SELECTION (MOFS)

The feature selection problem is an optimization problem where the chosen evaluation criterion is either maximized or minimized. When we use multiple criteria for selection, it becomes a multi-objective optimization problem. There are two approaches to solve it, which we detail below.

Scalarization (preference-based) method. This approach forms a composite objective function from the sum of the weighted normalized objectives [7]; the weight assigned to an objective reflects its relevance or importance. In the case of no preferences, the objectives receive equal weights. This approach is simple and suitable when a preference is known a priori.

Pareto method. This method produces a collection of solutions that are trade-off optimal (Pareto-optimal) solutions, i.e., solutions for which we can find no other solution which improves any of the considered objectives without deteriorating the other(s) [7]. Algorithms that apply heuristics such as genetic algorithms are used to find these Pareto-optimal solutions. At the end, one out of the collection of solutions is chosen based on other subjective knowledge.

In ML, a single metric alone is insufficient to describe models completely [16]; this is why many evaluation metrics exist to capture the various aspects of ML models. Given this, we do not rely on mono-objective feature selection but rather consider multiple metrics when measuring the features' relevance, which motivates MOFS. Out of the three classes of FS, only wrapper methods give us the freedom to combine criteria for the subset evaluation step. This makes wrapper methods play a unique role in enabling MOFS. This work compares the mono-objective and multi-objective wrapper methods that use the SFS search strategy. Although the multi-objective and mono-objective wrapper methods considered in this work use the same SFS search strategy, the main difference lies in how the features' relevance is measured, i.e., the evaluation criteria. We focus on a MOFS method that follows the scalarization approach to combine two predictive performance metrics - accuracy and AUC. In contrast, the mono-objective wrapper method considers each metric separately. Our experiments highlight the benefits of MOFS, in general.

3.1 SFS search strategy

For this work, we chose the SFS method as presented in Algorithm 1, due to its simplicity and resilience against overfitting.

Algorithm 1 Sequential Forward Selection (SFS) Algorithm

Input: $clf()$, $measure()$, k, M // classifier, evaluation criteria, number of features to select, and complete set of features
Output: S // set of selected features

```
1:  S ← {}
2:  if (k < |M|) then
3:      score ← 0
4:      do
5:          modified ← false
6:          for each feature ∈ M do
7:              S_temp ← (S ∪ {feature})
8:              model ← clf(S_temp)
9:              metric ← measure(model)
10:             if (metric >= score) then
11:                 modified ← true
12:                 selected ← feature
13:                 score ← metric
14:             end if
15:         end for
16:         if modified then
17:             S ← (S ∪ {selected})
18:             M ← (M \ {selected})
19:         end if
20:     while (modified and (|S| < k))
21: else
22:     print: "k must be less than |M|"
23: end if
24: return S
```

The input parameters for the SFS algorithm for FS are $clf()$ - any classifier, $measure()$ - an evaluation criterion, k - the desired number of features to select, and M - the complete set of features in the dataset. Following the SFS approach, we initialize the set of selected features to an empty set - S (line 1). Next, we ensure that the number of features to select is less than the total number of features (line 2); otherwise, an error message is generated, the algorithm ends, and an empty set is returned (lines 21-24). We use the variable $modified$ as a flag to detect if a feature was selected in an iteration. It is set to $true$ only when we find a feature that increases or preserves the subset score (line 10). The score of S is tracked by the variable $score$ initially set to 0 (line 3), which stores the best metric value of S after each iteration. The $metric$ variable holds the value of the evaluation criteria derived from the $measure$ function (line 9) which, in our case, are accuracy and AUC for mono-objective FS; and a linear combination of both for multi-objective FS.

In each iteration, we add each unselected feature to S to train $clf()$ and get the $metric$ through the $measure$ function (lines 6-9). Afterwards, the feature which most increases the $score$ or preserves it is added to S (lines 16-19). The number of iterations is limited to the number of features to be selected and $score$ change (line 20), i.e., if k features have been selected or if at the end of an iteration no feature improves or preserves the subset score,

then the selection process ends and the subset of features, S is returned (line 24).

As part of our experiments, we consider two performance metrics - accuracy and AUC. We pass them separately as the *measure*() function for mono-objective FS. For MOFS (multi-objective), we linearly combine them with equal preference by using the scalarization approach that forms a new evaluation criterion used in *measure*().

4 EXPERIMENTAL EVALUATION

As part of our study of existing feature selection methods, we conducted a set of robust experiments according to the systematic approach described in [17]. In what follows, we describe the ten steps of our experiments.

(1) **Define goals and system**
The purpose of these experiments is, first, to re-establish the superior performance of wrapper methods over filter methods for feature selection, and secondly, to highlight the unique role of wrapper methods in MOFS. The system we consider is comprised of 32 datasets (16 with binary target feature and 16 with multi-class target feature), six filter methods, and four wrapper methods (ten FS methods overall), using four classification algorithms - Naive Bayes (NB), K-Nearest Neighbors (KNN), Decision Trees (DT), and Support Vector Machines (SVM).

(2) **Define services and outcomes**
The FS methods and the classification algorithms are the parts of the system that take inputs and produce outputs. The first outcome of the system is the generation of four base classifiers from the four classification algorithms for each dataset. The subsequent outcome is ten feature subsets found for each dataset. Using these feature subsets, new classifiers are built, which are compared with the baseline classifiers in order to evaluate the influence of each FS method on the performance of the model.

(3) **Choose metrics**
The five metrics used to evaluate the system include the impact on the model performance in the form of percentage change in accuracy and AUC, runtime and peak memory usage of the FS methods, and the stability of the selected feature subsets.

Internal evaluation metrics.

- **Relative change in accuracy** quantifies the accuracy lost or gained as a result of reducing the number of features where accuracy measures the amount of correctly classified unseen samples.

$$Accuracy\ change^{3}(\%) = \left(\frac{New\ Accuracy - Base\ Accuracy}{Base\ Accuracy}\right) \times 100. \quad (1)$$

- **Relative change in AUC** similarly measures the AUC gain or loss after feature selection. The AUC considers not only the 'positively' classified unseen samples which are right, but also the 'positively' classified samples which are wrong.

$$AUC\ change^{4}(\%) = \left(\frac{New\ AUC - Base\ AUC}{Base\ AUC}\right) \times 100. \quad (2)$$

[3]There is an expected increase in accuracy i.e., New Accuracy > Base Accuracy.
[4]There is an expected increase in AUC i.e., New AUC > Base AUC.

External evaluation metrics.

- **Runtime** of the FS methods primarily allows to compare the various implementations available in the Python libraries but also to restate the need for wrapper methods to be made scalable considering their superior impact on predictive performance.
- **Peak memory usage** shows the peak size of memory block used by the FS method during execution. This is particularly relevant when resources are paid for and could be a guide to both provisioning resources and choosing FS methods.
- Finally, **stability** quantifies the robustness of FS methods to changes in the dataset. The most stable methods return the same feature subsets with slight changes in the dataset, while unstable methods yield different results from small changes in the dataset. The small change can come from data sampling and shuffling usually used in cross validation. To measure the stability of the FS methods, we use the metric in [24] defined as:

$$\hat{\Phi}(\mathbb{Z}) = 1 - \frac{\frac{1}{d}\sum_{f=1}^{d} s_f^2}{\frac{\bar{k}}{d}\left(1 - \frac{\bar{k}}{d}\right)}, \quad (3)$$

where $\mathbb{Z}$ is the set of feature subsets selected from the dataset by making little permutations in the dataset's rows and columns, d is the total number of features in the dataset, $\bar{k}$ is the average number of features selected over all feature subsets in $\mathbb{Z}$, and s_f^2 is the unbiased sample variance of selecting the f^{th} feature. $\hat{\Phi}$ has a range of $[0, 1]$, where 0 implies total instability and 1 means that the same feature subset is selected each time irrespective of the data changes.

(4) **Parameters**
The properties of the datasets that can influence the metrics measured in the system are referred to as **dataset parameters** and include:

- Number of features.
- Number of instances.
- Number of classes.
- Dimensionality.
- Class Balance.
- Class Entropy.
- Average feature correlation.

For a detailed discussion of these dataset parameters, we refer the reader to our previous work [23]. On the other hand, variable specifications of algorithms that have an impact on the output of the system are referred to as **algorithm parameters**. They include:

- Feature subset size, which is the maximum number of features to be selected; affects the results derived from the system.
- Hyper-parameters, which refer to the several parameters in FS and classification algorithms that can be set to different values to control the FS and model-building processes. These parameters affect the output of the algorithms and thus the system.

Lastly, there are characteristics of the hardware used to execute the experiments for the system performance evaluation that can also impact the outcomes, these are referred to as **system parameters** and include among others:

- Processor speed.

- RAM/Disk size.
- Operating system context switching overhead.

(5) **Choose factors to study**
We chose a few of the parameters discussed to be controlled in our experiments. Particularly, the number of features, number of instances, and class balance are considered in the selection of the datasets as described in [23]. Some other parameters are dependent on the factors chosen to study. For example, dimensionality depends on two of the considered factors, number of features and instances. For the system parameters, these are constant as one machine was used for all the experiments with non-essential applications closed to ensure the validity of results. Finally, the default algorithm parameters are used and the subset size is fixed at $\sqrt{\#features}$ since the focus of our analysis is on algorithm comparison and not on hyper-parameter tuning to find the best setting.

(6) **Define evaluation technique**
The experiments were implemented using Python 3.10 with three prominent packages that implement wrapper feature selection: *Mlxtend*[5], *Scikit-Feature*[6], *and Scikit-Learn*[7] detailed in Table 2. The latter also implements the classification algorithms. The *Tracemalloc*[8] library was used for measuring the execution and peak memory usage.

(7) **Define workload**
Following the systematic process for dataset selection described in [23], we used 32 datasets from OpenML[9], detailed in Table 1, for the experiments. These datasets are clean and numerical due to the input constraint of some algorithms used. In addition, we created a synthetic dataset of 1,600 predictive features, one binary target feature and 16,000 instances to evaluate the scalability and resource usage of wrapper methods.

(8) **Design experiment**
A total of 32 datasets and ten FS algorithms were considered - six filter methods (*CMIM, Gini, JMI, MRMR, ReliefF,* and *SPEC*) and four wrapper methods (SFS_{NB}, SFS_{KNN}, SFS_{DT}, and SFS_{SVM}). For the filter methods, a single implementation in the Scikit-Feature library was used to evaluate and measure the metrics. However, for the wrapper methods, multiple libraries were used. For SFS_{DT} and SFS_{SVM} all three libraries were used while for SFS_{NB} and SFS_{KNN}, only Mlxtend and Scikit-Learn were used because Scikit-Feature does not implement them.

(9) **Data analysis and interpretation**
We use descriptive analytics, in particular scatter plots to compare the various evaluation criteria used in wrapper methods, bar charts to show the variations in predictive performance as a result of data shuffling and sampling, and average rankings to compare FS methods based on the predictive performance of classifiers. With these, we analyze the results from the experiments and gain insights into the overall performance of the various FS algorithms in the system.

(10) **Report results**
The conclusions we reach from our analysis of the results are presented and discussed in Section 5.

[5]http://rasbt.github.io/mlxtend
[6]https://github.com/jundongl/scikit-feature
[7]https://scikit-learn.org/stable
[8]https://docs.python.org/3/library/tracemalloc
[9]https://www.openml.org

Table 1: Datasets and their characteristics.

OpenML ID	Features	Instances	Classes
1464	5	748	2
931	4	662	2
40983	6	4839	2
841	10	950	2
1061	30	107	2
834	101	250	2
40666	169	6598	2
41145	309	5832	2
1015	4	72	2
793	11	250	2
1021	11	5473	2
819	7	9517	2
1004	61	600	2
41966	618	600	2
995	48	2000	2
41158	971	3153	2
4340	6	383	3
1565	14	294	5
40496	8	500	10
1512	14	200	5
40498	12	4898	7
1540	4	9285	5
375	15	9961	9
1526	5	5456	4
275	63	71	6
277	63	74	4
377	61	600	6
1518	91	47	4
41972	221	9144	8
1560	36	2126	3
1468	857	1080	9
40499	41	5500	11

4.1 Tools and libraries

The need for FS during data preprocessing demands tools and libraries that implement different FS techniques. In Table 2, we provide a list of the available tools that implement wrapper methods. We list their characteristics with respect to the search strategies they use and the level of support in terms of scalability.

Table 2: Feature selection tools and libraries.

Tool	Lang.	Exhaustive	Population-based	Sequential	Parallelism
Weka	Java	X	X	✓	Partial
Scikit-Learn(SKL)	Python	X	X	✓	Partial
Rapidminer	Java	✓	✓	✓	Partial
Mlxtend(MLX)	Python	✓	X	✓	Yes
Scikit-Feature(SKF)	Python	X	X	✓	No
FeatureSelect	Matlab	X	✓	X	No

As shown in Table 2, the most commonly supported search strategy is the Sequential one. Its ample presence can be attributed to its simplicity. On the other hand, due to their complexity, the Exhaustive and Population-based strategies have very low support; only two out of six tools support these strategies. In terms of scalability, tools like SKF and FeatureSelect outrightly do not provide any form of parallelism. Weka, SKL and Rapidminer do support parallelism differently, each one of them in its own way. For Weka, the ability to use multiple processes is available only for the WrapperSubsetEval-GreedyStepwise method. For

SKL, it is available only for the SequentialFeatureSelector method, while for Rapidminer, a license is required because the studio free edition is limited to 10,000 rows and one Processor. However, MLX supports parallelism for all the available methods by simply using multiprocessing to evaluate different feature subsets in parallel. Finally, these tools implement mono-objective wrapper FS and do not support multiple evaluation criteria simultaneously.

4.2 Execution

The experiments were executed on a PC with eight 11th Gen Intel® Core™ i7-1185G7 @ 3.00GHz processors and 16 GB of memory, during which usage of non-essential applications was minimized to avoid interference.

For the accuracy predictive performance, we used all 32 datasets while we used the 16 binary datasets for the AUC, MOFS, and stability experiments.[10] For each of these datasets, we randomly shuffle the features and instances to introduce slight changes in the order of the data and produce ten variants for each dataset. For each of these dataset variants, the filter methods select features once while the wrapper methods select by the classification algorithm and evaluation metric pair (e.g., KNN & accuracy, KNN & AUC, NB & accuracy, e.t.c.). In all cases, we select $\sqrt{\#features}$ features from each dataset and report the predictive performance of the ten variants from each of the datasets. The selected features are used to build four classification models - NB, KNN, DT, and SVM - from which we measure the change in accuracy and AUC against the models built with the complete set of features. In building the classification models, we apply ten-fold cross-validation to mitigate overfitting [3].

For the scalability and memory experiments, we focused on the wrapper methods using DT classification and the synthetic datasets while also selecting $\sqrt{\#features}$ features. We repeated the wrapper experiments with the different libraries. Source code used for our experiments is available on Github.[11]

5 RESULTS AND DISCUSSION

One of the key reasons for performing FS is to improve data and model understanding by providing fewer relevant features to analyze. However, we also need to ensure the effectiveness of these FS methods through various measures like model performance, stability, scalability, and memory efficiency. In the following, we present the results of our experiments using all of these metrics to assess the effectiveness of ten FS methods over 32 datasets and four classification algorithms.

5.1 Model performance

Focusing on predictive performance, we compare the effect of filter and wrapper methods on four classifiers. This is the most common way to evaluate FS methods.

5.1.1 ***Relative change in accuracy.*** As defined in Equation 1, *Accuracy change* measures the impact of FS on the performance of a model, relative to the model's performance over all the features. By selecting relevant features, we expect this performance to increase or in the worst case, to remain the same. In Figure 1, we present the percentage of relative change in accuracy for the KNN classifier after each FS method (table columns) is applied to the 16 datasets (table rows). A positive percentage means an increase in accuracy, while a negative percentage means a decrease. Likewise, the color gradient represents the magnitude of change in accuracy, where dark blue means a high increase and dark red means a high decrease in accuracy. For each dataset (row), the FS method that results in the highest increase or least decrease in accuracy has the highest blue or least red shade, respectively. Also, we represent the wrapper result from all the libraries with one column, the last one, because all the libraries select the same features for all the classifiers except DT which is sensitive to the order of features in the input data.

We observe from Figure 1, that wrapper methods (SFS), have the highest number of blue colored cells. This shows that wrapper methods improve the accuracy of the classifiers with a higher percentage than filter methods. In Table 3, we summarize by average ranks the performances of the FS methods on all four classification algorithms.[12] In order to obtain the average rank, we begin by ranking all FS methods for each classifier. The FS method with the highest number (out of 16) of accuracy improvements is ranked first, and so on. Following that, the average of four ranks from the four classifiers is then calculated for each FS method. These averages are then ranked so that the FS method with the highest average rank comes first and the method with the least average rank comes last. We observe that the wrapper methods (SFS) rank first and outperform filter methods, with the JMI filter coming second and the SPEC method ranking last.

Table 3: Average rank of FS methods by accuracy change for four classifiers in binary problems.

Method	NB	KNN	DT	SVM	Average Rank
SFS	10	9	4	9	1
JMI	4	4	8	6	2
Gini	5	2	6	5	3
CMIM	3	2	4	5	4
MRMR	3	3	4	3	5
Relief	2	1	4	3	6
SPEC	0	2	1	2	7

To also gain perspective into multi-class problems, we experimented with the 16 multi-class datasets listed in Table 1. Table 4 shows a summary by average ranks of the FS methods accuracy change for the four classification algorithms on the 16 multi-class datasets. From Table 4, we see wrapper methods outperforming filter methods with an even greater margin in multi-class problems. We did not consider SPEC for the multi-class experiments due to it's poor performance in the binary experiments coupled with it's computational complexity.

5.1.2 ***Relative change in AUC.*** Similarly to *Accuracy change*, *AUC change* as defined in Equation 2 also measures the impact of FS on the model's predictive performance. The goal here is to see how the AUC of a model changes after applying FS. In Figure 2, we present the change in the AUC of the KNN model for each dataset after applying the FS methods. Also, the best performing FS method for each dataset is shaded with the darkest blue or lightest red. Again, this shows that wrapper methods outperform filter methods with the SFS column having the highest number and magnitude of blue cells. In Table 5, we also list the results

[10] AUC is strictly a binary classification metric.

[11] https://github.com/F-U-Njoku/Wrapper-Methods-in-Multi-Objective-Feature-Selection

[12] The supplementary results for other classifiers are available on the complementary Github page: https://github.com/F-U-Njoku/Wrapper-Methods-in-Multi-Objective-Feature-Selection

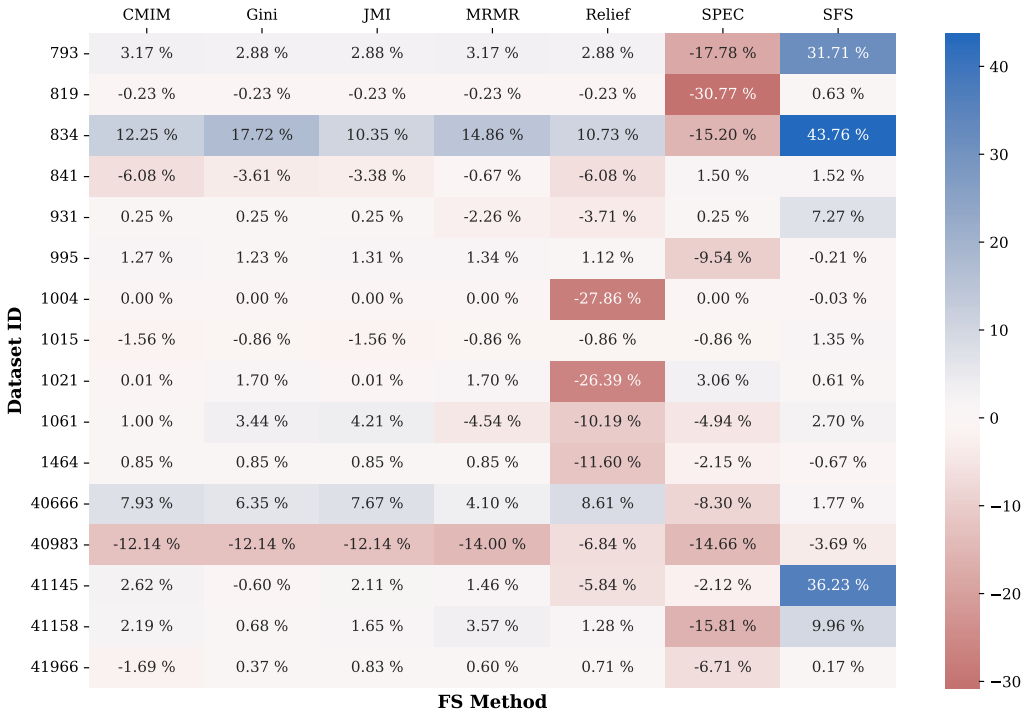


Figure 1: Change in accuracy of K-Nearest Neighbour classifier after feature selection.

Table 4: Average rank of FS methods by accuracy change for four classifiers in multi-class problems.

Method	NB	KNN	DT	SVM	Average Rank
SFS	15	11	12	12	1
JMI	2	3	4	4	2
CMIM	0	6	4	4	3
MRMR	1	4	4	4	3
Relief	2	0	0	2	5
Gini	0	0	0	2	6

obtained for the other learning algorithms where a similar pattern is observed, except for NB. This again shows that not one method of FS is suitable for every case. Finally, we sort them by their average rank for all the classification algorithms and see that the wrapper method ranks first, followed by the Gini filter and, lastly, SPEC.

Table 5: Average rank of feature selection methods by AUC change performance.

Method	NB	KNN	DT	SVM	Average Rank
SFS	5	13	7	8	1
Gini	7	4	3	1	2
MRMR	5	4	2	1	3
Relief	2	5	2	3	2
JMI	3	3	2	4	5
CMIM	3	2	2	2	6
SPEC	2	2	0	1	7

5.2 Stability

From the results discussed above, we can conclude that wrapper methods are more effective than filter methods in improving or maintaining predictive performance. Following that, we now delve into analyzing the stability of the FS methods, i.e., how slight changes in the data (e.g., shuffling and sampling) affect the final set of selected features. There are numerous metrics for quantifying the stability of FS methods [18]. However, most of them lack some of the critical properties of a stability metric proposed in [24], which are: *fully defined, strict monotonicity, bounds, maximum stability and correction for chance.* As a stability measure, we use the formula defined in Section 4 by Equation 3 which satisfies these properties.

Table 6: Legend for classifying scalability value.

$\hat{\Phi}$	Stability
<0.40	Poor
0.40 - 0.75	Intermediate to good
>0.75	Excellent

To measure the stability, we first shuffle in a random manner the instances and features of each dataset to obtain ten variants. We perform FS on each one of them and compute the stability metric over the selected features. The stability is interpreted as poor for values less than 0.40, intermediate to good for values in the range $[0.40, 0.75]$ and excellent for values above 0.75 [24] as shown in Table 6.

In Figure 3, we present the stability of the feature subsets from the ten variants of each dataset after applying the FS methods for

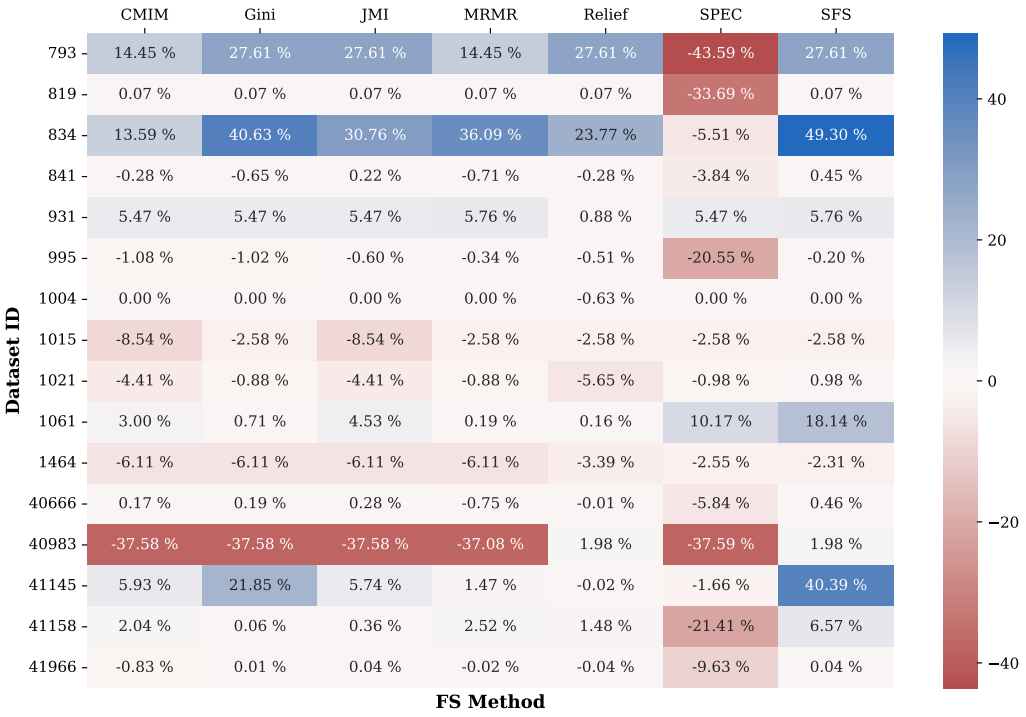


Figure 2: Change in AUC of K-Nearest Neighbour classifier after feature selection.

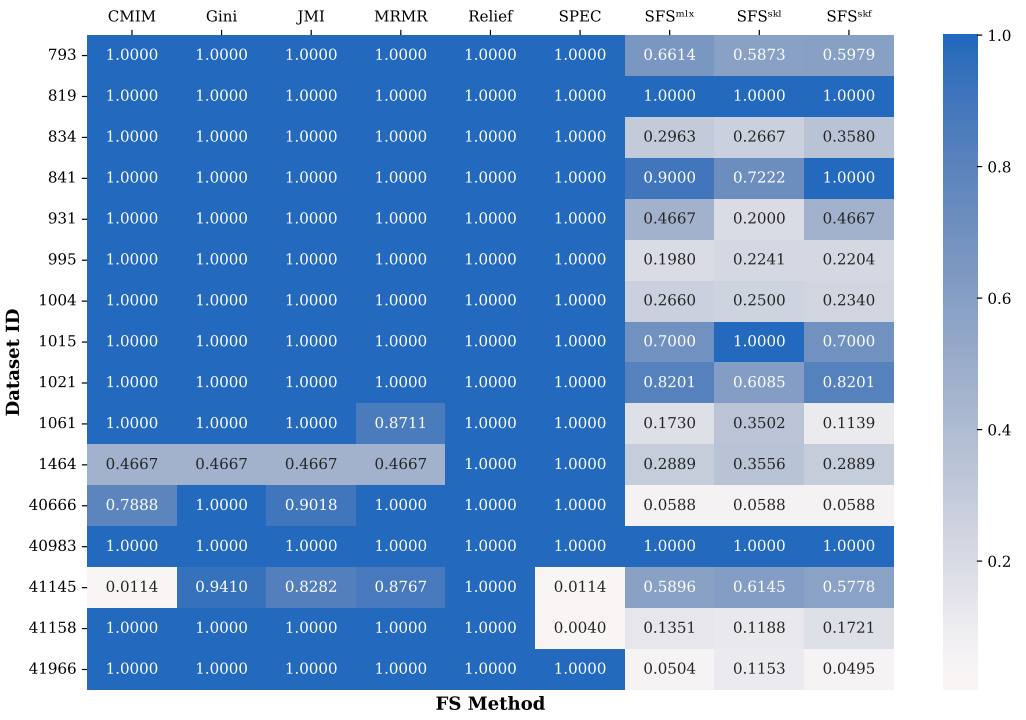


Figure 3: Stability of feature selection methods, using decision tree classifier.

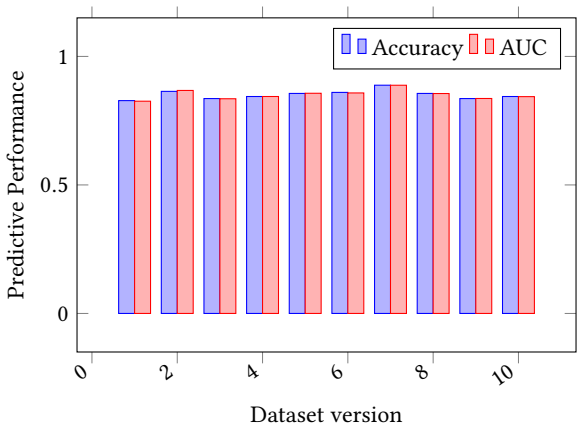


Figure 4: Accuracy and AUC after FS with SFS_{DT}^{MLX} on the ten versions of dataset 793 after shuffling and sampling.

the DT classifier.[13] From this figure, we observe a clear pattern of filter methods being more stable than wrapper methods. The lower stability of wrapper methods is attributed to the cross-validation used in the classifiers to combat model over-fitting. Cross-validation samples the data by splitting it into several folds; this introduces a variance captured in the stability measure.

Considering the dataset with the ID 793 (chosen randomly), using the MRMR filter method, we obtain a stability of 1 which means the same features were selected for all ten variants, thus no change in terms of the final result over the model learned (e.g., accuracy and AUC). On the other hand, the wrapper method SFS_{DT}^{MLX} has a stability of 0.6614, because different subsets are produced from the variants. In Figure 4 we present the accuracy and AUC (which range from 0.83 to 0.89) of the DT classifiers built with the corresponding features from the wrapper method on the ten variants. We see the effect of the changes after the shuffling in Figure 4. Yet, the variance in the accuracy and AUC are quite low (0.03%), and thus negligible. The superior results from wrapper methods (Section 5.1) despite the lower stability further shows that the variance from instability does not directly affect the quality of the selected features.

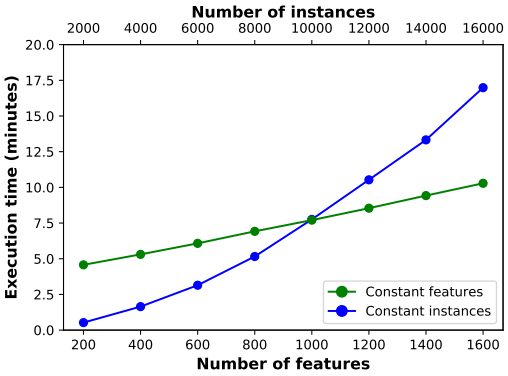
Therefore, in general, data shuffling does not impact the stability of filter methods; however, it destabilizes wrapper methods because of the use of cross-validation in model building. This destabilization, however, does not imply poor predictive performance.

5.3 Scalability and Peak memory

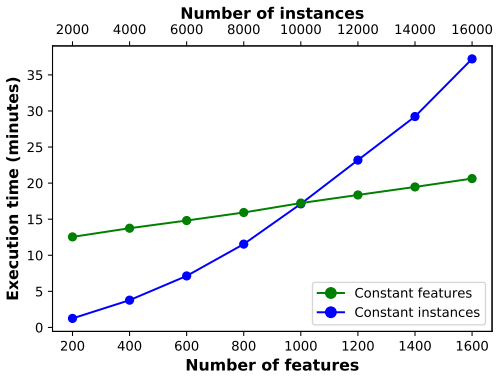
The scalability of FS methods has remained a hot topic especially due to the prevalence of big data [10]. Given that filter methods are indeed more computationally efficient than wrapper methods [32], we focus on analyzing the scalability within wrapper methods implemented in various tools. In particular, we use the DT classifier and a synthetic dataset with 1,600 predictive features, one target feature and 16,000 instances.

In Figures 5 and 6, we show the scalability results obtained after applying wrapper FS methods over the synthetic dataset at various fixed numbers of features and instances. We focus on the execution time and peak memory usage. Particularly, we consider two cases; first where we fixed 10,000 instances and vary the number of features from 200 to 1,600 (blue line) and

[13]For DT, the order of features in the dataset matters. So, shuffling the data causes the wrapper libraries to select different subset of features.

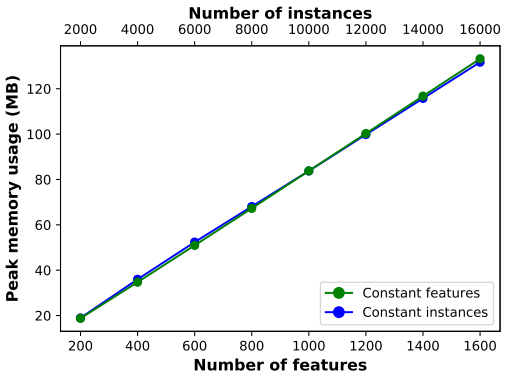


(a) Average execution time of mono-objective methods.

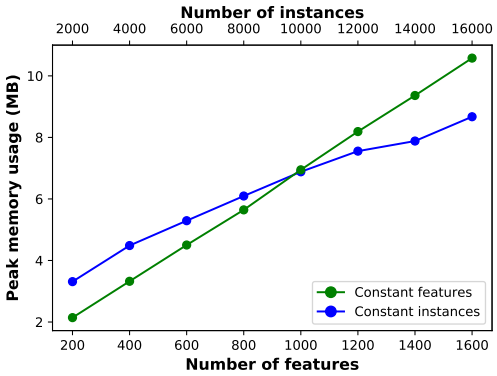


(b) Average execution time of multi-objective method.

Figure 5: Time scalability of wrappers with DT classifier.



(a) Average peak memory of MLX library with DT.



(b) Average peak memory of SKL, SKF, and MOFS.

Figure 6: Memory usage scalability of wrappers with DT.

Table 7: Accuracy and AUC change of MO and single criteria.

Data	Accuracy		AUC	
	MO	AUC	MO	ACC
1015	-3.82%	-2.02%	1.82%	0%
793	0%	-7.87%	-9.85%	-9.85%
1021	0.25%	0%	0.51%	0%
819	0.61%	0%	0.6%	0%
1004	0%	0%	0%	0%
41966	0.33%	-0.5%	0%	-0.04%
995	-0.76%	-2.67%	-0.69%	-1.08%
41158	1.64%	-7.03%	-2.57%	-5.35%
1464	-0.17%	-1.05%	3.74%	-6.18%
931	-7.19%	0%	-1.56%	0%
40983	-0.92%	0%	0.34%	-11.42%
841	-10.37%	-10.51%	0.92%	-11.08%
1061	-1.02%	-3.17%	-2.57%	-8.24%
834	-0.53%	-5.32%	-2.98%	-19.47%
40666	-0.92%	-15.13%	0%	-0.86%
41145	-2.38%	-3.72%	-1.18%	-4.56%

the second is vice versa with 1,000 fixed features and instances varied from 2,000 to 16,000 (green line).

Figures 5a and 5b show the average execution time for mono-objective (MLX, SKL, SFL) and multi-objective wrapper methods respectively. We show the results distinctly because the MOFS takes longer execution time since it simultaneously computes two metrics. Both figures show that a fixed number of features and increasing instances (green line) scale linearly while a fixed number of instances and growing features (blue line) cause a faster increasing computational complexity.

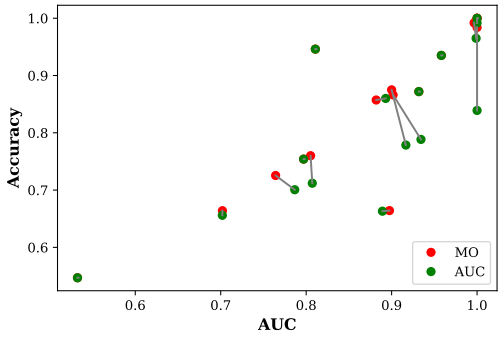
For the peak memory usage, we show the results of the MLX separately in Figure 6a because the library implementation uses significantly more memory than the others shown in Figure 6b. For both cases of fixed features and fixed instances, the figures show that the memory usage scales linearly for wrapper methods; using Decision Trees across all libraries.

From these results, we can conclude that in terms of memory usage these methods are rather efficient. For execution time, they are not scalable in scenarios where we have a large number of instances with the number of features growing rapidly (which is the case in big data). Thus, we contend that the focus should be on developing more scalable wrapper methods in order to benefit more from the positive impact these methods have in general on model performance (Section 5.1).

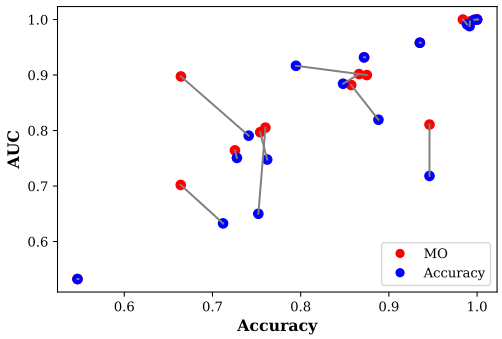
5.4 MOFS Trade-Off

In this section, we compare predictive performance in mono-objective and multi-objective FS. In particular, we consider two mono-objective evaluation criteria (accuracy and AUC) and one multi-objective (MO) criterion formulated by combining accuracy and AUC. In all cases, the search strategy is SFS presented in Algorithm 1. For this comparison, we fix two baselines for each dataset. The first is the accuracy obtained by using accuracy as the evaluation criterion, while the second is the AUC obtained when AUC is used as the evaluation criterion. We argue that these are the baselines for the individual metrics and now measure the deviation as a result of using the other criteria (e.g., how AUC changes when we optimize for accuracy and vice versa).

Beginning with the model accuracy, we use the other criteria (i.e, MO and AUC) to select features from each dataset and measure their accuracy. We then compare this derived accuracy to the accuracy baseline by measuring the percentage of change. A positive number indicates an increase in the accuracy while a negative number indicates a reduction. We present this accuracy change for each dataset in the second and third columns of Table 7. We see that when using an alternative single criterion -AUC, at best the accuracy is maintained (i.e., does not change); otherwise, in most of the cases accuracy is reduced by up to 15%. However, using a multi-objective criteria -MO, we observe a reduction of accuracy to at most 10% and even an increase in accuracy in four datasets. To visualize this, we show the results of using MO and AUC as evaluation criteria in Figure 7a with the accuracy presented in the vertical axis. The position of the red points above the green points represents a higher model accuracy from using MO over AUC as the evaluation metric.



(a) Comparison of accuracy change from MO and AUC criteria.



(b) Comparison of AUC change from MO and Accuracy criteria.

Figure 7: Accuracy and AUC changes per metric.

Next, we examine the AUC when we use the other criteria (i.e, MO and accuracy) to select features for each dataset. Also, we compare the derived AUC to the AUC baseline by measuring the percentage of change. The AUC change for each dataset is also presented, in the fourth and fifth columns of Table 7. We observe that the single accuracy criterion yields no improvement in the AUC, preserves the AUC in only five datasets, and leads to a loss in AUC of up to 19.47%. However, in using a multi-objective criteria -MO, we observe an improvement of the AUC in six datasets while keeping the AUC loss to just 9%. In Figure 7b, we show the results of using MO and accuracy as evaluation criteria. The lines connect the same datasets and the color determines

the optimized metric (accuracy or MO). The position of the red points which represents MO show a higher model AUC than when accuracy is used which are represented by the blue points.

6 RELATED WORK

The ongoing data boom creates an unprecedented need for efficient data processing techniques. In FS, big data threatens the efficacy of existing methods suitable for small and medium-sized datasets. For this reason, extensive research is ongoing around optimizing existing FS techniques and proposing new methods targeted at big data. We discuss the current works according to evaluation criteria, stability, and scalability.

Evaluation criteria. The evaluation of selected features on the target learning algorithm happens after selection for filter methods and during selection for wrapper methods. For filter methods, the evaluation criterion is fixed by definition. However, for wrapper methods, the evaluation criterion is set depending on the goal of the analysis and the ML algorithm used. This choice drives the eventual feature subset. Several criteria, such as accuracy [6], kappa index [35], mean misclassification error (MMCE) [32], AUC [1], and recently, fairness [31] have been used as evaluation criteria for wrapper methods. In some cases, it becomes essential to consider more than one criterion; consequently, custom multi-objective functions are formulated. In [11] and [27], the defined criterion maximizes the classification performance and feature reduction. In [12], the proposed multi-objective function optimizes for the training and validation kappa index to minimize classification error and overfitting respectively. By using NSGA-II [8], a multi-objective evolutionary algorithm, the authors in [15] and [20] proposed multi-objective FS methods. In the former, the goal was minimizing the number of used features and the classification error, while in the latter, the focus was on maximizing the Expected Maximum Profit (EMP) and minimizing the number of features used in a credit scoring model. However, these works either focus on one criterion or add the criterion of minimizing the number of features to a traditional criteria, such as accuracy, AUC, or kappa index.

Recently, the spotlight has been on creating fair ML models with reduced bias towards individuals and groups; FS has been instrumental in achieving this. Using the filter approach, the authors in [19] used information theoretic measures such as mutual, unique, shared, and synergistic information to quantify the accuracy and discriminatory effects of feature sets. They also used Shapley value functions to quantify both objectives for individual features. More commonly, wrapper methods have been used to find this *accuracy - fairness* trade-off. The authors in [31] used NSGA-II to select features that maximized the predictive performance (F1-score) and fairness (Statistical Parity Difference) of the ML models. The average of three metrics (Demographic Parity, Equality of Odds, Equal Opportunities) is combined to measure fairness in [9]. A new utility function is formed from the scalarization of the average fairness metric and F1-score to select features that have good enough predictive performance and are unbiased. Still focusing on fairness and predictive performance, the authors in [33] optimize for these two objectives in a two-phase approach. Firstly using SFS to select a feature set that maximizes predictive performance (F1-score) and next using SBS to eliminate features that reduce fairness (Ratio of Observational Discrimination).

Combining multiple predictive evaluation criteria could also be beneficial for selecting relevant features from a dataset as analysts often explore not just one but several predictive performance metrics to assess model quality. Our experiments combined two predictive evaluation criteria (accuracy and AUC) and demonstrate this advantage.

Stability. Beyond the internal evaluation of FS models through evaluation criteria, the selected features must be reliable and robust, i.e., little changes in the data should not lead to high variability in the selected features. Stability measures the robustness of FS methods to slight changes in the data. The instability of feature subsets from FS remains an issue, especially in domains where the datasets are high dimensional with low sample sizes, such as bioinformatics.

[21] studies the stability of feature rankers (a subset of filter methods) on four microarray data by running 100 Monte Carlo simulations and bootstrapping the original dataset each time to create a new, slightly altered dataset. They demonstrate the sensitivity of FS methods to changes in data perturbation, particularly in microarray data. In [36], the stability of filter and wrapper feature subset selectors were evaluated by randomly sampling a dataset without replacement 30 times for four overlap percentages. The study shows filter methods to be more stable than wrapper methods.

Although, stability is typically measured by data perturbation, cross-validation, or partitioning. The impact of data shuffling (i.e., a re-ordering of features and instances) on the stability of FS methods has yet to be studied. Practically, current tools and libraries for FS need to readily provide this supplementary information on the stability of FS methods necessary to strengthen confidence in the results of the methods.

Scalability. Similar to most ML tasks, big data impacts the time taken to train models and perform FS. Generally, filter methods are more scalable than wrapper and embedded methods. However, the focus has been on making filter methods more scalable using distribution/partitioning [5, 25, 34], parallel processing [18, 25, 26], GPUs [28], MPI[2, 13], and early dropping heuristics [22, 34]. On the other hand, even though the computational cost of wrapper methods worsens with the growing size of data, they have not received much attention when it comes to scalability. Distribution, partitioning [4, 27] and parallel processing [11, 27] have been used to improve some wrapper FS methods. However, as demonstrated in [1] and our experiments (see Section 4), the good performance of wrapper methods motivates the need to propose more scalable versions.

In summary, to the best of our knowledge, there are no works that perform a comparison of the existing tools and libraries for wrapper methods, with respect to their evaluation criteria, stability and scalability. Furthermore, there is no work that assesses the impact of multiple evaluation criteria on the selection of the final set of features and their impact over the performance of the learning algorithm. Hence, in this work, we give a detailed report of the specificities of tools and libraries for wrapper methods as well as demonstrate empirically that multi-objective feature selection of two performance metrics finds a subset of features that offers a better trade-off compared to using only one criterion.

7 CONCLUSIONS AND FUTURE WORK

With FS, the primary aim is to reduce the dimensionality of the data. However, preserving the performance of the model is equally important. We see from using two common model performance metrics –accuracy and AUC, that wrapper FS selects

a subset of features that not only reduces the dimensions but also preserves the model performance better than filter methods. The results of our experiments also demonstrated that multi-objective FS preserves, balances, and in some cases improves model performance when more than one model performance measure is considered.

Additionally, we saw that even though low stability has been identified as one of the challenges of wrapper FS, the variation in model performance was negligible. Consequently, the quality of the features selected is not adversely affected by wrapper methods being less stable than filter methods. Furthermore, the execution time and memory footprint results of the wrapper methods demonstrate that they are memory efficient but not scalable especially with growing number of features.

Our assessments of wrapper FS tools highlight two gaps. First, a lack of tools with population-based search wrapper methods and secondly, the scalability bottleneck with the available tools. With the superior results of wrapper methods, we argue that they deserve more attention to be made more scalable.

As next steps, we plan to extend our study into using population-based search to find Pareto optimal feature subsets selected by combining multiple criteria. We aim to provide a scalable FS tool with a suite of criteria to facilitate holistic feature selection and explainability of ML models.

8 ACKNOWLEDGMENTS

The project leading to this publication has received funding from the European Commission under the European Union's Horizon 2020 research and innovation programme (grant agreement No 955895). Besim Bilalli is partly supported by the Spanish Ministerio de Ciencia e Innovación, as well as the European Union - NextGenerationEU, under the project FJC2021-046606-I/AEI/10.13039/501100011033. Gianluca Bontempi was supported by Service Public de Wallonie Recherche under grant n° 2010235–ARIAC by DIGITALWALLONIA4.AI.

REFERENCES

[1] Farideh Bagherzadeh-Khiabani, Azra Ramezankhani, Fereidoun Azizi, Farzad Hadaegh, Ewout W Steyerberg, and Davood Khalili. 2016. A tutorial on variable selection for clinical prediction models: feature selection methods in data mining could improve the results. *Journal of clinical epidemiology* 71 (2016), 76–85.

[2] Bieito Beceiro, Jorge González-Domínguez, and Juan Touriño. 2022. ParallelFST: a Feature Selection Library for Multicore Clusters. *J. Parallel and Distrib. Comput.* (2022).

[3] Daniel Berrar. 2019. Cross-Validation. In *Encyclopedia of Bioinformatics and Computational Biology*. Academic Press, 542–545.

[4] Verónica Bolón-Canedo, Noelia Sanchez-Marono, and Amparo Alonso-Betanzos. 2013. A distributed wrapper approach for feature selection. In *21st European Symposium on Artificial Neural Networks, Computational Intelligence And Machine Learning*. Citeseer.

[5] Veronica Bolon-Canedo, Konstantinos Sechidis, Noelia Sanchez-Marono, Amparo Alonso-Betanzos, and Gavin Brown. 2019. Insights into distributed feature ranking. *Information Sciences* 496 (2019), 378–398.

[6] Joana Chong, Petra Tjurin, Maisa Niemelä, Timo Jämsä, and Vahid Farrahi. 2021. Machine-learning models for activity class prediction: A comparative study of feature selection and classification algorithms. *Gait & Posture* 89 (2021), 45–53.

[7] Kalyanmoy Deb. 2014. Multi-objective optimization. In *Search methodologies*. Springer, 403–449.

[8] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation* 6, 2 (2002), 182–197.

[9] Ginel Dorleon, Imen Megdiche, Nathalie Bricon-Souf, and Olivier Teste. 2022. Feature selection under fairness constraints. In *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*. 1125–1127.

[10] Naoual El Aboudi and Laila Benhlima. 2016. Review on wrapper feature selection approaches. In *2016 International Conference on Engineering & MIS*. IEEE, 1–5.

[11] Mikel Galar, Isaac Triguero, Humberto Bustince, and Francisco Herrera. 2018. A preliminary study of the feasibility of global evolutionary feature selection for big datasets under Apache Spark. In *2018 IEEE Congress on Evolutionary Computation*. IEEE, 1–8.

[12] Jesús González, Julio Ortega, Miguel Damas, Pedro Martín-Smith, and John Q Gan. 2019. A new multi-objective wrapper method for feature selection–Accuracy and stability analysis for BCI. *Neurocomputing* 333 (2019), 407–418.

[13] Jorge González-Domínguez, Verónica Bolón-Canedo, Borja Freire, and Juan Touriño. 2019. Parallel feature selection for distributed-memory clusters. *Information Sciences* 496 (2019), 399–409.

[14] Isabelle Guyon and André Elisseeff. 2003. An introduction to variable and feature selection. *Journal of machine learning research* 3, Mar (2003), 1157–1182.

[15] Tarek M Hamdani, Jin-Myung Won, Adel M Alimi, and Fakhri Karray. 2007. Multi-objective feature selection with NSGA II. In *International conference on adaptive and natural computing algorithms*. Springer, 240–247.

[16] Steven A Hicks, Inga Strümke, Vajira Thambawita, Malek Hammou, Michael A Riegler, Pål Halvorsen, and Sravanthi Parasa. 2022. On evaluation metrics for medical applications of artificial intelligence. *Scientific Reports* 12, 1 (2022), 1–9.

[17] Raj Jain. 1990. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. John Wiley & Sons.

[18] Utkarsh Mahadeo Khaire and R Dhanalakshmi. 2019. Stability of feature selection algorithm: A review. *Journal of King Saud University-Computer and Information Sciences* (2019).

[19] Sajad Khodadadian, Mohamed Nafea, AmirEmad Ghassami, and Negar Kiyavash. 2021. Information theoretic measures for fairness-aware feature selection. *arXiv preprint arXiv:2106.00772* (2021).

[20] Nikita Kozodoi, Stefan Lessmann, Konstantinos Papakonstantinou, Yiannis Gatsoulis, and Bart Baesens. 2019. A multi-objective approach for profit-driven feature selection in credit scoring. *Decision support systems* 120 (2019), 106–117.

[21] Panagiotis Moulos, Ioannis Kanaris, and Gianluca Bontempi. 2013. Stability of feature selection algorithms for classification in high-throughput genomics datasets. In *13th IEEE International Conference on BioInformatics and BioEngineering*. IEEE, 1–4.

[22] Thu Nguyen, Nhan Phan, Nhuong Nguyen, Binh T Nguyen, Pål Halvorsen, and Michael A Riegler. 2022. Parallel feature selection based on the trace ratio criterion. In *2022 International Joint Conference on Neural Networks*. IEEE, 1–8.

[23] Uchechukwu Njoku, Alberto Abelló, Besim Bilalli, and Gianluca Bontempi. 2022. Impact of Filter Feature Selection on Classification: An Empirical Study.. In *24th International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data*. 71–80.

[24] Sarah Nogueira, Konstantinos Sechidis, and Gavin Brown. 2017. On the stability of feature selection algorithms. *J. Mach. Learn. Res.* 18, 1 (2017), 6345–6398.

[25] Raul-Jose Palma-Mendoza, Luis de Marcos, Daniel Rodriguez, and Amparo Alonso-Betanzos. 2019. Distributed correlation-based feature selection in spark. *Information Sciences* 496 (2019), 287–299.

[26] Raul-Jose Palma-Mendoza, Daniel Rodriguez, and Luis De-Marcos. 2018. Distributed ReliefF-based feature selection in Spark. *Knowledge and Information Systems* 57, 1 (2018), 1–20.

[27] Daniel Peralta, Sara del Río, Sergio Ramírez-Gallego, Isaac Triguero, Jose M Benitez, and Francisco Herrera. 2015. Evolutionary feature selection for big data classification: A mapreduce approach. *Mathematical Problems in Engineering* 2015 (2015).

[28] Sergio Ramírez-Gallego, Iago Lastra, David Martínez-Rego, Verónica Bolón-Canedo, José Manuel Benítez, Francisco Herrera, and Amparo Alonso-Betanzos. 2017. Fast-mRMR: Fast minimum redundancy maximum relevance algorithm for high-dimensional big data. *International Journal of Intelligent Systems* 32, 2 (2017), 134–152.

[29] Sebastian Raschka, Joshua Patterson, and Corey Nolet. 2020. Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence. *Information* 11, 4 (2020), 193.

[30] Claudio Reggiani, Yann-Aël Le Borgne, and Gianluca Bontempi. 2017. Feature selection in high-dimensional dataset using MapReduce. In *Benelux Conference on Artificial Intelligence*. Springer, 101–115.

[31] Ayaz Ur Rehman, Anas Nadeem, and Muhammad Zubair Malik. 2022. Fair Feature Subset Selection using Multiobjective Genetic Algorithm. *arXiv preprint arXiv:2205.01512* (2022).

[32] Victor F Rodriguez-Galiano, Juan Antonio Luque-Espinar, M Chica-Olmo, and María Paula Mendes. 2018. Feature selection approaches for predictive modelling of groundwater nitrate pollution: An evaluation of filters, embedded and wrapper methods. *Science of the total environment* 624 (2018), 661–672.

[33] Ricardo Salazar, Felix Neutatz, and Ziawasch Abedjan. 2021. Automated feature engineering for algorithmic fairness. *Proceedings of the VLDB Endowment* 14, 9 (2021), 1694–1702.

[34] Ioannis Tsamardinos, Giorgos Borboudakis, Pavlos Katsogridakis, Polyvios Pratikakis, and Vassilis Christophides. 2019. A greedy feature selection algorithm for Big Data of high dimensionality. *Machine learning* 108, 2 (2019), 149–202.

[35] Susana M Vieira, Uzay Kaymak, and João MC Sousa. 2010. Cohen's kappa coefficient as a performance measure for feature selection. In *International*

conference on fuzzy systems. IEEE, 1–8.

[36] Randall Wald, Taghi M Khoshgoftaar, and Amri Napolitano. 2013. Stability of filter-and wrapper-based feature subset selection. In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence*. IEEE, 374–380.