E2E-FS: An End-to-End Feature Selection Method for Neural Networks

Brais Cancela, Verónica Bolón-Canedo, and Amparo Alonso-Betanzos

Abstract—Classic embedded feature selection algorithms are often divided in two large groups: tree-based algorithms and LASSO variants. Both approaches are focused in different aspects: while the tree-based algorithms provide a clear explanation about which variables are being used to trigger a certain output, LASSO-like approaches sacrifice a detailed explanation in favor of increasing its accuracy. In this paper, we present a novel embedded feature selection algorithm, called End-to-End Feature Selection (E2E-FS), that aims to provide both accuracy and explainability in a clever way. Despite having non-convex regularization terms, our algorithm, similar to the LASSO approach, is solved with gradient descent techniques, introducing some restrictions that force the model to specifically select a maximum number of features that are going to be used subsequently by the classifier. Although these are hard restrictions, the experimental results obtained show that this algorithm can be used with any learning model that is trained using a gradient descent algorithm.

Index Terms—Feature selection, end-to-end, non-convex problem

1 INTRODUCTION

High dimensional problems are very common nowadays and pose an important challenge for machine learning researchers. Dealing with thousands or even millions of features is not practical, particularly as some will be redundant or non-informative. It is therefore important to correctly identify the relevant features for a given task in a process known as feature selection (FS). Reducing the dimensionality of a problem has several acknowledged advantages, such as improved interpretability (and therefore explainability), reduced execution times, and in some cases, improved learning performance [1].

FS methods can be grouped as classifier independent (filter methods) and classifier independent (wrappers and embedded methods). Filters use independent metrics (such as mutual information, correlation, or statistics) to decide which features are more relevant with respect to the predictive class. In this way, the selected features are generic, and the process to extract them is usually not computationally expensive. Examples of filter approaches are Mutual Information (MD) [2], ReliefF [3] and the Infinite Feature Selection variants, InfFS [4], [5] and ILFS [6].

Wrappers and embedded procedures implement a learning method (e.g., a classifier) to determine the subset of relevant features [7]. Wrappers search through the space of features, choosing subsets of features and evaluating their importance with respect to the class using the classification accuracy of a particular classifier. Typical strategies for constructing the different subsets of features to be tested are sequential forward selection or sequential backward selection. This approach tends to be computationally expensive (since the algorithm is trained for each subset of features to test), and the selected features are specific to the classifier used to obtain them. Embedded methods, which lie halfway between wrappers and filters (in terms of their computational cost), determine the relevant features by training a classifier (for example, optimizing weights in a neural network [8]). Embedded methods, as well as less computationally costly, are less prone to overfitting than wrappers, and have the additional advantage that both FS and classification training can be simultaneous.

One of the best known embedded methods is Recursive Feature Elimination for Support Vector Machine (SVM-RFE) [9], which computes the importance of the features in training a SVM. More recently, the Saliency-based Feature Selection (SFS) method [10] uses the saliency technique [11] to infer the most relevant features. However, both methods have a high computational cost, as a classifier has to be trained several times to obtain a good result. LASSO [12] is also very popular, as it is based on an extracted subset that includes shape and density features. However, contrary to the filter methods, there is no control over the number of features that are finally chosen.

In this paper we aim to merge the best characteristics of filter methods and the LASSO approach in a single algorithm which we call End-to-End Feature Selection (E2E-FS). The advantages of our approach are as follows:

1) It can be used with any model trained using gradient descent techniques and so is not restricted to classification problems.

2) Similar to LASSO, FS is performed simultaneously with model training and so considerably reduces the
computational cost and avoids the computation of multiple models as in SVM-RFE.

3) Similar to ranker filter methods, and contrary to LASSO, we can specify the maximum number of features that are finally selected while still using gradient descent techniques.

4) It is very efficient in terms of both computational time and memory, as only a vector of the size of the number of the initial features is required.

5) It is implemented on a plug-and-play basis. Just by defining the model and the number of features, the algorithm can be trained with a single line of Python code.¹

Other embedded methods available in the literature can specify an exact number of features and train a learning model in a single step without using a recursive approach. For instance, Balin et al. [13] proposed a similar approach to ours, but with three main differences: their method can only be used with linear architectures; the mask variable is not a vector but a matrix, making the computational cost highly dependent on the number of selected features; and it does not introduce any regularization term in the mask, but relies on a temperature parameter to force the matrix to have the desired binary structure, while convergence is not guaranteed. Similarly, Chen et al. [14] created the same matrix, but as an output of an encoder-like architecture. FS is first carried out on the instance level, and is performed over the median of the instance-based feature selection matrices. Although this approach can also be used with Convolutional Neural Networks (CNNs), it also has the same problems as described for the previous approach.

In this paper, we demonstrate that, using only constraints and regularization parameters, we can obtain a final solution that is not only more accurate than the two approaches mentioned above, but also reduces both computational cost and memory requirements.

The rest of the paper is organized as follows: Section 2 describes the intuition behind our proposal; Section 3 describes implementation of our E2E-FS algorithm; Section 4 reports experimental results for a wide range of public datasets; and finally, Section 5 offers some conclusions and describes future work.

2 E2E-FS ALGORITHM

Let \( \mathbf{X} \in \mathbb{R}^{N \times F} \) be our input data, where \( N \) is the number of instances and \( F \) is the total number of different features. Let \( \mathbf{Y} \in \mathbb{R}^{N \times C} \) be the expected output, where \( C \) is the number of classes. For the sake of simplicity, only classification problems will be taken into account, although our approach can also be used for other problems (e.g., regression) that can be solved using gradient descent techniques, without making further modifications. Thus, let \( \mathbf{\hat{Y}} = f(\mathbf{X}; \Theta) \in \mathbb{R}^{N \times C} \) be our classification model, where \( \Theta \) are the classifier parameters. Our aim is to solve a minimization problem by forcing the classifier to only select a maximum number of features, denoted by \( M \). Formally speaking, our algorithm aims to solve the following minimization problem:

\[
\begin{align*}
\text{minimize} & \quad \mathcal{L}(f(\mathbf{y} \circ \mathbf{X}; \Theta), \mathbf{Y}) \\
\text{subject to} & \quad \mathbf{y} \in \{0, 1\}^F, \\
& \quad \|\mathbf{y}\|_1 \leq M.
\end{align*}
\]

where \( \mathcal{L} \) is the loss function, \( M \) is the maximum number of features to be selected, and \( \mathbf{y} \) is the mask layer. The idea is simple: we train a classification problem while introducing a binary mask layer to be in charge of selecting, at most, \( M \) relevant features. Initially, this is not a problem that can be solved using gradient descent techniques, as the binary mask is not differentiable. However, some approximations can be made to fulfill the requirements.

3 E2E-FS IMPLEMENTATION

The first decision is to select the shape of \( \mathbf{y} \). The easiest approach is to select \( \mathbf{y} \in \mathbb{R}^{M \times F} \) (as [14]). We can then force \( \|\mathbf{y}\|_\infty = 1 \) and \( \|\mathbf{y}\|_1 \leq 1 \), i.e., all zeros but one in each row, and at most, one non-zero per column. This problem can be solved using gradient descent techniques and the \( l_{1-2} \) regularization [15], as described in [16] for learning permutation matrices. However, this approach would require a huge amount of memory space in Big Data environments. For instance, selecting 10,000 features from an initial dataset with more than 100,000 variables would require, for 32-bit floating-point precision, nearly 4 GB merely to store the \( \mathbf{y} \) matrix. For this reason, we developed an alternative solution that only requires a vector of size \( F \), that is, \( \mathbf{y} \in \mathbb{R}^F \). This solution is based on introducing two extra constraints in the loss function.

3.1 E2E-FS Using Soft Regularization Techniques

We fulfill the restrictions using only derivable regularization terms. Thus, our solution, called E2E-FS, transforms the initial problem (Eq. (1)) into

\[
\begin{align*}
\text{minimize} & \quad \mathcal{L}_1(f(\mathbf{\hat{y}} \circ \mathbf{X}; \Theta), \mathbf{Y}) \\
\text{subject to} & \quad \mathbf{\hat{y}} \in [0, 1]^F, \\
& \quad \frac{\mathcal{L}_1(\mathbf{\hat{y}})}{L_{1-2}} + \frac{\mathcal{L}_M(\|\mathbf{\hat{y}}\|_1)}{L_{1-2}} + (1 + \mu) \|\mathbf{M} - \|\mathbf{\hat{y}}\|_1\| = 0,
\end{align*}
\]

where \( \mu > 0 \) is a hyper-parameter (set by default to 1).

The intuition behind the idea is simple: \( L_{1-2} \) forces \( \mathbf{\hat{y}} \) values to be binary (either 0 or 1), while \( L_M \) ensures that the sum of all values in \( \mathbf{\hat{y}} \) is close to the desired maximum number of features \( M \). By default, we initialized \( \mathbf{\hat{y}} = 1 \).

Complexity. The complexity of this approach remains at \( O(F \cdot F) \), as it only introduces a vector of size \( F \) with regularization terms over the model \( f \).

Implementation Details. As the restrictions in this approach are more relaxed, we cannot guarantee that \( M \) features are selected after a fixed number of epochs. Instead, we need to check the loss function for confirmation. We can ensure that

\[
\mathcal{L}_1 = \mathcal{L}_{1-2} + \mathcal{L}_M = 0 \Rightarrow nzz(\mathbf{\hat{y}}) = M,
\]

1. Code: https://github.com/braisCB/E2E-FS.git

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
where \( \text{nnz}(\cdot) \) stands for number of non zeros.

Even if Eq. (1) is changed into a problem with regularization terms, we still need to satisfy the constraints to obtain the desired result. Thus, we need to force \( \mathcal{L}_{1-2} \) and \( \mathcal{L}_M \) to be zero. To do so, we define the gradient w.r.t. \( \hat{\mathbf{y}} \) as

\[
\frac{\partial \mathcal{L}_f}{\partial \hat{\mathbf{y}}} = (1 - \alpha) \frac{\partial \mathcal{L}_f}{\partial \mathbf{y}} + \alpha \frac{\partial \mathcal{L}_g}{\partial \mathbf{y}},
\]

where \( \alpha \in [0, 1] \) is a hyperparameter to control the gradient focus between the classification loss and the regularization terms. Note that the same \( \alpha \) hyperparameter controls both the \( \mathcal{L}_{1-2} \) and \( \mathcal{L}_M \) terms. In the proof of convergence we show why these two terms should always be treated as a whole.

Applying gradient descent to minimize \( \mathcal{L}_f \hat{\mathbf{y}} \) presents a couple of issues, described below along with possible solutions.

1) **\( \mathcal{L}_{1-2} \) is non-convex.** As depicted in Fig. 1a, \( \mathcal{L}_{1-2} \) is a concave function which, when restricted to \( \hat{\mathbf{y}} \in [0, 1] \), has its minimum values over the corners (that is, when \( \hat{\mathbf{y}} = \{0, 1\} \)). This is a problem because this function, when reaches its minimum, may lead to extreme configurations, like selecting or discarding all features. However, when the \( \mathcal{L}_M \) loss is also introduced (see Fig. 1b), we can guarantee that the minimum is reached if and only if the number of selected features is equal to \( M \). While we still have a concave function that leads to gradient acceleration on approaching the minimum, this is not a huge problem, as the purpose of this algorithm is to force every value in the vector \( \hat{\mathbf{y}} \) to reach one of its extreme values (0 or 1). In the proof of convergence, we show how our combination \( \mathcal{L}_{1-2} + \mathcal{L}_M \) can prevent our algorithm from discarding more features than \( M \) and also increase the separation between relevant and irrelevant features.

2) **Classification problem derivative and regularization term scales may differ.** Initially, we would like the gradient of gamma to be guided by the classification loss \( \mathcal{L}_f \) and gradually change the \( \alpha \) parameter to 1, forcing the training to remove the least important features. To ensure both values have the same scale we change Eq. (4) to

\[
\frac{\partial \mathcal{L}_f}{\partial \hat{\mathbf{y}}} = \beta_{th} \left( (1 - \alpha) z \frac{\partial \mathcal{L}_f}{\partial \hat{\mathbf{y}}} + \alpha z \frac{\partial \mathcal{L}_g}{\partial \hat{\mathbf{y}}} \right),
\]

where the normalization equations are

\[
z(x) = \frac{x}{\|x\|_2},
\]

and

\[
\beta_{th}(x) = \min(th, \|x\|_\infty) \text{ sign}(x) \frac{x}{\|x\|_\infty + \epsilon}.
\]

By default, \( th = 0.1 \). The \( \text{ sign}(x) \) is used to prevent the reappearance of features that have been discarded. Experimental evaluations show that this solution can dramatically increase speed without reducing accuracy.

Regarding \( \alpha \), it is difficult to establish a satisfactory value, as a low value can cause restrictions not to be fulfilled, whereas a high value can create the binary vector \( \hat{\mathbf{y}} \) without taking into account the information of the problem (encoded in \( \mathcal{L}_f \)). To prevent this issue, we replace \( \alpha \) with a moving factor defined as

\[
\alpha_t = \min(1, t/T),
\]

where \( t \) is the training iteration and \( T \) is a hyperparameter that controls how smoothly we want to introduce the \( \mathcal{L}_\hat{\mathbf{y}} \) loss in the \( \hat{\mathbf{y}} \) update. The moving \( \alpha_t \) allows gradual changing of the gradient importance from the classification to the regularization loss. In order to speed up binary mask creation, \( \hat{\mathbf{y}} \) is always updated using the Adam optimizer with \( \beta_1 = 0.5 \).

3) **Removing new features as \( \text{nnz}(\mathbf{y}) \) approaches \( M \) is difficult.** This problem is related to \( \mathcal{L}_{1-2} \) behavior, as the gradient descent of values above 0.5 tend to run towards 1 rather than towards 0. In a similar way as explained above, we prevent this by introducing a variant of our E2E-FS algorithm, called E2E-FS-Soft, that replaces the parameter \( M \) with a moving value

\[
M_\rho = \begin{cases} 
(1 - \rho)M & \text{if } \text{nnz}(\mathbf{y}) > M \\
\text{nnz}(\mathbf{y}) & \text{otherwise}
\end{cases}
\]

By default, we set \( \rho = 0.75 \). The \( M_\rho \) parameter allows us to remove features faster. Note that this moving
factor can cause the algorithm to remove more features than expected.

### 3.1.1 Proof of Convergence

This section demonstrates that the constraints of the soft regularization definition (Eq. (2)) are analogous to the initial problem formulation (Eq. (1)). In the first place, we explain why the configuration $L_{1-2} + L_M$ is chosen, and how it can be used to successfully obtain the desired binary vector $\mathbf{y}$.

**Lemma 1.** Given any $M > 0$, $|\mathbf{y}|_1 > M \Rightarrow \frac{\partial L_{1-2} + L_M}{\partial \mathbf{y}} > 0$.

**Proof.** Knowing that $|\mathbf{y}|_1 > M$, we can rewrite $L_M = (1 + \mu) (|\mathbf{y}|_1 - M)$. Thus, $L_{1-2} + L_M = (2 + \mu) |\mathbf{y}|_1 - \|\mathbf{y}\|_2^2 = (1 + \mu) M$. Its gradient is defined as

$$\frac{\partial L_{1-2} + L_M}{\partial \mathbf{y}} = 2 (1 - \mathbf{y})_i + \mu, \quad \forall i \in (1 \ldots F).$$

As, by definition, $\mathbf{y}_i \in [0, 1]$, we have that

$$\min \left( \frac{\partial L_{1-2} + L_M}{\partial \mathbf{y}} \right) = \mu, \quad \forall i \in (1 \ldots F).$$

Given that $\mu > 0$, the proof is complete.

**Lemma 2.** Given any $M > 0$, $|\mathbf{y}|_1 < M \Rightarrow \frac{\partial L_{1-2} + L_M}{\partial \mathbf{y}} < 0$.

**Proof.** As in the previous lemma, we have $L_M = (1 + \mu) (M - |\mathbf{y}|_1)$ and $L_{1-2} + L_M = M - \|\mathbf{y}\|_2^2 - \mu \|\mathbf{y}\|_1$. The gradient is defined as

$$\frac{\partial L_{1-2} + L_M}{\partial \mathbf{y}} = -2 \mathbf{y}_i - \mu, \quad \forall i \in (1 \ldots F).$$

Given that $\mathbf{y}_i \in [0, 1]$, we have

$$\max \left( \frac{\partial L_{1-2} + L_M}{\partial \mathbf{y}} \right) = -\mu, \quad \forall i \in (1 \ldots F).$$

Given that $\mu > 0$, the proof is complete.

**Theorem 1.** The $L_{1-2} + L_M$ regularization term ensures that $|\mathbf{y}|_1 \approx M$, preventing features from falling from the extreme values (0 and 1) while maximizing the separation between them.

**Proof.** According to Lemma 1 we can ensure that, if $|\mathbf{y}|_1 > M$, all $\mathbf{y}$ values will be reduced, preventing them from quickly reaching the extreme value 1. Furthermore, according to the same lemma, we know that the features which reduce their value most are those closer to 0, forcing the system to drop the irrelevant features and maximizing distance with respect to the relevant features. In contrast, if $|\mathbf{y}|_1 < M$, Lemma 2 guarantees that all features increase their values. And, again, the features that increase their value most are those with the highest $\mathbf{y}$ values, maximizing distance from the irrelevant features.

Below we explain when Eq. (2) satisfies the restrictions imposed in our original problem.

**Lemma 3.** $L_{1-2} = 0$ $\Rightarrow$ $\mathbf{y} \in \{0, 1\}$.

**Proof.** Knowing that $\mathbf{y} > 0$, we have $L_{1-2} = 0 \Rightarrow \|\mathbf{y}\|_1 - \|\mathbf{y}\|_2^2 = 0 \Rightarrow \sum_{i=1}^F \mathbf{y}_i (1 - \mathbf{y}_i) = 0$. The rest of the proof is straightforward.

**Lemma 4.** Given any $M > 0$, $L_M = 0 \Rightarrow \|\mathbf{y}\|_1 = M$.

**Proof.** By the definition of $L_M$, the proof is straightforward.

**Lemma 5.** Given any $M > 0$ value, $L_{1-2} + L_M = 0 \Rightarrow \|\mathbf{y}\|_1 = M$ and $\mathbf{y} \in \{0, 1\}$.

**Proof.** By Lemma 3, we know that all $\mathbf{y}$ values are integers. Thus, the 1-norm is also an integer. By Lemma 4, we know that $\|\mathbf{y}\|_1 = M$, thus proving the lemma.

**Theorem 2.** Given any $M > 0$ value, a classifier $f$ with an E2E-FS mask needs to be trained until $L_{1-2} + L_M = 0$ to ensure that M features are selected and the binary mask is properly formed.

**Proof.** By Lemma 5.

**Algorithm 1.** E2E-FS Algorithm. It Outputs the Binary Mask With the $M$ Selected Values. By Default, the Hyperparameters are Set to $(\rho = 0.75$ for E2E-FS-SofT, $T = 1e4$, warmup $= 2e3$, th $= 0.1$, $\alpha_M = 0.99$).

**Input:** $X, Y, f, M_\rho, T,$ warmup, th, $\alpha_M$

1: $t, \alpha_T \leftarrow 0$
2: $\mathbf{y} \leftarrow \text{ones}(F)$
3: while $L_{\mathbf{y}} > 0$ do
4: $X_b, Y_b \leftarrow \text{batch from } X, Y$
5: $Y_{\text{pred}} \leftarrow f(\mathbf{y} \odot X_b)$
6: Train $f$ with $L(f(\mathbf{y}_{\text{pred}}, Y_b)$
7: $\frac{\partial F}{\partial \mathbf{y}} \leftarrow \beta_{\mathbf{y}} ((1 - \alpha) z_{\mathbf{y}}(\mathbf{y}_{\text{pred}}, Y_b)) + \alpha z_{\mathbf{y}}(\mathbf{y}_{\text{pred}}, Y_b))$
8: Update $\mathbf{y}$ with $\frac{\partial F}{\partial \mathbf{y}}$
9: $\mathbf{y} \leftarrow \min (1, \max (0, \mathbf{y}))$
10: $t \leftarrow t + 1$
11: $\alpha_T \leftarrow \min (\alpha_M, \max (0, (t - \text{warmup})/T))$
12: if $\alpha_T = \alpha_M$ then
13: $M_\rho \leftarrow (1 - \epsilon) M_\rho$
14: end if
15: end while

**Output:** $\mathbf{y}$

### 3.2 E2E-FS Algorithm

The E2E-FS core algorithm is described in Algorithm 1. It requires input data $X \in \mathbb{R}^{N \times F}$ and a model $f$. First, we select, in each iteration, a training batch. Then, after the model $f$ is updated, we also update the binary mask $\mathbf{y}$ using Eq. (5). We also manually force that $\mathbf{y} \in [0, 1]$. The rest of the parameters are used to control how the learning parameter $\alpha$ in Eq. (5) changes over time. We first let the model be trained without any influence of the regularization terms ($\alpha = 0$). The number of iterations is controlled by the warmup parameter. We then linearly increase $\alpha$ over a fixed number of iterations controlled by $T$. When the binary vector is successfully created, the algorithm stops.
TABLE 1

FS Approaches Considered in the Experiments According to Their Time and Memory Complexity

| Method     | Type                   | Time complexity             | Memory Complexity          |
|------------|------------------------|----------------------------|----------------------------|
| LASSO-RFE [17] | Wrapper               | $O(|O_f|)$                  | $O(|O_f|)$                  |
| SVM-RFE [9]   | Wrapper               | $O(|O_f| + F)$              | $O(|O_f| + F)$              |
| MIM [2]       | Filter                | $O(N^3F^3)$                | $O(F^3)$                   |
| Fisher [18]    | Filter                | $\approx O(CNF)$           | $O(F^2)$                   |
| ReliefF [3]    | Filter                | $O(|F|)$                   | $O(F)$                     |
| InfFS [4, 5]   | Filter                | $O(N^{2.5} + iN + F + C)$  | $O(F^2)$                   |
| ILFS [6]       | Filter                | $O(|O_{FNC}|)$             | $O(F^2)$                   |
| DFS [19]       | Embedded              | $O(|O_f| + F)$              | $O(|O_f| + F)$              |
| SFS [10]       | Wrapper               | $O(|O_f|)$                  | $O(|O_f| + F)$              |
| CAE [13]       | Embedded              | $O(|O_f| + O_{AE} + FM)$    | $O(|O_f| + O_{AE} + FM)$    |
| L2X [14]       | Embedded              | $O(|O_f| + F)$              | $O(|O_f| + F)$              |
| E2E-FS        | Embedded / Wrapper     | $O(|O_f| + F)$              | $O(|O_f| + F)$              |

$N$ is the number of samples, $F$ is the number of initial features, $K$ is a multiplicative constant, $i$ is the number of iterations in the case of iterative algorithms, $C$ is the number of classes, $O_f$ is classifier complexity (in wrappers), $O_{AE}$ is auto-encoder complexity in the L2X method, and $M$ is the number of predefined selected features.

Algorithm 2. E2E-FS-Ranking. It Outputs the Importance of Each Feature (the Higher the Value in heatmap, the Better).

By Default, the Hyperparameters are Set to $(T = 2e4, \text{warmup} = 2e3, th = 0.1, \alpha_M = 0.99, \tau = 4)$

Input: $X, Y, f, T, \text{warmup}, th, \alpha_M, \tau$

1: $t, \alpha_f' \leftarrow 0$
2: $h_{\text{min}} \leftarrow (T - 1)/T$
3: $M_{f_{T_{\tau}}} \leftarrow F$
4: $\vec{y} \leftarrow \text{ones}(F)$
5: $\text{heatmap} \leftarrow \text{ones}(F)$
6: while $L_{f_{T_{\tau}}} > 0$ do
7: $X_0, Y_0 \leftarrow \text{batch from } X, Y$
8: $Y_{\text{pred}} \leftarrow f(\vec{y} \circ X_0)$
9: Train $f$ with $L_{f_{T_{\tau}}}(Y_{\text{pred}}, Y_0)$
10: $\frac{\partial L_{f_{T_{\tau}}}}{\partial \vec{y}} \leftarrow \beta_{\text{th}}((1 - \alpha)(\frac{\partial L_{f_{T_{\tau}}}(Y_{\text{pred}}, X_0)}{\partial \vec{y}}) + \alpha \frac{\partial L_{f_{T_{\tau}}}}{\partial \vec{y}}))$
11: Update $\vec{y}$ with $\frac{\partial L_{f_{T_{\tau}}}}{\partial \vec{y}}$
12: $\vec{y} \leftarrow \min(1, \max(0, \vec{y}))$
13: $t \leftarrow t + 1$
14: $\text{heatmap} \leftarrow (1 - h_{\text{min}}) \text{heatmap} + h_{\text{min}} \text{sign}(\vec{y})$
15: $\alpha_f' \leftarrow \min(\alpha_f, \max(0, (1 - \text{warmup})/T))$
16: $M_{f_{T_{\tau}}} \leftarrow \max(1, \text{pow}(F, 1 - \tau * \max(0, (1 - \text{warmup})/T)))$
17: end while
Output: $\text{heatmap}$

Essentially, the algorithm works like an RFE approach. It starts with a mask full of 1’s. During training, the algorithm eliminates irrelevant features while keeping the most important features, and stops when the number of non-zero elements of the mask is equal to or lower than the selected maximum number of features. Therefore, the larger the number of features, the faster the algorithm.

We found that, when using $\rho = 0$, the algorithm sometimes got stuck with more features than desired, despite the fact that the mask update is totally driven by the regularization terms $\alpha \approx 1$. To solve this problem, we gradually reduced $M_{f_{T_{\tau}}}$ until requirements were met. Using this trick, the approach never gets stuck and no other control has to be made.

Two default configurations are E2E-FS, with $\rho = 0$, and E2E-FS-Soft, with $\rho = 0.75$. All the other parameters are set by default.

3.3 E2E-FS-Ranking Algorithm

As depicted in Algorithm 1, this approach successfully returns a binary vector with the $M$ most important features.

However, if we want to change the $M$ value, we need to run the algorithm again. Fortunately, this algorithm can be described as an RFE algorithm, as the sign function included in $\vec{y}$ gradient definition (Eq. (5)) guarantees that discarded values are never important.

For this reason, E2E-FS can be easily converted in a ranking method merely by forcing $M = 1$, and controlling the order in which features are discarded. However, experimental results show that some features may be prematurely discarded because of the use of this low $M$ value. To prevent this we define a moving $M$ value, called $M_{f_{T_{\tau}}}$, defined as

$$M_{f_{T_{\tau}}} = F^{1 - \min(T: \tau)/T},$$

where the $\tau$ parameter controls the decay speed w.r.t $T$. This is done because to ensure that $M_{f_{T_{\tau}}} = 1$ long before the $\alpha$ parameter (controlled by $T$) becomes too high. By default, we set $\tau = 4$. Thus, when $M_{f_{T_{\tau}}} = 1$ we can easily demonstrate that $\alpha = 0.25$.

This approach, called E2E-FS-Ranking, is described in Algorithm 2.

4 EXPERIMENTAL RESULTS

To test our proposed algorithms, we carried out a series of experiments for two different scenarios: spatially and non-spatially correlated data. To do so, we selected different datasets in terms of number of samples and features. For the spatially correlated data we used classic image datasets. For the non-spatially correlated data, we evaluated three type of datasets: microarrays, datasets artificially modified for FS challenges [20], [21], and image datasets. Table 2 shows details of the datasets used.

Our algorithm was implemented using the Keras framework [22], in such a way that just a single line of code was needed. All methods used to test our methodology are implemented or accessible via Python scripts. Table 1 summarizes all the methods used for the experiments, with details of their time and memory complexity. We tried to select different types of methods to make our comparison as fair as possible.
From the filter family, we chose representatives of information-theoretic methods (MIM\(^4\) [21]), graph-based algorithms (InfFS\(^5\) [4], [5] and ILFS\(^3\) [6]), statistical measures (Fisher\(^4\) [18]) and methods based on distances (ReliefF\(^3\) [3]). Of the embedded methods, we again included a representative of information theory (L2X\(^6\) [14]) and methods based on deep neural networks architectures (CAE\(^7\) [13] and DFS\(^9\) [19]). Finally, as wrappers we selected two of the most popular methods (LASSO\(^8\) [17] and SVM-RFE\(^9\) [9]) and our own previous work based on saliencies (SFS\[^10\]).

Although one of the uses of L2X is to perform classic FS, its main purpose is to create an instance-wise FS algorithm that selects, for each input, its most important features. To that end, we show the L2X results in both scenarios, as we also wish to check the behavior of our algorithm compared with other explainable techniques. In order to ensure a fair competition between all FS methods, we used a default configuration for all of them. Thus, we set InfFS and ILFS parameters without any cross-validation (\(\alpha = 0.5\) for the InfFS and \(T = 6\) for the ILFS). For the RFE approaches (LASSO-RFE and SVM-RFE), we removed, for each iteration, 0.1% of the total features. In the case of the L2X algorithm, we attached the auto-encoder-like architecture provided by the authors, consisting of 2 hidden layers with 100 units each. The temperature parameter was also set by those authors to 0.1.

### Non-Spatially Correlated Datasets

To train our model we created a naive SVM in Keras, consisting of a neural network with no hidden units and square-hinge as the loss function using weight balance. \(l2\)-norm regularization, applied to the model’s weights, is set to \(100/N\), where \(N\) is the number of samples in the training set. We trained the model for 150 epochs, using the Adam optimizer with a learning rate of \(1e - 2\) and dividing its value by 5 after every 50 epochs. The batch size was set to \(\max(2, N/50)\).

To force convergence in both the L2X [14] and the CAE [13] methods, we added 200 and 800 epochs, respectively, at the beginning of training (the learning rate remained fixed to its initial value). All the hyperparameters in our three algorithms, E2E-FS, E2E-FS-Soft and E2E-FS-Ranking, are set by default. We ran both Algorithms 1 and 2 until the binary mask was created, using the Adam optimizer with a learning rate of \(1e - 2\) and \(\beta_1 = 0.9\) (\(\gamma\) is always trained with its default \(\beta_1 = 0.5\) parameter).

As data normalization we used the function

\[
\bar{X} = \text{erf}\left(\frac{X - \mu_t}{2\sigma_t}\right)
\]

where \(\mu_t\) and \(\sigma_t\) are the training set feature-wise mean and sample standard deviation; and \(\text{erf}(\cdot)\) is the Gauss error function. To test our algorithm, we followed the same procedure reported in [4]: we applied stratified 3-fold cross validation to the complete dataset, testing the algorithms against both our network as previously described and a linear SVM, for which the parameter \(C\) was chosen by performing a grid search over the training dataset (using a 5-fold partition).

As some datasets have unbalanced data, we used two different measures: balance accuracy (BA) when using only 10 features, and area under the curve for balance accuracy (AUC-BA), for which we averaged performance obtained with the first 10, 50, 100, 150, and 200 selected features. As L2X, CAE, and our methods cannot be directly used with a linear SVM model trained with the SMO algorithm, we report the results of three different configurations: the first two configurations (Naive and SVM) trained the classifier with the selected subset of features, while the third configuration (NaiveF) performed selection and training at the same time (an option only available for the embedded methods CAE, L2X, and our E2E-FS variants). A non-parametric statistical analysis was conducted to select which algorithms obtained the most statistically significant best results. Specifically, a Friedman test with \(\alpha = 0.05\) was used to determine if the distributions of the results of each FS algorithm were similar. If the Friedman test indicated dissimilar results, then the Neményi post-hoc analysis (also for \(\alpha = 0.05\)) was used to determine which algorithms were not significantly different from the method that achieved the highest results.

### 4.1 Non-Spatially Correlated Datasets

From the filter family, we chose representatives of information-theoretic methods (MIM\(^4\) [21]), graph-based algorithms (InfFS\(^5\) [4], [5] and ILFS\(^3\) [6]), statistical measures (Fisher\(^4\) [18]) and methods based on distances (ReliefF\(^3\) [3]). Of the embedded methods, we again included a representative of information theory (L2X\(^6\) [14]) and methods based on deep neural networks architectures (CAE\(^7\) [13] and DFS\(^9\) [19]). Finally, as wrappers we selected two of the most popular methods (LASSO\(^8\) [17] and SVM-RFE\(^9\) [9]) and our own previous work based on saliencies (SFS\[^10\]).

Although one of the uses of L2X is to perform classic FS, its main purpose is to create an instance-wise FS algorithm that selects, for each input, its most important features. To that end, we show the L2X results in both scenarios, as we also wish to check the behavior of our algorithm compared with other explainable techniques. In order to ensure a fair competition between all FS methods, we used a default configuration for all of them. Thus, we set InfFS and ILFS parameters without any cross-validation (\(\alpha = 0.5\) for the InfFS and \(T = 6\) for the ILFS). For the RFE approaches (LASSO-RFE and SVM-RFE), we removed, for each iteration, 0.1% of the total features. In the case of the L2X algorithm, we attached the auto-encoder-like architecture provided by the authors, consisting of 2 hidden layers with 100 units each. The temperature parameter was also set by those authors to 0.1.
TABLE 3
Microarray and FS Challenge BA Results When Using the Best 10 Features

|          | LYNPHOMA | COLON | LEUKEMIA | LUNG | DEXTER | GINA | GISETTE |
|----------|----------|-------|----------|------|--------|------|---------|
| NO FS    | Naive    | SVM   | Naive    | SVM  | SVM    | SVM  | SVM     |
|          | 89.48 ± 8.75 | 77.93 ± 9.52 | 93.11 ± 7.26 | 97.04 ± 4.49 | 92.97 ± 1.47 | 86.28 ± 0.88 | 97.10 ± 0.26 |
| LASSO-RFE| Naive    | SVM   | Naive    | SVM  | SVM    | SVM  | SVM     |
|          | 47.99 ± 5.98 | 74.22 ± 9.20 | 89.92 ± 5.98 | 59.29 ± 6.39 | 51.42 ± 2.57 | 61.58 ± 1.24 | 58.60 ± 1.05 |
| SVM-RFE  | 48.87 ± 10.73 | 77.72 ± 7.89 | 92.81 ± 6.12 | 56.38 ± 7.12 | 51.58 ± 2.53 | 61.35 ± 1.25 | 58.74 ± 1.05 |
| SVM      | 63.24 ± 13.97 | 63.70 ± 13.88 | 76.03 ± 11.26 | 73.78 ± 10.90 | 51.10 ± 2.96 | 63.50 ± 5.20 | 60.48 ± 7.69 |
| MIM      | 90.28 ± 5.89 | 76.72 ± 10.58 | 97.39 ± 3.46 | 97.73 ± 2.78 | 79.36 ± 3.96 | 78.28 ± 1.25 | 87.66 ± 0.80 |
| FISHER   | Naive    | SVM   | Naive    | SVM  | SVM    | SVM  | SVM     |
|          | 92.19 ± 6.20 | 82.84 ± 7.41 | 97.67 ± 3.75 | 97.30 ± 2.84 | 81.03 ± 2.58 | 78.31 ± 1.39 | 87.79 ± 0.63 |
| RELIEF   | Naive    | SVM   | Naive    | SVM  | SVM    | SVM  | SVM     |
|          | 91.58 ± 5.87 | 81.59 ± 7.54 | 94.63 ± 4.25 | 95.90 ± 3.46 | 81.42 ± 3.21 | 77.05 ± 1.32 | 85.55 ± 1.08 |
| ILFS     | SVM      | Naive | SVM      | Naive | Naive | SVM  | SVM     |
|          | 79.73 ± 8.62 | 92.18 ± 5.68 | 91.56 ± 5.80 | 82.14 ± 17.88 | 50.12 ± 3.14 | 50.17 ± 1.01 | 71.36 ± 14.91 |
| CAE      | NaiveF   | NaiveF| NaiveF   | Naive | Naive | SVM  | SVM     |
|          | 80.62 ± 10.21 | 63.04 ± 10.06 | 92.17 ± 5.07 | 88.15 ± 10.03 | 78.58 ± 2.87 | 81.30 ± 1.32 | 90.67 ± 0.81 |
| SVM      | 86.07 ± 8.72 | 76.37 ± 11.56 | 96.77 ± 3.27 | 98.26 ± 2.00 | 81.83 ± 2.76 | 81.53 ± 1.28 | 90.65 ± 0.77 |
| L2X      | NaiveF   | NaiveF| NaiveF   | Naive | Naive | SVM  | SVM     |
|          | 54.58 ± 9.90 | 67.44 ± 10.30 | 75.96 ± 15.14 | 54.98 ± 9.40 | 52.69 ± 5.91 | 83.50 ± 2.13 | 89.96 ± 1.89 |
| SVM      | 63.30 ± 12.75 | 69.35 ± 10.82 | 85.52 ± 9.10 | 79.30 ± 11.66 | 56.55 ± 6.77 | 79.70 ± 1.78 | 86.78 ± 1.12 |
| DFS      | Naive    | SVM   | Naive    | SVM  | SVM    | SVM  | SVM     |
|          | 90.31 ± 5.66 | 82.56 ± 8.45 | 96.80 ± 3.13 | 96.25 ± 3.32 | 89.89 ± 2.26 | 80.72 ± 1.81 | 90.63 ± 1.08 |
| SFS      | SVM      | Naive | SVM      | Naive | Naive | SVM  | SVM     |
|          | 91.51 ± 6.34 | 78.73 ± 9.49 | 96.56 ± 3.88 | 97.12 ± 2.72 | 88.38 ± 2.21 | 77.05 ± 3.15 | 88.58 ± 2.17 |
| E2E-FS   | NaiveF   | NaiveF| NaiveF   | Naive | Naive | SVM  | SVM     |
|          | 92.25 ± 5.97 | 83.61 ± 7.93 | 97.14 ± 3.74 | 98.14 ± 2.08 | 90.07 ± 1.63 | 83.08 ± 1.09 | 92.04 ± 0.56 |
| SVM      | 92.14 ± 5.89 | 83.67 ± 7.99 | 97.25 ± 3.64 | 98.12 ± 2.08 | 90.02 ± 1.61 | 83.09 ± 1.06 | 92.04 ± 0.55 |
| E2E-FS-Soft| NaiveF   | NaiveF| NaiveF   | Naive | Naive | SVM  | SVM     |
|          | 92.38 ± 6.27 | 83.14 ± 7.91 | 97.25 ± 3.89 | 98.00 ± 2.10 | 90.03 ± 1.63 | 83.28 ± 1.03 | 92.05 ± 0.65 |
| SVM      | 92.38 ± 6.97 | 83.36 ± 7.68 | 97.25 ± 3.89 | 97.98 ± 2.11 | 90.01 ± 1.62 | 83.28 ± 1.01 | 92.06 ± 0.65 |
| E2E-FS-Ranking| Naive | Naive | Naive | SVM  | SVM     |
|          | 91.98 ± 5.78 | 83.61 ± 7.12 | 97.44 ± 3.64 | 97.73 ± 2.83 | 89.98 ± 1.72 | 83.33 ± 0.98 | 92.24 ± 0.63 |
| SVM      | 91.37 ± 6.21 | 83.05 ± 7.49 | 96.99 ± 2.18 | 97.22 ± 2.09 | 90.47 ± 1.78 | 82.29 ± 1.08 | 92.25 ± 0.60 |

The same dataset splits were performed for each FS method (stratified 3-fold, 20 splits). NaiveF means our Naive network started with all features. Naive is the same network but with previously selected features (selected by the NaiveF model for CAE, L2X and our methods). Indicated in bold face are the best methods using a Friedman test and a pairwise Nemeyti test, both with α = 0.05. The datasets are ordered by the number of samples. The first row shows results when all features are used.

achieved very good results with 100 or more features, performance below 100 dropped significantly because the algorithm assigned the highest score to a huge number of features (between 20 to 80, depending on the dataset) and subsequently could not distinguish between them. The poor results for L2X were caused by the lack of regularization terms over the matrix mask, causing the matrix not to converge to the expected binary behavior. In contrast, in the CAE algorithm the moving temperature parameter acted as a soft regularization term, allowing it to achieve better results than L2X. However, as can seen in Table 3, our proposed algorithms achieved the best performances for all datasets. It is specially notable that our algorithms obtained better results when only using 10 features than when using all variables.

The fact that our algorithm constantly removed the least important features made its behavior outstanding when compared to both SVM-RFE and LASSO-RFE.

Regarding computational cost, Figs. 2a and 2b show the time needed to obtain a specific number of selected features. Unsurprisingly, Fisher was the fastest as it requires far less computation time. Classic techniques like MIM or ReliefF were extremely fast because of the low number of samples per each dataset. Comparing our technique to L2X and CAE (the two that were closest in terms of definition), CAE was systematically the slowest algorithm. Although this is partially explained by our conservative approach of using 800 epochs to train it, it is also worth noting that, even with that conservative approach, good convergence was not achieved in some configurations, like when testing for the Lymphoma or Colon datasets with only 10 features. The same applies to the L2X algorithm, which was slightly faster than our E2E-FS-Ranking algorithm. This is a remarkable result, since E2E-FS-Ranking only needs to be computed once, while L2X (and also CAE, E2E-FS, and E2E-FS-Soft) need to be computed each time we change the desired number of features.
with the image datasets: as the number of selected features increased, so too did the computational cost for both the L2X and CAE algorithms, due to the need for the $R^{F \times F}$ matrix to be trained in both algorithms. In contrast, as our algorithm only used a vector of size $R^F$, a higher number of selected features $M$ did not lead to an increase in computational time, which was sometimes even reduced, especially when using the E2E-FS-Soft algorithm. As the nature of our algorithms is to recursively remove features, the fewer features they need to remove, the faster they converge. Compared to the classic approaches, the computational times of our algorithms were extremely competitive, especially for E2E-FS-Soft, which ranked second faster for both the Dexter and Gisette datasets. It is also interesting to note that LASSO-RFE performance time increased significantly, due to a problem of algorithm convergence in the Scikit implementation, which meant that the algorithm was forced to run until the maximum number of steps was reached.

### 4.1.2 FS Challenge Datasets

We also tested our algorithm against challenging datasets that contain distractor features. The last three columns of Tables 3 and 4 show results for the same configurations as in the microarray datasets experiments. For higher numbers of samples, our proposed models achieved the best results for all datasets. Note that we excluded the MADEON dataset from this table, as it cannot be correctly classified using a linear classifier; furthermore, it is known that only 5 features are relevant, containing another 15 features that are linear combinations of those 5 relevant features, so it is pointless to evaluate its performance for more features. Using only 10 features, our algorithm obtained lower scores than those achieved without feature removal, although the accuracy drop was less than 5%. However, comparing our AUC-BA results over the range of 10-200 most important features (Table 4), our E2E-FS variants obtained the same results, except for the GINA dataset, for which accuracy dropped by 2%.

Regarding computational time, Figs. 2c and d show an interesting effect that will be dramatically exposed later
To test the Madelon dataset, we performed an experiment using a dense network with 3 hidden layers (50, 25 and 10 units). Batch Normalization [30] and the ReLU function were used in each layer, with a weight decay of $1e^{-3}$. All the other configurations were as in the previous tests. Table 5 shows results on varying the number of selected features. Remarkable were the big differences in performance when the number of features was extremely low, pointing to interesting behavior. As mentioned above, this dataset contains only 5 relevant features along with 15 linear combinations of those 5 relevant features. Our algorithms obtained their best results using only 5 features, suggesting that, whenever the classifier is not carefully chosen, the algorithm tends to remove redundant information before noisy features.

4.2 Spatially Correlated Datasets
We also tested how our algorithms tackled more complex classifiers like CNNs, performing the same experiment as described in [10], based on four datasets (MNIST, Fashion-MNIST, CIFAR-10 and CIFAR-100), and three networks (the
TABLE 5

| Features | MADELON |
|----------|---------|
|          | 5       | 10     | 15     | 20     |
| LASSO-RFE Naive | 49.12  | 49.78  | 50.17  | 54.89  |
|            | ±1.63  | ±1.87  | ±1.64  | ±1.62  |
| SVM-RFE Naive | 50.29  | 51.03  | 51.12  | 50.95  |
|            | ±2.29  | ±2.57  | ±2.53  | ±2.11  |
| MIM Naive | 63.51  | 63.46  | 62.34  | 61.33  |
|            | ±6.17  | ±5.45  | ±4.54  | ±3.76  |
| FISHER Naive | 69.52  | 87.21  | 85.62  | 80.46  |
|            | ±3.14  | ±1.79  | ±1.62  | ±1.64  |
| RELIEFF Naive | 74.86  | 87.88  | 89.20  | 89.41  |
|            | ±3.49  | ±1.78  | ±1.26  | ±1.04  |
| InfFS Naive | 66.89  | 76.09  | 79.44  | 77.99  |
|            | ±9.87  | ±9.37  | ±7.87  | ±5.96  |
| ILFS Naive | 58.37  | 62.28  | 68.44  | 72.96  |
|            | ±1.31  | ±7.17  | ±4.18  | ±3.84  |
| CAE NaiveF | 74.86  | 79.09  | 80.17  | 79.57  |
|            | ±4.32  | ±2.34  | ±2.28  | ±2.30  |
| Naive | 78.10  | 80.42  | 78.56  | 76.20  |
|            | ±4.85  | ±3.43  | ±2.20  | ±3.30  |
| L2X NaiveF | 59.67  | 56.86  | 56.31  | 56.39  |
|            | ±4.89  | ±6.20  | ±5.85  | ±6.05  |
| Naive | 61.97  | 62.88  | 59.76  | 62.78  |
|            | ±5.17  | ±8.22  | ±6.72  | ±7.83  |
| DFS Naive | 88.20  | 85.52  | 81.70  | 78.77  |
|            | ±3.17  | ±2.11  | ±2.00  | ±2.01  |
| SFS Naive | 64.82  | 72.31  | 76.82  | 77.70  |
|            | ±4.66  | ±9.12  | ±8.36  | ±6.14  |
| E2S-FS NaiveF | 88.51  | 84.40  | 80.97  | 77.93  |
|            | ±1.61  | ±1.81  | ±1.47  | ±1.56  |
| Naive | 88.92  | 84.11  | 80.43  | 77.26  |
|            | ±1.83  | ±2.06  | ±1.58  | ±1.63  |
| E2S-FS-Soft NaiveF | 88.56  | 86.56  | 83.05  | 79.28  |
|            | ±1.09  | ±2.40  | ±1.46  | ±1.77  |
| Naive | 89.12  | 86.31  | 82.89  | 78.06  |
|            | ±1.15  | ±2.75  | ±1.62  | ±2.15  |
| E2S-FS-Ranking Naive | 88.96  | 85.30  | 80.90  | 77.79  |
|            | ±1.69  | ±1.42  | ±1.65  | ±1.72  |

Table 5: Madelon Accuracy Results for Different Numbers of Selected Features (second row), Using a Fully-Connected Network as the Classifier.

Indicated in bold face are best methods using a Friedman test and a pairwise Nemenyi test, both with α = 0.05. When all features are used accuracy is 60.76 ± 1.84.

Wide Residual Network [31] WRN-16-4; a DenseNet version specially designed for the CIFAR-10 dataset [32]; and the EfficientNetB0 [33]. Testing was for five different approaches: the Deep Feature Selection (DFS) method [19], which is a variant of the LASSO algorithm, specially designed to be used in deep-learning architectures; the SFS algorithm and its iterative version called iSFS; a combination of both DFS and SFS; and the L2X. CAE was not tested as it can only be used with fully-connected networks. In this experiment, we followed the predefined dataset partitions. As our algorithms’ hyperparameters, we again used the defaults. After the mask was correctly obtained, we continued training the model for another 60 epochs with all features. To do that, we used the SGD optimizer, fixing the learning rate to 0.1 during mask generation, and then dividing its value by 5 after 30 and 50 epochs; we also used \(l_2\)-norm regularization for the weights, set at \(5\varepsilon - 4\). As we were dealing with CNNs, \(\mathbf{y}\) needed to be reshaped to the image input size in \(L_f\). To train the L2X model, we used the same auto-encoder-like network described above, and trained the model for 110 epochs. Starting the learning rate at 0.1, this was reduced by 5 after 60 and 90 epochs. In the case of the DenseNet architecture, the RMSprop optimizer was used. The learning rate schedule was the same as for the SGD version, but starting at 0.01.

Table 6 shows results when no data transfer was used (FS performed over the same network used for classification) for the WRN-16-4 datasets (the results for the DenseNet and EfficientNetB0 architectures can be found in the Appendix), available online. The E2E-FS-Soft algorithm achieved the best results for almost all datasets and configurations. A possible explanation for the poor performance of the E2E-FS algorithm for the MNIST dataset may be that it did not properly behave against features with zero deviation (they always had the same value). Even so, accuracy overall was close to the state-of-the-art. Accuracy was significantly higher for the other datasets. Again, a huge improvement was obtained when the number of selected features was low (with 153 variables, performance increased for CIFAR-10 and 100 by 18% and 41%, respectively). Note that our algorithms, using only 25% of the features, obtained almost the same results as those of the same network without using FS techniques. Furthermore, accuracy was increased by a short margin on using 50% of the features.

Another interesting effect was related to the disparity in results obtained on comparing E2E-FS-Soft and E2E-FS-Ranking for both CIFAR-10 and CIFAR-100. On examining the selected features, we discovered that both algorithms selected almost the same features. The only difference was the network initialization: while E2E-FS-Soft continued training after mask generation, using the network configuration obtained right after Algorithm 1 stopped, E2E-FS-Ranking required the network to fully start over, as the network state after the end of Algorithm 2 did not have a direct relation to the number of features we may select. This clearly suggests that network initialization was a relevant factor for ultimate accuracy whenever dataset complexity is high. Although L2X may benefit from the same effect as mentioned above, in practice we found that the increase in accuracy was marginal. We believe that this reflects the nature of the approaches: while we forced our mask to be strictly binary, both L2X and CAE only required the matrix to be close to binary, and we think that these small values included in their solution affected the final result. Regarding computational cost, both iSFS and L2X were highly dependent on the number of selected features, whereas our methods proved invariant to that issue.

4.3 Effect of Transfer Learning

The experimental study results described in [10] clearly suggest that it is very difficult to obtain a good set of features when using a complex network like the WRN. In contrast, using a much simpler feature ranking method, and using this ranking to train the complex network, leads to a huge increase in accuracy values.

We tested our algorithms’ performance when different architectures were used for FS and classification. As the FS...
algorithm, we used the same neural network as used for the Madelon dataset (fully-connected network with three hidden layers with 50, 25 and 10 units). Used for the classification task were the biggest convolutional architectures. We trained the L2X and CAE algorithms for 80 epochs, while our algorithm was trained until convergence (it required close to 20 epochs on average). Table 7 shows the results for the DenseNet and EfficientNetB0 architectures (WRN-16-4 results can be found in the Appendix), available in the online supplemental material. Again, our proposed E2E-FS-Soft and E2E-FS-Ranking algorithms obtained state-of-the-art results for every configuration. It is also worth noting that the gap, compared with the results shown in Table 3, was reduced.

The accuracy of our approaches was lower than the accuracy for simultaneous FS and classification. This confirms that model initialization plays an important role in the results in the latter case. Note that, compared to CAE and L2X, our algorithm was not only faster, but also was the only algorithm that guaranteed convergence in binary mask creation. Both CAE and L2X obtained values that were close to the binary mask; however, results that were close to zero were maximized in the classification network because of the Batch Normalization layers, nullifying the effect of the mask. Our method, in contrast, in forcing the low values to be zero, was able to neutralize this problem, with the outcome that our results were good regardless of the architecture configuration used.

The accuracy of our approaches was lower than the accuracy for simultaneous FS and classification. This confirms that model initialization plays an important role in the results in the latter case. Note that, compared to CAE and L2X, our algorithm was not only faster, but also was the only algorithm that guaranteed convergence in binary mask creation. Both CAE and L2X obtained values that were close to the binary mask; however, results that were close to zero were maximized in the classification network because of the Batch Normalization layers, nullifying the effect of the mask. Our method, in contrast, in forcing the low values to be zero, was able to neutralize this problem, with the outcome that our results were good regardless of the architecture configuration used.

algorithm, we used the same neural network as used for the Madelon dataset (fully-connected network with three hidden layers with 50, 25 and 10 units). Used for the classification task were the biggest convolutional architectures. We trained the L2X and CAE algorithms for 80 epochs, while our algorithm was trained until convergence (it required close to 20 epochs on average). Table 7 shows the results for the DenseNet and EfficientNetB0 architectures (WRN-16-4 results can be found in the Appendix), available in the online supplemental material. Again, our proposed E2E-FS-Soft and E2E-FS-Ranking algorithms obtained state-of-the-art results for every configuration. It is also worth noting that the gap, compared with the results shown in Table 3, was reduced.

The accuracy of our approaches was lower than the accuracy for simultaneous FS and classification. This confirms that model initialization plays an important role in the results in the latter case. Note that, compared to CAE and L2X, our algorithm was not only faster, but also was the only algorithm that guaranteed convergence in binary mask creation. Both CAE and L2X obtained values that were close to the binary mask; however, results that were close to zero were maximized in the classification network because of the Batch Normalization layers, nullifying the effect of the mask. Our method, in contrast, in forcing the low values to be zero, was able to neutralize this problem, with the outcome that our results were good regardless of the architecture configuration used.

The accuracy of our approaches was lower than the accuracy for simultaneous FS and classification. This confirms that model initialization plays an important role in the results in the latter case. Note that, compared to CAE and L2X, our algorithm was not only faster, but also was the only algorithm that guaranteed convergence in binary mask creation. Both CAE and L2X obtained values that were close to the binary mask; however, results that were close to zero were maximized in the classification network because of the Batch Normalization layers, nullifying the effect of the mask. Our method, in contrast, in forcing the low values to be zero, was able to neutralize this problem, with the outcome that our results were good regardless of the architecture configuration used.

algorithm, we used the same neural network as used for the Madelon dataset (fully-connected network with three hidden layers with 50, 25 and 10 units). Used for the classification task were the biggest convolutional architectures. We trained the L2X and CAE algorithms for 80 epochs, while our algorithm was trained until convergence (it required close to 20 epochs on average). Table 7 shows the results for the DenseNet and EfficientNetB0 architectures (WRN-16-4 results can be found in the Appendix), available in the online supplemental material. Again, our proposed E2E-FS-Soft and E2E-FS-Ranking algorithms obtained state-of-the-art results for every configuration. It is also worth noting that the gap, compared with the results shown in Table 3, was reduced.

The accuracy of our approaches was lower than the accuracy for simultaneous FS and classification. This confirms that model initialization plays an important role in the results in the latter case. Note that, compared to CAE and L2X, our algorithm was not only faster, but also was the only algorithm that guaranteed convergence in binary mask creation. Both CAE and L2X obtained values that were close to the binary mask; however, results that were close to zero were maximized in the classification network because of the Batch Normalization layers, nullifying the effect of the mask. Our method, in contrast, in forcing the low values to be zero, was able to neutralize this problem, with the outcome that our results were good regardless of the architecture configuration used.
5 Conclusion

We have described a novel FS method that can be attached to any model trained using gradient descent techniques. We propose our algorithm as a general optimization problem, with an approximation that can be used to efficiently solve it. The Python implementation allows for easy implementation, as just a single line of code is needed. The experimental results show that our proposal, using only default parameters, achieves state-of-the-art or better results for a wide variety of datasets (from microarray to larger image datasets), using different classifiers (SVMs, dense NNs and CNNs) and optimizers (Adam and SGD).

As future work, several research paths are possible, including developing different approaches that can solve the E2E-FS optimization problem, and evaluating the possibility of using E2E-FS as a classic regularization technique, rather than forcing it to select a specific number of features – an approach that would involve the combination of our optimization techniques with algorithms for selecting the optimal number of features [34], [35]. Two further research lines could be to extend this algorithm to the instance-based domain, in a similar way as proposed for the L2X algorithm, and to transform the algorithm to be used in similar fields, for instance, filter pruning [36], [37].

Acknowledgments

We also gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research.

References

[1] I. Guyon, S. Gunn, M. Nikravesh, and L. A. Zadeh, Feature Extraction: Foundations and Applications, vol. 207. Berlin, Germany: Springer, 2008.
[2] B. C. Ross, “Mutual information between discrete and continuous data sets,” PLoS One, vol. 9, no. 2, 2014, Art. no. e97357.
[3] I. Kononenko, E. Simec, and M. Robnik-Sikonja, “Overcoming the myopia of inductive learning algorithms with relief,” Appl. Intell., vol. 7, no. 1, pp. 39–55, 1997.
[4] G. Roffo, S. Melzi, and M. Cristiani, “Infinite feature selection,” in Proc. IEEE Int. Conf. Comput. Vis., 2015, pp. 4202–4210.
[5] G. Roffo et al., “Infinite feature selection: A graph-based feature filtering approach,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 43, no. 12, pp. 4396–4410, Dec. 2021.
[6] G. Roffo, S. Melzi, U. Castellani, and A. Vinciarelli, “Infinite latent feature selection: A probabilistic latent graph-based ranking approach,” in Proc. IEEE Int. Conf. Comput. Vis., 2017, pp. 1398–1406.
[7] V. Bolón-Canedo, N. Sánchez-Maroño, and A. Alonso-Betanzos, “A review of feature selection methods on synthetic data,” Knowl. Inf. Syst., vol. 34, no. 3, pp. 483–519, 2013.
[8] M. Mejía-Lavalle, E. Sucar, and G. Arroyo, “Feature selection with a perception neural net,” in Proc. Int. Workshop Feature Selection Data Mining, 2006, pp. 131–135.
[9] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, “Gene selection for cancer classification using support vector machines,” Mach. Learn., vol. 46, no. 1/3, pp. 389–422, 2002.
[10] B. Cancela, V. Bolón-Canedo, A. Alonso-Betanzos, and J. Gama, “A scalable saliency-based feature selection method with instance-level information,” Knowl.-Based Syst., vol. 192, 2020, Art. no. 105326.
[11] K. Simonyan, A. Vedaldi, and A. Zisserman, “Deep inside convolutional networks: Visualising image classification models and saliency maps,” 2013, arXiv:1312.6034.
[12] R. Tibshirani, “Regression shrinkage and selection via the lasso,” J. Roy. Stat. Soc.: Ser. B (Methodol.), vol. 58, no. 1, pp. 267–288, 1996.
[13] M. F. Balin, A. Abid, and J. Y. Zou, “Concrete autoencoders: Differentiable feature selection and reconstruction,” in Proc. Int. Conf. Mach. Learn., 2019, pp. 444–453.
[14] J. Chen, L. Song, M. Wainwright, and M. Jordan, “Learning to explain: An information-theoretic perspective on model interpretability,” in Proc. Int. Conf. Mach. Learn., 2018, pp. 853–861.
[15] P. Yin, Y. Lou, Q. He, and J. Xin, “Minimization of l1, 2 for compressed sensing,” SIAM J. Sci. Comput., vol. 37, no. 1, pp. A536–A563, 2015.
[16] J. Luu, S. Zhang, Y. Qi, and J. Xin, “AutoShuffleNet: Learning permutation matrices via an exact lipschitz continuous penalty in deep convolutional neural networks,” 2019, arXiv:1901.08624.
[17] F. Santosa and W. W. Symes, “Linear inversion of band-limited reflection seismograms,” SIAM J. Sci. Stat. Comput., vol. 7, no. 4, pp. 1307–1330, 1986.
[18] X. He, D. Cai, and P. Niyogi, “Laplacian score for feature selection,” in Proc. Adv. Neural Inf. Process. Syst., 2006, pp. 507–514.
[19] Y. Li, C.-Y. Chen, and W. W. Wasserman, “Deep feature selection: Theory and application to identify enhancers and promoters,” J. Comput. Biol., vol. 23, no. 5, pp. 522–536, 2016.
[20] I. Guyon, S. Gunn, H. Ben-Hur, and G. Dror, “Result analysis of the nips 2003 feature selection challenge,” in Proc. Adv. Neural Inf. Process. Syst., 2005, pp. 545–552.
[21] I. Guyon, A. Saifari, G. Dror, and G. Cawley, “Agnostic learning vs. prior knowledge challenge,” in Proc. Int. Joint Conf. Neural Net., 2007, pp. 829–834.
[22] F. Chollet et al., “Keras,” 2015. [Online]. Available: https://keras.io
[23] T. Geulb et al., “Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring,” Science, vol. 286, no. 5439, pp. 531–537, 1999.
[24] U. Alon et al., “Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays,” Proc. Nat. Acad. Sci. USA, vol. 96, no. 12, pp. 6745–6750, 1999.
[25] G. J. Gordon et al., “Translation of microarray data into clinically relevant cancer diagnostic tests using gene expression ratios in lung cancer and mesothelioma,” Cancer Res., vol. 62, no. 17, pp. 4963–4967, 2002.
[26] I. Guyon, J. Li, T. Mader, P. A. Pletcher, G. Schneider, and M. Uhr, “Competitive baseline methods set new standards for the NIPS 2003 feature selection benchmark,” Pattern Recognit. Lett., vol. 28, no. 12, pp. 1438–1444, 2007.
[27] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” Proc. IEEE, vol. 86, no. 11, pp. 2278–2242, Nov. 1998.
[28] H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms,” 2017, arXiv:1708.07747.
[29] A. Krizhevsky et al., “Learning multiple layers of features from tiny images,” Master’s thesis, Dept. Comput. Sci., Univ. Trent, Toronto, Canada, 2009.
[30] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” 2015, arXiv:1502.03167.
[31] S. Zagoruyko and N. Komodakis, “Wide residual networks,” 2016, arXiv:1605.07146.
[32] I. Savenitza, Advanced Deep Learning With TensorFlow 2 and Keras: Apply DL, GANs, VAEs, Deep RL, Unsupervised Learning, Object Detection and Segmentation, and More. Birmingham, U.K.: Packt Publishing Ltd, 2020.
[33] M. Tan and Q. Le, “EfficientNet: Rethinking model scaling for convolutional neural networks,” in Proc. Int. Conf. Mach. Learn., 2019, pp. 6105–6114.
[34] P. Somol, P. Pudil, and J. Kittler, “Fast branch & bound algorithms for optimal feature selection,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 23, no. 5, pp. 322–336, 2001.
[35] M. H. Nguyen and F. De la Torre, “Optimal feature selection for support vector machines,” Pattern Recognit., vol. 43, no. 3, pp. 584–591, 2010.
[36] P. Molchanov, A. Malyuta, S. Tyree, I. Frosio, and J. Kautz, “Importance estimation for neural network pruning,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2019, pp. 11264–11272.
[37] Y. Li, S. Gu, K. Zhang, L. Van Goel, and R. Timofte, “DHP: Differentiable meta pruning via hypernetworks,” in Proc. Eur. Conf. Comput. Vis., 2020, pp. 608–624.
Brais Cancela received the PhD degree for his work in the area of computer vision from the University of A Coruña, Spain, in 2015. Later, he was a postdoctoral researcher with the INESC TEC, Portugal, in 2017. He is currently a postdoctoral researcher with the Department of Computer Science, University of A Coruña, Spain. His main current areas are convex optimization, machine learning and explainable models.

Verónica Bolón-Canedo received the BS, MS, and PhD degrees in computer science from the University of A Coruña, Spain, in 2009, 2010, and 2014, respectively. After a postdoctoral fellowship with the University of Manchester, U.K., in 2015, she is currently an associate professor with the Department of Computer Science, University of A Coruña. Her main current areas are machine learning and feature selection.

Amparo Alonso-Betanzos received the PhD degree for her work in the area of medical expert systems from the University of Santiago de Compostela, Spain, in 1988. Later, she was a postdoctoral fellow with the Medical College of Georgia, Augusta. She is currently a full professor with the Department of Computer Science, University of A Coruña, Spain. Her main current areas are intelligent systems, scalable machine learning and feature selection.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/csdl.