A framework for feature selection through boosting

Alsahaf, Ahmad; Petkov, Nicolai; Shenoy, Vikram; Azzopardi, George

Published in:
Expert systems with applications

DOI:
10.1016/j.eswa.2021.115895

Expert Systems With Applications 187 (2022) 115895

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

A framework for feature selection through boosting

Ahmad Alsahaf [a,*], Nicolai Petkov [a], Vikram Shenoy [b], George Azzopardi [a]

[a] Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence, University of Groningen, P.O. Box 407, 9700 AK Groningen, The Netherlands
[b] Khoury College of Computer Sciences, Northeastern University, West Village Residence Complex H, 440 Huntington Avenue 202, Boston, MA 02115, United States

ARTICLE INFO

Keywords:
Feature selection
Boosting
Ensemble learning
XGBoost

ABSTRACT

As dimensions of datasets in predictive modelling continue to grow, feature selection becomes increasingly practical. Datasets with complex feature interactions and high levels of redundancy still present a challenge to existing feature selection methods. We propose a novel framework for feature selection that relies on boosting, or sample re-weighting, to select sets of informative features in classification problems. The method uses as its basis the feature rankings derived from fast and scalable tree-boosting models, such as XGBoost. We compare the proposed method to standard feature selection algorithms on 9 benchmark datasets. We show that the proposed approach reaches higher accuracies with fewer features on most of the tested datasets, and that the selected features have lower redundancy.

1. Introduction

The presence of irrelevant and redundant features in a dataset can lower the performance of predictive models, due to over-fitting and the curse of dimensionality. Moreover, even if a model's performance is robust to redundancy and noise, the presence of those features has other disadvantages, like increasing storage and computational costs, and limiting the model's interpretability. Feature selection can mitigate these problems by identifying and selecting relevant features, and removing irrelevant and redundant ones.

Model interpretability has become specially pertinent with the rise of interest in explainable AI (Holzinger, 2018; Gunning, 2017). The interprertability of machine learning models is important as it enables compliance with the socially relevant requirements of those models, such as fairness, unbiasedness, privacy, trust, and reliability (Doshi-Velez & Kim, 2017).

Since modern machine learning applications contain an ever growing number of features, interpretability and intuitive understanding of prediction outcomes has become virtually impossible before some form of dimensionality reduction. Other aides of model interpretation, like data visualization, are also made easier by reducing the number of features.

Furthermore, in certain biomedical applications, knowledge discovery is the primary task of feature selection, more so than improving the prediction outcome or increasing computational efficiency (Borboudakis & Tsamardinos, 2019). In gene expression studies, feature selection is used to discover the genetic networks associated with diseases (Tabus & Astola, 2005). In other biomarker discovery studies, the workflow relies heavily on feature selection, as the studies often begin - for practical limitation - with large feature, small sample raw data (Christin et al., 2013).

Another benefit of feature selection is found in applications where the acquisition of features is costly. In such cases, it is useful to identify if a costly feature happens to be irrelevant or redundant (Early, Fienberg, & Mankoff, 2016; Bolón-Canedo & Alonso-Betanzos, 2019). For example, in medical data, a feature could be associated with a clinical test, which could either be expensive, inconvenient to patients, or both.

Reducing the number of features in any dataset can be achieved with principally different approaches. Therefore, many methods of feature selection have been proposed in literature. The most popular taxonomy of these methods divides them into three broad categories: filter methods, wrapper methods, and embedded methods (Guyon & Elisseeff, 2003).

This categorization pertains to how the selection process and the associated prediction task are connected. Filter methods rank features independently of any prediction model, whereas wrapper methods evaluate the performance of candidate feature subsets on a pre-chosen predictor. Finally, embedded methods reduce the number of features in conjunction with solving a prediction problem (Guyon & Elisseeff, 2003; Dash & Liu, 1997; Tang, Alelyani, & Liu, 2014; Kumar & Minz, 2014). Other taxonomies may include additional application-specific categories, such as methods for structured or streaming features

* Corresponding author.
E-mail addresses: a.m.j.a.alsahaf@rug.nl (A. Alsahaf), n.petkov@rug.nl (N. Petkov), shenoy.vi@northeastern.edu (V. Shenoy), g.azzopardi@rug.nl (G. Azzopardi).

https://doi.org/10.1016/j.eswa.2021.115895
Received 19 February 2021; Received in revised form 20 June 2021; Accepted 7 September 2021
Available online 16 September 2021

(Jović, Brkić, & Bogunović, 2015; AlNuaimi, Masud, Serhani, & Zaki, 2020), and class-specific feature selection methods (Pineda-Bautista, Carrasco-Ochoa, & Martinez-Trinidad, 2011; Gao, Hu, & Zhang, 2018; Nardone, Ciaramella, & Staiano, 2019).

Compared to filters, wrapper methods select subsets with high accuracy at the expense of increased computational cost. Since an exhaustive search of all possible feature subsets is intractable for large datasets, wrapper methods make use of efficient search strategies for finding candidate subsets to evaluate with the wrapped predictor (Guyon & Elisseeff, 2003; Tang et al., 2014; El Aboudi & Benhlima, 2016). Examples of these strategies include hill-climbing, best-first, genetic algorithms (Kohavi & John, 1997), particle swarm optimization (Ibrahim, Ewees, Oliva, Abd Elaziz, & Lu, 2019), and Whale optimization (Mafarja & Mirjalili, 2018).

Boosting, a popular meta-algorithm in ensemble classification, can also be used to modify the feature search space in wrapper-based feature selection. This use of boosting, or sample re-weighting, has been explored to limited capacity in past studies (Das, 2001; Tieu & Viola, 2004; Liu, Liu, & Zhang, 2009; Barddal, Enembreck, Gomes, Bifet, & Pfahringer, 2019). In this paper, we expand on this concept and propose a greedy forward selection algorithm that we call FeatBoost. The algorithm uses embedded feature importance scores of tree ensemble models for choosing the candidate features, then uses boosting to update the importance scores, and thus the search space, after every iteration.

Compared to past studies, we contribute the following: (1) a sample weighting strategy which weights each sample according to its prediction probability. In contrast, previous methods up-weight all misclassified samples by the same amount, (2) a modular algorithm architecture, which decouples feature ranking from selection. This overcomes inconsistencies in feature rankings, and potentially increases the robustness of the selected subsets, (3) a sample weighting reset strategy, which prevents premature stopping of the algorithm.

2. Article structure

In Section 4, we describe the novel elements of the proposed approach in relation to existing algorithms, and give an overview of recent, relevant developments in feature selection with tree ensembles. We then give the details of the proposed algorithm in Section 5. Then, we give the experimental settings in Section 6, followed by the results in Section 7. Finally, we discuss the implications of the results and future developments in Section 8.

3. Notations and definitions

In this section, we introduce the notations used in the rest of the paper. Whenever possible, we unify notations describing the different methods we compare. Therefore, the notations do not necessarily reflect those used in the original works.

Each instance of feature selection is solved on a given dataset $\mathscr{D}$ with p features, n samples, and a discrete target output y with n_c classes. A prediction of y is referred to as $\hat{y}$. The subset of selected features for the given output is denoted by $\mathscr{X}$.

In methods that take as input the number of desired features to select, or the maximum number of features that can be selected, this number is denoted as p'. The actual number of features selected is denoted as p^*. If a method selects features sequentially, or produces a rank for the features in $\mathscr{X}$, then $\mathscr{X}^i$ refers to $\mathscr{X}$ at the ith iteration, for $i = 1, \ldots, p^*$.

For clarity, we stress here the difference between two types of boosting that are present in the proposed algorithm. The first occurs within the ensemble model that is used to generate the feature importance rankings - if a boosting method was used for that purpose - while the second is an outer layer of boosting (or sample re-weighting) performed specifically for the feature selection process, and bears no direct functional relation to the first type. From this point onward, mentions of sample re-weighting or boosting in this paper refer to the second type, unless otherwise specified.

4. Background and related work

In this section, we elaborate on the use of boosting in feature selection by highlighting a number of existing methods, and how the proposed algorithm differs from them. We also consider more broadly methods that use the embedded feature importance scores of decision-tree models as bases for feature selection.

4.1. Tree-based feature importance

The embedded feature importance scores of tree-based ensembles are powerful starting points for feature selection (Tuv, Borisov, Runger, & Torkkola, 2009). This is largely due to the following factors: First, the versatility of those models; being scale invariant, scalable to large datasets, and able to handle numerical, categorical, and missing data. Second, the fact that the intrinsic importance scores can be derived at no additional cost over that of model training.

In a decision tree, the importance of a feature is defined as the total value of a node-splitting criterion that the feature is responsible for (e.g. Information Gain or the Gini Index). When used in ensembles, it is commonly defined as the sum or average of the splitting criterion that a feature causes across all trees (Breiman, 2002; Louppe, Wehenkel, Sutera, & Geurts, 2013).

In an ensemble of trees, like random forest (Breiman, 2001), the importance scores of two or more redundant features will be spread evenly among them, due to feature sub-sampling and bootstrapping. Using those scores as a basis for feature selection in large datasets could lead to falsely selecting a feature with many of its redundant copies (Genuer, Poggi, & Tuleau-Malot, 2010).

Other issues with tree-derived feature scores include the bias towards categorical features of high cardinality (Strobl, Boulesteix, Zeileis, & Hothorn, 2007), sensitivity to hyper-parameters (Genuer et al., 2010), and inconsistencies (Lundberg, Erion, & Lee, 2018). The latter refer to cases where the assigned importance of a feature decreases when its true impact on model performance has increased.

For the importance scores of tree models to be used reliably for feature selection, these shortcomings should be overcome. This can be partially achieved through boosting, or sample re-weighting.

Boosting algorithms, like AdaBoost and its derivatives (Freund & Schapire, 1997), rely on a sequential procedure of varying the sample weights of weak classifiers, typically decision-trees, based on the accuracy of previous boosting rounds. This leads the classifiers in later rounds of the algorithm to focus on samples that were misclassified in earlier rounds. Then, each classifier votes to determine the final outcome of the ensemble.

Through sample re-weighting, the process of boosting also affects the feature importance scores (and rankings) produced by the classifier being boosted. This presumably happens in such a way that in later boosting rounds, some features which initially ranked poorly, appear in top ranked positions due to becoming effective in classifying samples which were misclassified in earlier rounds. We explain this further in the following section.

4.2. Boosting and feature selection

The effect of boosting on feature importance scores has been exploited in the past to design feature selection algorithms (Das, 2001; Tieu & Viola, 2004; Liu et al., 2009). In short, this is done by relying on a weight-sensitive feature ranking algorithm, and a sequential procedure of adding features, often one at a time. At each round, the ranking algorithm is applied to a re-weighted version of the training data. And the re-weighting is done based on the classification error produced by features selected so far. The best feature from each round, according to the

feature ranking, is added to the selected subset.

Early examples of directly using boosting for feature selection include the following: Boosted Decision Stump Feature Selection (BDSFS) (Das, 2001), Boosting Image Retrieval (Tieu & Viola, 2004), and Boosted Mutual Information Feature Selection (BMIFS) (Liu et al., 2009). A recent example is Adaptive Boosting for Feature Selection (ABFS) (Barddal et al., 2019).

Das (2001) used a tree stump as a base classifier, and followed the sample re-weighting strategy from AdaBoost for the purpose of feature selection. Namely, all training samples are initially given a weight of $\frac{1}{n}$, with n being the number of training samples. Then, at each subsequent iteration, the sample weights are given as a function of the classification error from the previous iteration. All misclassified samples are equally up-weighted according to Eq. (1). The best feature from every iteration, according to Information Gain, is selected.

$$\begin{aligned} \alpha &= \log\frac{1-err}{err} \\ \omega_j^{i+1} &= \omega_j^i \cdot \exp(\alpha);\ \forall j = 1,\cdots,n \\ \omega_j^{i+1} &= \frac{\omega_j^{i+1}}{\sum_{j=1}^{n}\omega_j^{i+1}};\ \forall j = 1,\cdots,n \end{aligned} \tag{1}$$

where err is the classification error from the previous iteration and ω_j^i is the sample weight for sample j at the ith iteration.

Boosted Image Retrieval follows a similar procedure to BDSFS, adapted for the purpose of image retrieval. The BMIFS method differs in that the error used for updating sample weights is estimated from an information metric, Mutual Information, and not from a base classifier. Barddal et al. (2019) use this form of feature selection to solve the problem of feature drift in data streams.

Our approach follows a framework that is similar to the aforementioned methods, but differs from them in a number of ways. First, we use a different sample re-weighting strategy. The AdaBoost weighting strategy, used in BDSFS (Eq. 1), re-weights all misclassified samples by the same amount, which disregards how far a sample is from being correctly predicted. To improve this, we weight each sample inversely proportional to its prediction probability, according to Eq. 2.

$$\begin{aligned} \alpha^{\mathbf{i}} &= -\sum_{c=1}^{n_c}\mathbf{Y_c}\log(\mathbf{P_c}) \\ \alpha_j^i &= \alpha_j^i \big/ \alpha_j^{i-1};\ \forall j = 1,\cdots,n \\ \omega_j^{i+1} &\leftarrow \omega_j^i \cdot \alpha_j^i;\ \forall j = 1,\cdots,n \\ \omega_j^{i+1} &= \frac{\omega_j^{i+1}}{\sum_{j=1}^{n}\omega_j^{i+1}}\ \forall j = 1,\cdots,n \end{aligned} \tag{2}$$

where $\mathbf{Y_c}$ is a one-hot encoded matrix indicating the correct class for each sample, $\mathbf{P_c}$ is an $n \times n_c$ matrix containing the class probabilities for each sample, obtained from a classifier, and ω_j^i is the sample weight for sample j at the ith iteration.

For a given sample, the associated weighting term α_j is decreased as the probability of the correct class for that sample approaches 1, and increased as it approaches zero. Therefore, samples which are far from being correctly classified are given higher weights in the next iteration.

We use a gradient boosting trees model, XGBoost, as a base learner for obtaining the feature scores (Chen & Guestrin, 2016). Despite being computationally demanding compared to individual trees, ensembles of trees are more predictive in large datasets, and their feature importance scores reflect more complex interactions. XGBoost is a powerful example of such models, and it outperforms traditional tree-ensemble models in many applications (Luckner, Topolski, & Mazurek, 2017; Alsahaf, Azzopardi, Ducro, Veerkamp, & Petkov, 2018; Murauer & Specht, 2018).

Moreover, we introduce a few procedural changes, some of which were inspired by Iterative Input Selection (IIS) (Galelli & Castelletti, 2013), a tree-based forward selection method for regression problems.

For instance, in FeatBoost, we use a two-step process to select the best feature at each iteration. First, the top ranked features are obtained from the embedded feature scores of a tree model trained on all features. Then, we use a classifier to evaluate the classification performance of the top-ranked features obtained from the embedded scores. This strategy is used in IIS, where evaluations of a regressor determine the best feature at each iteration (Galelli & Castelletti, 2013). We use this approach of model evaluation in FeatBoost by testing m of the top ranked features at each iteration. However, instead of evaluating each of the candidate features individually as a single-input model, we append each feature to the selected features thus far, and evaluate the classification accuracy of the resulting models. Namely, at the ith iteration, we evaluate m models of order i. With this approach, the algorithm could be viewed as a step-wise greedy search in which the search space in each iteration is reduced from p features, to a user-specified number of features. And those candidate features change through the process of boosting.

The justification for this two-step process is the following: First, choosing the top feature from the feature ranking may not be reliable in the presence of feature redundancy in large datasets. Second, using model evaluations decouples the selection from the feature ranking algorithm, making it more robust (Galelli & Castelletti, 2013). Moreover, this process solves the issue of inconsistency highlighted by Lundberg et al. (2018).

We make FeatBoost modular by allowing the model choice for the evaluation step to be different than the model used for feature ranking. The decoupling of the two procedures - ranking and model evaluation - could lead to improvements in computational efficiency. This could be achieved by choosing the second model to be a computationally efficient one, as opposed to the first model, which produces the feature rankings.

Another element of IIS which we use in FeatBoost is that once a feature is selected, it is not dropped from the list of candidate features of subsequent iterations. This way, a feature may be selected twice, which translates to an automatic stopping condition for the algorithm.

The motivation for using this strategy is that future iterations on re-weighted samples will rank new features in the presence of all other features. This means that if a feature ranks higher as a result of sample re-weighting, it does so in interaction with features that were selected before, and those that might be selected after. And the only difference between rankings of different iterations comes from sample re-weighting, and not from explicitly removing features from the list of candidates once they have been selected.

We make further adjustments to the boosting process to make it more adapted to feature selection. When boosting for the purpose of classification, as in AdaBoost, it is not necessary that each classifier is trained on a highly different sample distribution. In other words, if sample weights do not change significantly after a boosting round, this will not necessarily hinder performance, as the final classification will be determined by a majority vote of all classifiers.

On the other hand, in a feature selection context like the proposed method, a relevant feature is added at each boosting round. Therefore, classification error is expected to decrease with rounds. Consequently, the difference in α (Eqs. (1) and (2)) between consecutive rounds will decrease. If the difference becomes low enough, sample weights swill stop changing, and therefore the desired variation in the top ranked features will stop or diminish, causing the algorithm to prematurely terminate.

We solve this problem with two strategies. First, we base the re-weighting of samples not on the performance of the base classifier of the current iteration, but on the relative performance between the current iteration and the preceding one, hence the normalization step of α in the second line of Eq. (2). Second, we use the following *reset* scheme: At any given iteration, if an existing feature is selected again, or the

Algorithm 1. Pseudocode of FeatBoost.

input : Dataset $\mathcal{D}$ with p features, n samples, and output $\boldsymbol{y}$ with n_c classes.
output: $\mathcal{X}^i$ (a subset of selected features at the ith iteration).
initialize: $i \leftarrow 1$, $\mathcal{X}^0 \leftarrow \phi$, reset $\leftarrow 0$, and sample weights $\omega_j^0 \leftarrow \frac{1}{n} \forall j = 1, \cdots, n$;
choose: stopping conditions $\{\epsilon, p'\}$, parameters m, k, and classifiers H_1 and H_2.
while $i < p'$ **and** $reset \leq 1$ **do**
  fit H_1 to $\mathcal{D}$ with ω^i and rank all features;
  fit H_2 to $\{\mathcal{X}^{i-1}, x\}$ for all x in the top m ranked features;
  find the top performing feature x^i, in cross-validation;
  compute $\Delta Acc = Acc[H_2(\mathcal{X}^{i-1}, x^i)] - Acc[H_2(\mathcal{X}^{i-1})]$, in cross-validation;
  if $\Delta Acc > \epsilon$ **and** $x^i \notin \mathcal{X}^i$ **then**
    set $\mathcal{X}^i \leftarrow \{\mathcal{X}^{i-1}, x^i\}$;
    fit H_1 to $\mathcal{X}^i$ to find the class probabilities, $\mathbf{P_c}$, and compute $\alpha^{\mathbf{i}}$: $\alpha^{\mathbf{i}} = -\sum_{c=1}^{n_c} \mathbf{Y_c} \log(\mathbf{P_c})$;
    re-normalize $\alpha_j^i = \alpha_j^i / \alpha_j^{i-1}$; $\forall j = 1, \cdots, n$;
    update $\omega_j^{i+1} \leftarrow \omega_j^i \cdot \alpha_j^i$; $\forall j = 1, \cdots, n$;
    re-normalize $\omega_j^{i+1} = \frac{\omega_j^{i+1}}{\sum_{j=1}^{n} \omega_j^{i+1}}$; $\forall j = 1, \cdots, n$;
    reset $\leftarrow 0$;
    i += 1
  else
    re-initialize weights $\omega_j^i \leftarrow \frac{1}{n} \forall j = 1, \cdots, n$;
    reset += 1
  end
end

selected feature causes no increase in performance, we re-initialize the samples to have equal weights, and repeat the iteration with the new weights. In effect, this reboots the algorithm with a non-empty feature set, which could allow for the selection of additional useful features.

4.3. Ensemble tree models and feature selection

The simplest way to use the embedded scores of ensemble tree models for feature selection is by thresholding, or by only retaining features with non-zero scores. An alternative approach is to use a simple forward selection procedure; relying on the feature rankings to introduce one feature at a time to the selected subset, if the feature causes a significant gain in performance (Genuer et al., 2010). This approach will suffer from the various inconsistencies and biases of those scores (see Section 1).

More elaborate ways of using these scores have been proposed. One such example is to select features according to their importance scores when compared to artificial features, designed to trick the ranking algorithm (Kursa et al., 2010; Tuv et al., 2009). A popular example of this approach is the Boruta method (Kursa et al., 2010). It works by creating shadow features, which are copies of the original ones whose values are shuffled across samples, then computing the feature importance scores of the original set plus shadow features using a random forest classifier. Original features that score lower than the top-scoring shadow features are deemed irrelevant, and are subsequently removed. The process is repeated until all the remaining original features are relevant. This method, by design, does not solve issues of redundancy, as it selects all relevant features (Kursa et al., 2010).

Tree-based models can also be used in combination with other approaches to improve their feature selection capabilities. Rao et al. (2019) combine gradient boosted trees with artificial bee colony algorithms for feature selection. Peker, Arslan, Şen, Çelebi, and But (2015) combine the scores from random forest with the filter method ReliefF for selecting feature extracted from EEG signals.

Other methods attempt to solve the issues with tree-based feature scores not through external procedures, but by modifying, or redefining the importance scores themselves to address particular weaknesses (Lundberg et al., 2018; Nguyen, Huang, & Nguyen, 2015; Strobl et al., 2007; Strobl, Boulesteix, Kneib, Augustin, & Zeileis, 2008).

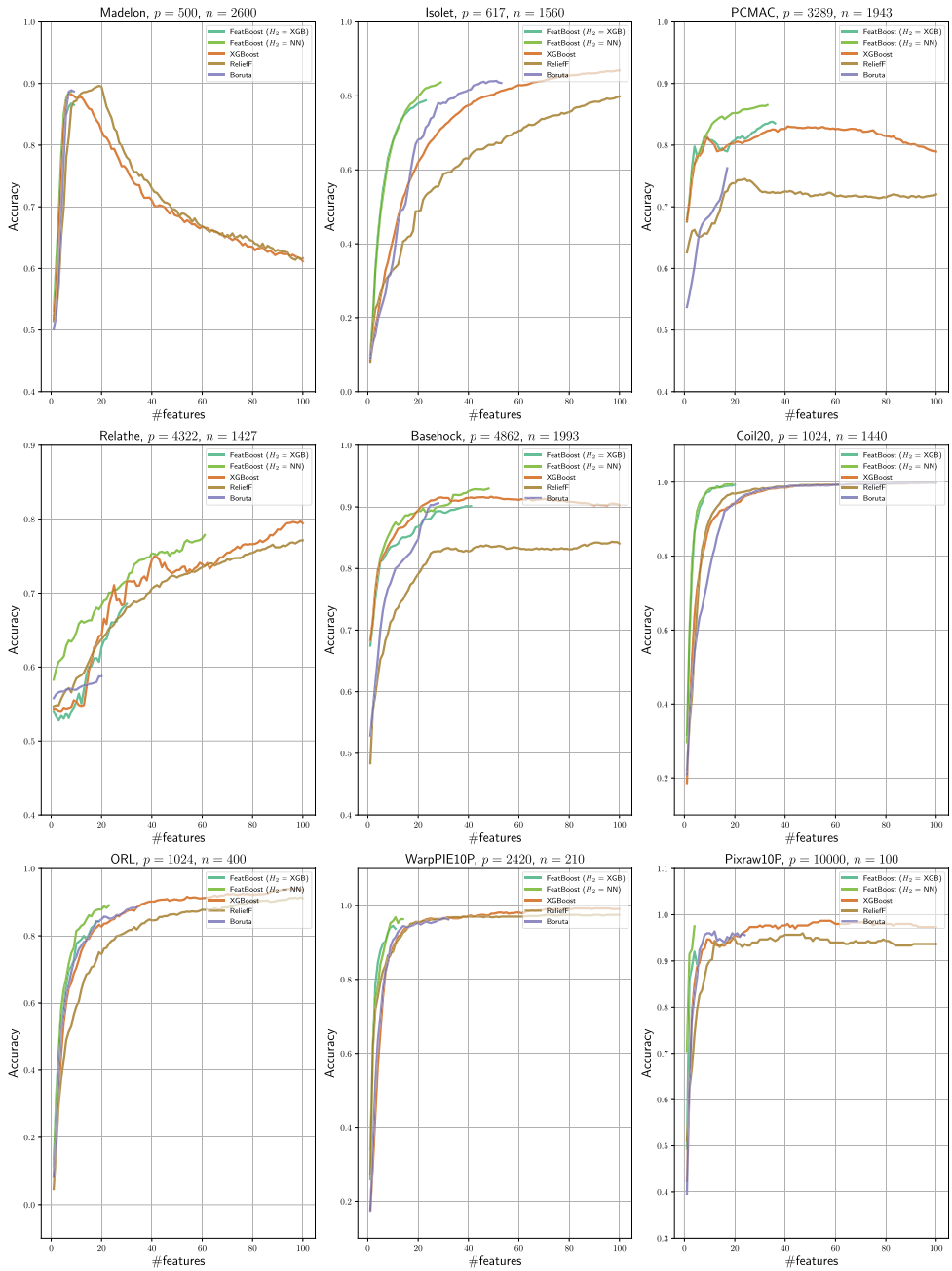

Fig. 1. Accuracies of the subsets selected by each feature selection algorithm. The validation classifier used to compute the accuracy is Nearest Neighbor.

5. Methodology

For a given dataset $\mathscr{D}$ with p features, n training samples, and output y with n_c classes, the FeatBoost algorithm proceeds as follows: First, the selected subset of features, $\mathscr{X}$, is initialized to empty, and a user selected tree-based classifier, H_1, is trained on all samples with initial weights equal to $\frac{1}{n}$, to produce a ranking of all features. We choose H_1 to be an XGBoost classifier for the remainder of the paper.

Then, a user-specified number, m, of the top ranking features are evaluated and compared as single-input classifiers in k-fold cross validation, using either H_1, or a different classifier, H_2.[1] The best performing feature with H_2, according to an appropriate metric (e.g. classification accuracy, F-score, or area under the ROC curve) is added to the selected subset $\mathscr{X}$. In iterations other than the first, classifier H_2 is used to evaluate each of the m features appended to the features selected so far, $\mathscr{X}^{i-1}$.

Finally, H_1 is trained on all selected features, and its prediction probabilities are used to update the sample weights for the following iteration according to Eq. (2).

If the increase in accuracy of $\mathscr{X}$ with respect to the previous iteration is below a user-defined threshold ϵ, or if a feature is selected twice, the algorithm is temporarily paused, and a sample weight reset scheme is initiated.

The reset scheme functions as follows: If at iteration i, a feature is selected which already belongs to $\mathscr{X}^{i-1}$, or if the feature does not improve classification performance, the algorithm normally terminates,

[1] Note that H_2 does not need to be tree-based, nor sensitive to sample weights, since it is not used in the feature ranking process.

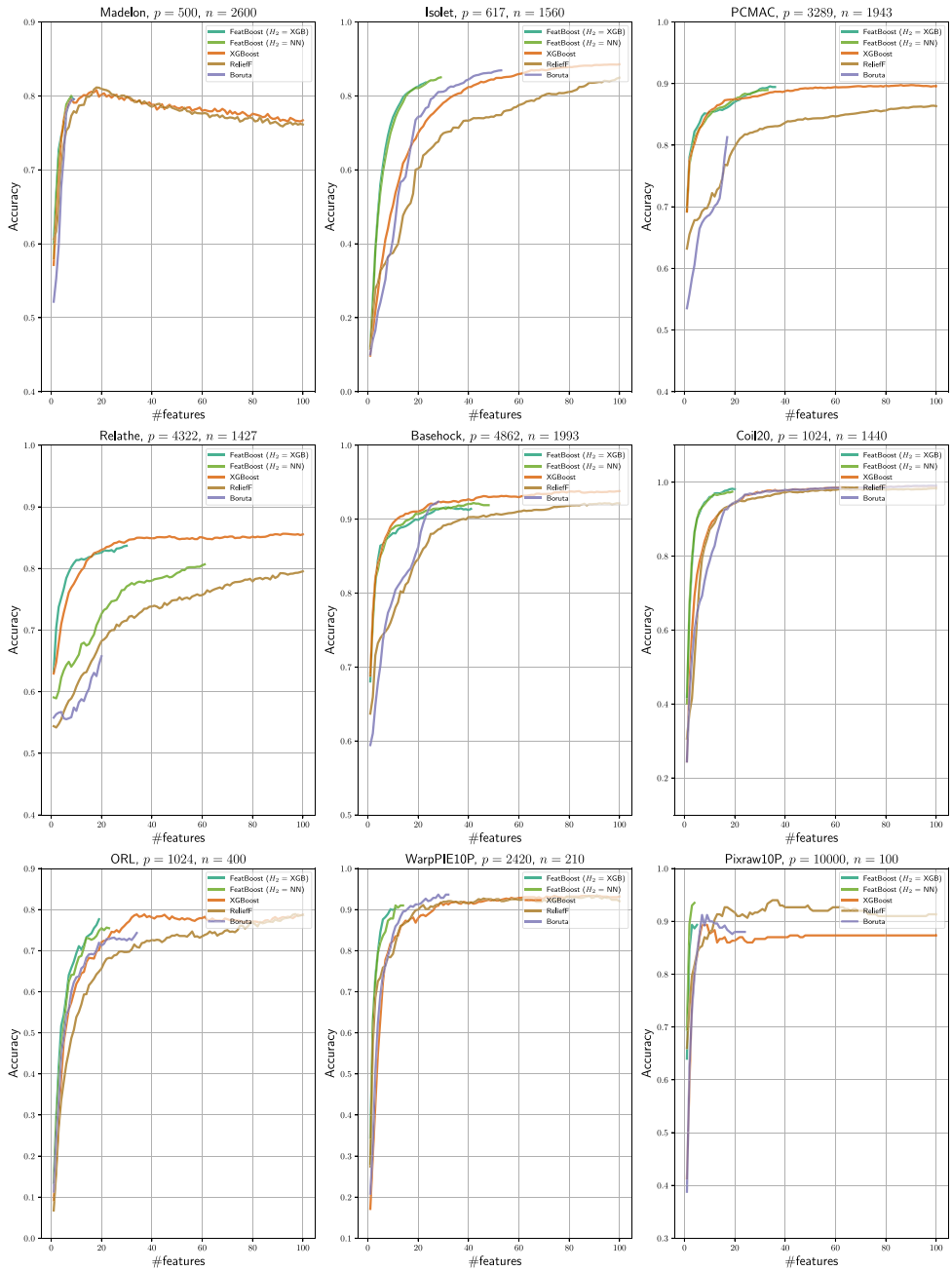

Fig. A.2. Accuracies of the subsets selected by each feature selection algorithm. The validation classifier used to compute the accuracy is XGBoost.

and $\mathscr{X}^{i-1}$ is taken as the final subset. Under the reset scheme, this is temporarily overcome by resetting the sample weights to their initial values, and repeating iteration i. This will lead to the feature ranking being equal to that of the first iteration, albeit with an initial $\mathscr{X}$ that is not empty. This could lead the model evaluation step (with H_2) to select a feature that is partially redundant to one or more features in $\mathscr{X}^{i-1}$, but nonetheless having additional predictive value. If that occurs, the algorithm resumes its normal course from iteration i until a stopping condition is reached again: a feature is selected twice, or the selected feature does not improve accuracy. No further resets are initiated at that point. If the reset scheme does not lead to selecting a new useful feature, the algorithm stops.[2]

The asymptotic time complexity of the FeatBoost, computed in terms of its parameters is given in Eq. (3).

$$O(p' \cdot (O(H_1) + m \cdot k \cdot + p \cdot O(H_2) \log p)) \tag{3}$$

where $O(H_1)$ and $O(H_2)$ are the asymptotic complexities of the chosen classifiers. For reference, the time complexity of training each tree in XGBoost is $O(n \log n)$ (Chen & Guestrin, 2016). The complexity of FeatBoost, therefore, depends highly on the choices of H_1 and H_2, and to a lesser extent on the other parameters choices.

The pseudocode of the algorithm, along with the details of weighting

[2] A software implementation of the algorithm is available at https://github.com/amjams/FeatBoost.

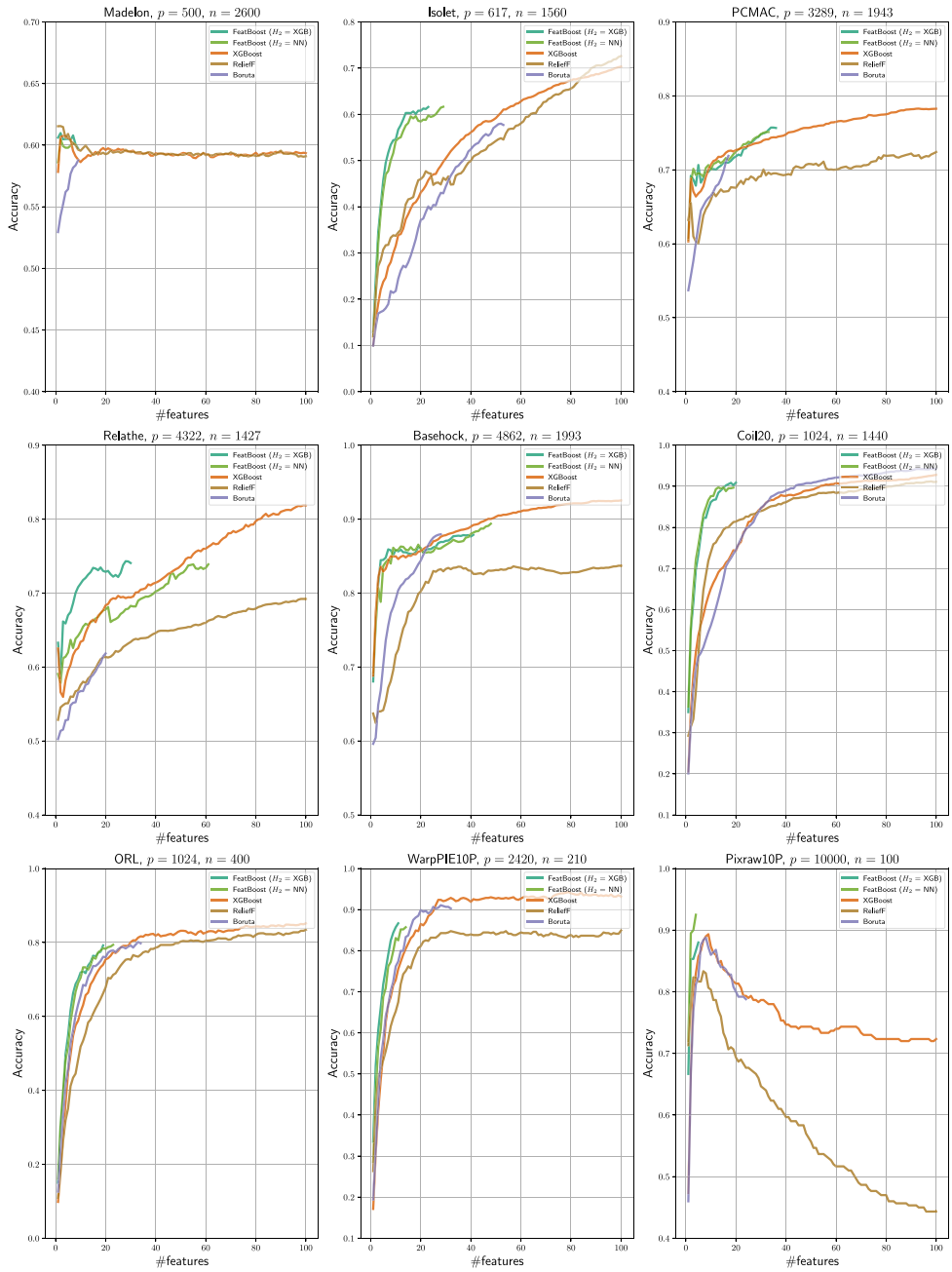

Fig. A.3. Accuracies of the subsets selected by each feature selection algorithm. The validation classifier used to compute the accuracy is Gaussian Naive Bayes.

and reset strategies are given in Algorithm 1.

6. Experimental settings and evaluation

In this section, we describe the data and experimental settings. Then, we briefly describe Boruta and ReliefF, the methods we compared FeatBoost with.

We applied each algorithm to 8 real datasets, and one artificial dataset, Madelon, which was designed to benchmark feature selection algorithms (Guyon et al., 2006). Table 1 contains a description of the datasets, and their dimensions.

On each dataset, we apply an l-by-k-fold cross-validated selection procedure, with $l = 3$, and $k = 10$: We split each dataset into ten equally sized folds, and apply each feature selection algorithm to the training folds separately. Then, we use the selected features to train a classifier on the training folds, and apply it to the held-out test folds, on which the classification performance is evaluated. The performance is measured in terms of the average classification accuracy of the selected subsets on the test data, and the computation time of the feature selection algorithm. We repeat the entire procedure $l = 3$ times with random shuffles of the sample set, for a total of 30 runs of each algorithm. A similar validation procedure is used by Song, Ni, and Wang (2013).

In cases where an algorithm selects more than 100 features, we only evaluate the accuracies of the first 100. Moreover, since each algorithm produces multiple subsets, we exclude those that could skew the average performance at the validation stage. Therefore, for each algorithm, we evaluate only the resulting subsets with a number of features equal to or larger than the mode of all subset sizes for that algorithm. Subsets which are larger than the mode are truncated to have a number of features equal to the mode.

Table 1
Descriptions of datasets[†] used in the comparison.

Name	#Features	#Samples	#Classes	Domain
Madelon	500	2600	2	Synthetic
Isolet	617	1560	2	Speech recognition
PCMAC	3289	1943	2	Text
Relathe	4322	1427	2	Text
Basehock	4862	1993	2	Text
Coil20	1024	1440	20	Face image
ORL	1024	400	40	Face image
WarpPIE10P	2420	210	10	Face image
Pixar10P	10000	100	10	Face image

[†] The datasets are ordered by their domain and number of features. They were obtained from the ASU feature selection repository.

We used a Nearest Neighbor classifier to validate the selected subsets. A Nearest Neighbor classifier is a sensible choice for validating feature subsets, as its performance is more likely to suffer from the inclusion of irrelevant, redundant, or noisy features, when compared to more complex classifiers (Loughrey & Cunningham, 2005). Validation with a Gaussian Naive Bayes classifier and an XGBoost classifier with default parameters are given in Appendix A.

In addition to subset accuracy, we evaluate the performance in terms of computation time of the algorithms, and the redundancy rate (RED) (Yamada, Jitkrittum, Sigal, Xing, & Sugiyama, 2014; Zhao, Wang, & Liu, 2010):

$$\mathrm{RED}(\mathscr{X}) = \frac{1}{p(p-1)} \sum_{f_i, f_j \in \mathscr{X}, i>j} |\rho(f_i, f_j)|, \tag{4}$$

where $\rho(f_i, f_j)$ is the Pearson correlation coefficient between features f_i and f_j. A large value of $\mathrm{RED}(\mathscr{X})$ means that subset $\mathscr{X}$ contains high redundancy. Thus, lower values of $\mathrm{RED}(\mathscr{X})$ are desired in feature selection.

The average computation time of each method is given in Table 2. The average redundancy rates for up to the top 10 and top 100 features selected from each method are given in Tables 3 and 4. We examine the top 10 as well as the complete top 100 features because in most cases, FeatBoost and Boruta select smaller subsets than the other methods, which are ranking algorithms that always select up to the user defined 100 features. Comparing the redundancy rate of up to the top 10 features leads to the subsets across all compared methods to be of similar size, and thus a better assessment of redundancy. Moreover, smaller subsets

Table 2
The mean and standard deviation of computation time in minutes of each feature selection algorithm.

	FeatBoost (H_2 = XGB)	FeatBoost (H_2 = NN)	XGBoost	ReliefF	Boruta
Madelon ($p = 500, n = 2600$)	58.71 (11.89)	3.56 (0.36)	**0.25 (0.01)**	1.02 (0.02)	20.81 (3.01)
Isolet ($p = 617, n = 1560$)	168.45 (31.62)	52.53 (21.78)	**1.04 (0.02)**	0.72 (0.01)	154.91 (3.23)
PCMAC ($p = 3289, n = 1943$)	186.07 (57.88)	45.07 (12.04)	**1.11 (0.03)**	2.81 (0.08)	105.67 (4.79)
Relathe ($p = 3289, n = 1943$)	163.38 (32.6)	60.92 (21.09)	**1.08 (0.01)**	2.77 (0.03)	91.73 (3.06)
Basehock ($p = 4862, n = 1993$)	217.52 (40.95)	73.33 (24.3)	**1.54 (0.07)**	4.34 (0.1)	131.14 (4.76)
Coil20 ($p = 1024, n = 1440$)	333.85 (60.68)	36.67 (9.27)	**1.18 (0.04)**	2.19 (0.06)	241.23 (17.47)
ORL ($p = 1024, n = 400$)	168.38 (48.91)	18.18 (4.65)	**0.56 (0.03)**	0.57 (0.03)	102.35 (12.82)
WarpPIE10P ($p = 2420, n = 210$)	22.91 (4.88)	5.27 (1.45)	**0.25 (0)**	3.28 (0.05)	24.12 (1.06)
Pixraw10P ($p = 10000, n = 100$)	6.91 (1.51)	3.4 (0.49)	**0.37 (0.01)**	5.87 (0.08)	41.15 (0.7)

Table 3
The mean and standard deviation of the redundancy rate of up to the top 10 selected features of each algorithm.

	FeatBoost (H_2 = XGB)	FeatBoost (H_2 = NN)	XGBoost	ReliefF	Boruta
Madelon ($p = 500, n = 2600$)	**0.125 (0.023)**	0.144 (0.029)	0.139 (0.022)	0.172 (0.005)	0.145 (0.028)
Isolet ($p = 617, n = 1560$)	0.074 (0.009)	**0.058 (0.012)**	0.067 (0.007)	0.059 (0.005)	0.066 (0.003)
PCMAC ($p = 3289, n = 1943$)	0.016 (0.008)	**0.007 (0.004)**	0.021 (0.006)	0.017 (0.003)	0.021 (0.006)
Relathe ($p = 3289, n = 1943$)	**0.017 (0.006)**	0.018 (0.005)	0.019 (0.004)	0.024 (0.002)	0.017 (0.005)
Basehock ($p = 4862, n = 1993$)	0.008 (0.002)	**0.006 (0.003)**	0.007 (0.002)	0.008 (0.002)	0.007 (0.001)
Coil20 ($p = 1024, n = 1440$)	**0.118 (0.021)**	0.139 (0.031)	0.122 (0.014)	0.176 (0.018)	0.127 (0.021)
ORL ($p = 1024, n = 400$)	0.162 (0.022)	**0.14 (0.022)**	0.148 (0.009)	0.166 (0.008)	0.142 (0.006)
WarpPIE10P ($p = 2420, n = 210$)	0.32 (0.029)	0.287 (0.019)	0.281 (0.007)	0.286 (0.006)	**0.274 (0.016)**
Pixraw10P ($p = 10000, n = 100$)	**0.431 (0.033)**	0.444 (0.03)	0.466 (0.016)	0.474 (0.004)	0.47 (0.001)

Table 4
The mean and standard deviation of the redundancy rates of up to the top 100 selected features of each algorithm.

	FeatBoost (H_2 = XGB)	FeatBoost (H_2 = NN)	XGBoost	ReliefF	Boruta
Madelon ($p = 500, n = 2600$)	0.12 (0.027)	0.144 (0.029)	**0.013 (0)**	0.014 (0)	0.137 (0.024)
Isolet ($p = 617, n = 1560$)	0.076 (0.005)	**0.058 (0.012)**	0.104 (0.003)	0.093 (0)	0.066 (0.005)
PCMAC ($p = 3289, n = 1943$)	0.015 (0.008)	**0.007 (0.004)**	0.012 (0.002)	0.011 (0.001)	0.02 (0.005)
Relathe ($p = 3289, n = 1943$)	0.016 (0.005)	0.018 (0.005)	**0.013 (0.001)**	0.014 (0.001)	0.017 (0.005)
Basehock ($p = 4862, n = 1993$)	0.008 (0.002)	**0.006 (0.003)**	0.011 (0.001)	0.012 (0.001)	0.008 (0.002)
Coil20 ($p = 1024, n = 1440$)	0.118 (0.021)	0.139 (0.031)	**0.113 (0.004)**	0.122 (0.004)	0.124 (0.017)
ORL ($p = 1024, n = 400$)	0.16 (0.022)	**0.14 (0.022)**	0.154 (0.006)	0.158 (0.005)	0.142 (0.009)
WarpPIE10P ($p = 2420, n = 210$)	0.32 (0.026)	0.287 (0.019)	0.31 (0.008)	0.319 (0.004)	**0.275 (0.018)**
Pixraw10P ($p = 10000, n = 100$)	**0.433 (0.03)**	0.444 (0.03)	0.44 (0.011)	0.44 (0.006)	0.471 (0.001)

could better reflect the redundancy limiting ability of the feature selection method when compared to larger sets, since the redundancy rate of the latter could be a reflection of the inherent level of redundancy in the complete dataset.

6.1. Compared methods

6.1.1. XGBoost

Since we build FeatBoost around a specific feature importance score, one derived from an XGBoost classifier, then a suitable benchmark to compare against is the same base score but with a simpler threshold. For that purpose, we define the first method in the comparison to be the feature importance scores from the same classifier used for ranking in FeatBoost, with the threshold being the mean of all feature scores.[3] That is, features with an importance score lower than the mean of all feature scores are discarded.

6.1.2. Boruta

The second method we compare against is Boruta. In the comparison, we use XGBoost instead of random forest as the base ranking algorithm for Boruta. That way, we are able to achieve a fairer comparison, and determine which of the approaches makes better use of the same underlying feature scores. Moreover, since Boruta does not rank the elements of the selected subset by default, we post-rank them with an additional fitting of the classifier. This allows us to compare the performance of all methods iteratively with each added feature.

6.1.3. ReliefF

Finally, we compare FeatBoost to the ReliefF algorithm (Kira & Rendell, 1992). ReliefF is a powerful filter-based approach which belongs to the Relief family of feature selection methods (Urbanowicz, Meeker, La Cava, Olson, & Moore, 2018; Kira & Rendell, 1992). We use it in the comparison to represent a baseline of filter-based approaches.

Since Relief-based methods are feature ranking algorithms, a suitable threshold is needed in order to use them for feature subset selection. As in XGBoost, we used the mean value of the scores as a lower threshold.

We configured the algorithms as follows:

1. **XGBoost:** We configured XGBoost with 100 trees, and a maximum tree depth of 20. We set the maximum depth parameter to a high value in order to detect higher order feature interactions that might be present in some datasets (Johnson, 2009). We set the remaining parameters of the classifier to their default values.[4]
2. **FeatBoost:** We used FeatBoost in two configurations. In the first, we set H_1 and H_2 to be the same XGBoost classifier used individually. In the second, we set H_2 as a Nearest Neighbor classifier. We set the remaining parameters as follows: $k = 3, m = 50, p' = 100$, and $\epsilon = 10^{-18}$.
3. **Boruta:** We used Boruta with the same classifier used in XGBoost, and default settings otherwise. We implemented the algorithm with the BorutaPy Python package, which we modified to be compatible with XGBoost.
4. **ReliefF:** We used ReliefF with a number of neighbors equal to 10 and implemented the algorithm with the Skrebate Python package.

7. Results and discussion

The results of the feature selection comparison are summarized in Fig. 1, which shows the average classification accuracies of the subsets selected by the compared algorithms. The average computation times of each algorithm on all datasets are shown in Table 2.

In all datasets, FeatBoost selects better performing features in the leading ranks than the feature importance score on which it is based, XGBoost. This shows that sample re-weighting and model evaluation improve the performance of the base ranking. Moreover, the automatic stopping conditions in FeatBoost lead to selecting significantly smaller subsets than the mean-value threshold that we used for XGBoost.

In most datasets, FeatBoost outperforms Boruta and ReliefF as well, reaching higher accuracies with fewer features. This is most apparent in Isolet, PCMAC, COIL20, Orl, and WarpPIE10P. In those datasets, the performance of subsets selected by FeatBoost converges significantly faster than the second best method, indicating that the relevant features are found with smaller subsets.

In terms of computation time, XGBoost is the most efficient approach across all datasets, followed by ReliefF. This is expected since the former ranks features with a single fitting of the data, while the latter is a filter method that performs no model fitting. As for FeatBoost and Boruta, we observe that the former, when configured with an XGBoost wrapped classifier, is slightly faster than Boruta, which uses the same classifier to provide its base feature ranking. On the other hand, when FeatBoost uses a NN classifier, it becomes significantly faster than Boruta.

It is worth noting that this increase in efficiency in FeatBoost – obtained by using NN instead of XGBoost as the evaluation classifier – does not sacrifice the performance of the algorithm. In fact, this configuration performs better when the validation classifier is also NN (Fig. 1). It also performs well when XGBoost is the validation classifier (Fig. A.2). This demonstrates that the modular architecture of the algorithm can take advantage of a powerful ranking algorithm, that of XGBoost, while using a simpler and more efficient evaluation classifier.

Table 3, which shows the redundancy rates of up to the top 10 selected features, indicates that FeatBoost in either of its iterations selects subsets with lower redundancy than the other compared methods. When the top 100 features in ReliefF and XGBoost are examined (Table 4), FeatBoost with an NN classifier still retains its advantage, performing best in 4 out of 9 datasets. In both cases, Boruta performs best on a single dataset; WarpPIE10P.

The algorithm's ability to reduce redundancy, at least in the top ranked features, is very promising, since this resulted only from the sample re-weighting and model evaluation procedures, and not from explicit minimization of redundancy, as is the case in other feature selection approaches (Peng, Long, & Ding, 2005; Zhao et al., 2010; Yamada et al., 2014).

8. Conclusion

We showed that the proposed FeatBoost algorithm, which is based on boosting, and a stage-wise greedy procedure, is able to use the feature importance scores derived from an ensemble of decision-trees to select high performing feature subsets for classification problems. The resulting subsets outperform those obtained by simple thresholding of the baseline scores. They also outperform in most cases the Boruta algorithm, which we configured to use the same feature scores as a basis, and improved with a post-ranking of the selected subsets.

The computational cost of the algorithm is sensitive to several design choices, most importantly, the ranking algorithm, the parameter m, and the wrapped classifier. We have shown that changing the latter from XGBoost to the much simpler Nearest Neighbor improved the speed of the algorithm without significantly affecting the performance of the selected features.

The proposed boosting framework, and the two-step selection procedure, circumvent the weaknesses in the importance scores of tree-based models externally, without changing how the scores are computed. The algorithm is therefore independent of the particular scores being used, or the wrapped classifier. It would be useful to investigate the use of other classifiers, and to test if other ranking algorithms, like filter-based Relief methods, would react similarly within the same algorithm.

[3] We used the feature importance type Gain as it was defined in XGBoost's stable Python release 0.90.

[4] Default values as per XGBoost's stable Python release 0.90.

CRediT authorship contribution statement

Ahmad Alsahaf: Conceptualization, Methodology, Data curation, Software, Formal analysis, Investigation, Writing - original draft. **Nicolai Petkov:** Conceptualization, Funding acquisition, Methodology, Resources, Supervision, Writing - review & editing. **Vikram Shenoy:** Data curation, Formal analysis, Methodology, Investigation, Software. **George Azzopardi:** Conceptualization, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Supervision, Validation, Writing - review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

We thank Nik Dijkema for his work on time complexity analysis of the algorithm. This paper is part of the Breed4Food research program, project Smart animal breeding with advanced machine learning with project number 14295, which is financed by the Netherlands Organisation for Scientific Research (NWO), the Dutch Ministry of Economic Affairs (TKI Agri & Food project 12018) and the Breed4Food Partners Cobb Europe CRV, Hendrix Genetics, and Topigs Norsvin.

Appendix A. Validation with other classifiers

In this appendix, we give the results of the main feature selection comparison (described in Section 6) with two additional validation classifiers, XGBoost, and Gaussian Naive Bayes. The results are shown Figs. A.2 and A.3.

References

AlNuaimi, N., Masud, M. M., Serhani, M. A., & Zaki, N. (2020). Streaming feature selection algorithms for big data: A survey. In *Applied Computing and Informatics*.

Alsahaf, A., Azzopardi, G., Ducro, B., Veerkamp, R. F., & Petkov, N. (2018). Predicting slaughter weight in pigs with regression tree ensembles. In *APPIS* (pp. 1–9).

Barddal, J. P., Enembreck, F., Gomes, H. M., Bifet, A., & Pfahringer, B. (2019). Boosting decision stumps for dynamic feature selection on data streams. *Information Systems, 83*, 13–29.

Bolón-Canedo, V., & Alonso-Betanzos, A. (2019). Ensembles for feature selection: a review and future trends. *Information Fusion, 52*, 1–12.

Borboudakis, G., & Tsamardinos, I. (2019). Forward-backward selection with early dropping. *The Journal of Machine Learning Research, 20*, 276–314.

Breiman, L. (2001). Random forests. *Machine Learning, 45*, 5–32.

Breiman, L. (2002). Manual on setting up, using, and understanding random forests v3. 1. Statistics Department University of California Berkeley, CA, USA, 1, 58.

Chen, T., & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 785–794).

Christin, C., Hoefsloot, H. C., Smilde, A. K., Hoekman, B., Suits, F., Bischoff, R., & Horvatovich, P. (2013). A critical assessment of feature selection methods for biomarker discovery in clinical proteomics. *Molecular & Cellular Proteomics, 12*, 263–276.

Das, S. (2001). Filters, wrappers and a boosting-based hybrid for feature selection. In *Icml (Vol. 1)* (pp. 74–81).

Dash, M., & Liu, H. (1997). Feature selection for classification. *Intelligent Data Analysis, 1*, 131–156.

Doshi-Velez, F., & Kim, B. (2017). Towards a rigorous science of interpretable machine learning. arXiv preprint arXiv:1702.08608.

Early, K., Fienberg, S., & Mankoff, J. (2016). Cost-effective feature selection and ordering for personalized energy estimates. In *Workshops at the thirtieth AAAI conference on artificial intelligence*.

El Aboudi, N., & Benhlima, L. (2016). Review on wrapper feature selection approaches. In *2016 International conference on engineering & MIS (ICEMIS)* (pp. 1–5). IEEE.

Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences, 55*, 119–139.

Galelli, S., & Castelletti, A. (2013). Tree-based iterative input variable selection for hydrological modeling. *Water Resources Research, 49*, 4295–4310.

Gao, W., Hu, L., & Zhang, P. (2018). Class-specific mutual information variation for feature selection. *Pattern Recognition, 79*, 328–339.

Genuer, R., Poggi, J.-M., & Tuleau-Malot, C. (2010). Variable selection using random forests. *Pattern Recognition Letters, 31*, 2225–2236.

Gunning, D. (2017). Explainable artificial intelligence (xai). Defense Advanced Research Projects Agency (DARPA), nd Web, 2.

Guyon, I., & Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research, 3*, 1157–1182.

Guyon, I., Li, J., Mader, T., Pletscher, P. A., Schneider, G., & Uhr, M. (2006). Feature selection with the CLOP package. *Technical Report*.

Holzinger, A. (2018). From machine learning to explainable ai. In *2018 World symposium on digital intelligence for systems and machines (DISA)* (pp. 55–66). IEEE.

Ibrahim, R. A., Ewees, A. A., Oliva, D., Abd Elaziz, M., & Lu, S. (2019). Improved salp swarm algorithm based on particle swarm optimization for feature selection. *Journal of Ambient Intelligence and Humanized Computing, 10*, 3155–3169.

Johnson, N. (2009). A study of the nips feature selection challenge.

Jović, A., Brkić, K., & Bogunović, N. (2015). A review of feature selection methods with applications. In *2015 38th International convention on information and communication technology, electronics and microelectronics (MIPRO)* (pp. 1200–1205). Ieee.

Kira, K., & Rendell, L. A. (1992). A practical approach to feature selection. In *Machine Learning Proceedings 1992* (pp. 249–256). Elsevier.

Kohavi, R., & John, G. H. (1997). Wrappers for feature subset selection. *Artificial Intelligence, 97*, 273–324.

Kumar, V., & Minz, S. (2014). Feature selection: a literature review. *SmartCR, 4*, 211–229.

Kursa, M. B., Rudnicki, W. R., et al. (2010). Feature selection with the boruta package. *Journal of Statistical Software, 36*, 1–13.

Liu, H., Liu, L., & Zhang, H. (2009). Boosting feature selection using information metric for classification. *Neurocomputing, 73*, 295–303.

Loughrey, J., & Cunningham, P. (2005). *Using early stopping to reduce overfitting in wrapper-based feature weighting*. Technical Report Trinity College Dublin, Department of Computer Science.

Louppe, G., Wehenkel, L., Sutera, A., & Geurts, P. (2013). Understanding variable importances in forests of randomized trees. In *Advances in neural information processing systems* (pp. 431–439).

Luckner, M., Topolski, B., & Mazurek, M. (2017). Application of xgboost algorithm in fingerprinting localisation task. In *IFIP international conference on computer information systems and industrial management* (pp. 661–671). Springer.

Lundberg, S. M., Erion, G. G., & Lee, S.-I. (2018). Consistent individualized feature attribution for tree ensembles. arXiv preprint arXiv:1802.03888.

Mafarja, M., & Mirjalili, S. (2018). Whale optimization approaches for wrapper feature selection. *Applied Soft Computing, 62*, 441–453.

Murauer, B., & Specht, G. (2018). Detecting music genre using extreme gradient boosting. In *Companion Proceedings of the web conference 2018* (pp. 1923–1927).

Nardone, D., Ciaramella, A., & Staiano, A. (2019). A sparse-modeling based approach for class specific feature selection. *PeerJ Computer Science, 5*, Article e237.

Nguyen, T.-T., Huang, J. Z., & Nguyen, T. T. (2015). Unbiased feature selection in learning random forests for high-dimensional data. *The Scientific World Journal, 2015*.

Peker, M., Arslan, A., Şen, B., Çelebi, F. V., & But, A. (2015). A novel hybrid method for determining the depth of anesthesia level: Combining relieff feature selection and random forest algorithm (relieff+ rf). In *2015 International symposium on innovations in intelligent systems and applications (INISTA)* (pp. 1–8). IEEE.

Peng, H., Long, F., & Ding, C. (2005). Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 27*, 1226–1238.

Pineda-Bautista, B. B., Carrasco-Ochoa, J. A., & Martinez-Trinidad, J. F. (2011). General framework for class-specific feature selection. *Expert Systems with Applications, 38*, 10018–10024.

Rao, H., Shi, X., Rodrigue, A. K., Feng, J., Xia, Y., Elhoseny, M., Yuan, X., & Gu, L. (2019). Feature selection based on artificial bee colony and gradient boosting decision tree. *Applied Soft Computing, 74*, 634–642.

Song, Q., Ni, J., & Wang, G. (2013). A fast clustering-based feature subset selection algorithm for high-dimensional data. *IEEE Transactions on Knowledge and Data Engineering, 25*, 1–14.

Strobl, C., Boulesteix, A.-L., Kneib, T., Augustin, T., & Zeileis, A. (2008). Conditional variable importance for random forests. *BMC Bioinformatics, 9*, 307.

Strobl, C., Boulesteix, A.-L., Zeileis, A., & Hothorn, T. (2007). Bias in random forest variable importance measures: Illustrations, sources and a solution. *BMC Bioinformatics, 8*, 25.

Tabus, I., & Astola, J. (2005). Gene feature selection. *Genomic Signal Processing and Statistics*, 67–92.

Tang, J., Alelyani, S., & Liu, H. (2014). Feature selection for classification: A review. *Data Classification: Algorithms and Applications*, 37.

Tieu, K., & Viola, P. (2004). Boosting image retrieval. *International Journal of Computer Vision, 56*, 17–36.

Tuv, E., Borisov, A., Runger, G., & Torkkola, K. (2009). Feature selection with ensembles, artificial variables, and redundancy elimination. *Journal of Machine Learning Research, 10*, 1341–1366.

Urbanowicz, R. J., Meeker, M., La Cava, W., Olson, R. S., & Moore, J. H. (2018). Relief-based feature selection: introduction and review. *Journal of Biomedical Informatics*.

Yamada, M., Jitkrittum, W., Sigal, L., Xing, E. P., & Sugiyama, M. (2014). High-dimensional feature selection by feature-wise kernelized lasso. *Neural Computation, 26*, 185–207.

Zhao, Z., Wang, L., & Liu, H. (2010). Efficient spectral feature selection with minimum redundancy. In Proceedings of the AAAI Conference on Artificial Intelligence. volume 24.