A scalable saliency-based feature selection method with instance-level information

Brais Cancela[a,b,*], Verónica Bolón-Canedo[a], Amparo Alonso-Betanzos[a], João Gama[b]

[a] *CITIC. Universidade da Coruña. 15006, A Coruña, Spain*
[b] *LIAAD, INESCTEC. Rua Dr. Roberto Frias 4200-465 Porto, Portugal*

Abstract

Classic feature selection techniques remove irrelevant or redundant features to achieve a subset of relevant features in compact models that are easier to interpret and so improve knowledge extraction. Most such techniques operate on the whole dataset, but are unable to provide the user with useful information when only instance-level information is required; in other words, classic feature selection algorithms do not identify the most relevant information in a sample. We have developed a novel feature selection method, called saliency-based feature selection (SFS), based on deep-learning saliency techniques. Our algorithm works under any architecture that is trained by using gradient descent techniques (Neural Networks, SVMs, ...), and can be used for classification or regression problems. Experimental results show our algorithm is robust, as it allows to transfer the feature ranking result between different architectures, achieving remarkable results. The versatility of our algorithm has been also demonstrated, as it can work either in big data environments as well as with small datasets.

Keywords: Feature Selection, Deep Learning, Saliency

*Corresponding author
Email address: brais.cancela@udc.es (Brais Cancela)
URL: https://www.lidiagroup.org/ (Brais Cancela)

1. Introduction

The rise of big data has been accompanied by a growing need for techniques that reduce the input space [1]. Such techniques are typically classified as [2] feature selection (FS) and feature extraction (FE) techniques. Fig. 1 shows a graphic representation about how these two approaches work. FE approaches reduce the number of characteristics by combining (linearly or otherwise) input space features [3]. An FE technique in deep learning obtains what are called deep features, i.e., data representation resulting from removal of the last neural network (NN) layer. The result is a new feature set, typically more compact and with a greater discriminant capacity. FE is mostly used for image analysis, signal processing and information retrieval [4, 5, 6]. In contrast, FS approaches achieve dimensionality reduction by removing irrelevant and redundant features [7]. Since FS techniques preserve the original features, they are especially useful in applications where those attributes are essential for understanding the model and for knowledge inference [8, 9, 10].

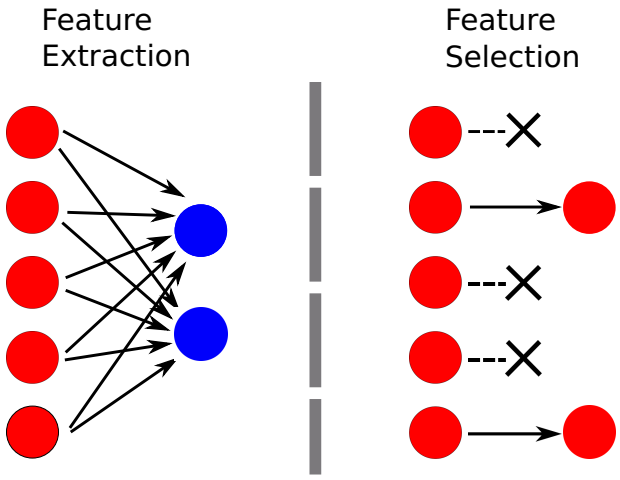

Figure 1: Feature extraction creates new features by combining elements of the original input, whereas feature selection removes features that are considered irrelevant or redundant, while retaining the remaining features.

FS techniques are frequently classified in three broad groups: filters, embedded methods and wrappers [3, 11]. Filters are independent of the inductive

model and use the general characteristics of the data as a measure of the relevance of features, e.g., mutual information among the features and correlation classes. Embedded methods and wrappers select relevant features according to the performance of a given induction model. Embedded methods select features during training of a classifier; e.g., recursive feature elimination for support vector machines (RFE-SVM) [12] chooses relevant features during support vector machine (SVM) training. Finally, wrappers use the prediction model to evaluate the relevance of subsets of features by applying a given prediction algorithm (e.g., an SVM or an NN) in combination with a search strategy. Examples are a spam detection wrapper based on evolutionary random weight networks[13] and a grasshopper optimization algorithm with evolutionary population dynamics for FS that was successfully applied to 22 UCI datasets [14]. Figure 2 summarizes the main advantages and disadvantages of the three FS approaches.

The concept underlying the three approaches is similar: drawing from the entire dataset, features that contain the most discriminant information are selected. We describe an algorithm, however, that, instead of using the information contained in the whole dataset, draws on a sample taken from the dataset to build an FS model.

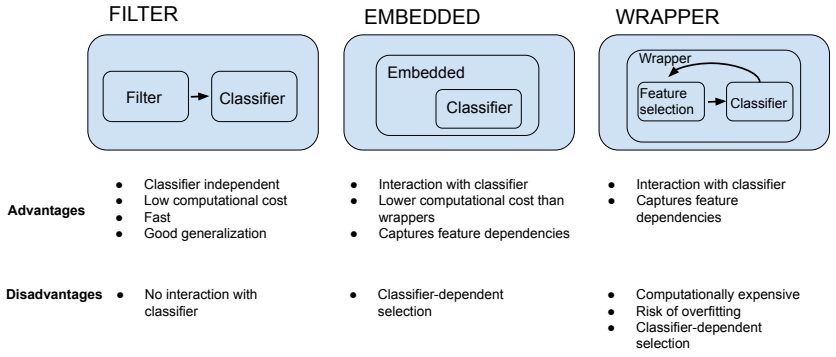

Figure 2: Classic feature selection techniques.

To illustrate, using a medical dataset, suppose we want to predict if a patient is likely to develop cancer. While classic FS algorithms select the most important

features that will help the classifier make a good prediction, in our approach, we infer the features the classifier uses to predict a certain outcome and use these to build the FS algorithm. The advantage of this approach is that it customizes feature relevance information for each sample.

In terms of model explainability, typically used is the LIME algorithm [15], which applies small perturbations to the input to establish the importance of each feature in the model's decision. Although this black-box method has the advantage that it can be used for any classifier model, it has two main drawbacks: 1) it requires too much time to evaluate each sample (up to 10 minutes in the ImageNet dataset [16], as indicated by the authors); and 2) it does not improve model explainability, as it is designed to be applied after model training.

To overcome this problem, we developed a novel FS algorithm, called saliency-based FS (SFS), designed to provide personalized information for any given example. This algorithm was created by including those features that contain a higher discrimination coefficient. Since scalability is an important requirement for any machine learning algorithm, our algorithm was designed to work in big data environments.

The most important contributions of the SFS are as follows: 1) it can be used in NNs and in any architecture trained using gradient descent techniques; 2) it is robust, as demonstrated by experimental results for challenging datasets (e.g., the NIPS 2003 Feature Selection Challenge), for which it achieves state-of-the-art performance in many different scenarios, including for datasets such as MNIST, CIFAR-10 and CIFAR-100; and 3) it provides the kind of instance-level information that is very useful for explainability purposes.

The rest of the paper is organized as follows: Section 2 describes the personalized information algorithm; Section 3 describes the metrics used to compute saliency; Section 4 describes our novel SFS algorithm; Section 5 describes an application of the SFS to an ablation study; Section 6 reports experimental results for certain public datasets, and finally, Section 7 concludes and describes future work.

2. A personalized information algorithm

A mere handful of research works address the problem of FS in big data environments. While deep Bayesian networks have been used to select relevant features [17, 8] in big data settings, this approach only functions for small datasets (with either few examples or few features). A deep FS technique has been used to reduce the input space in short-term wind forecasting models [18], but the use of recursive feature elimination (RFE) required exponentially training several different models, which means this approach cannot be used when the number of features is large. Perhaps the most interesting approach is one called deep feature selection (DFS) [19], which consists of an elastic net variant that can be introduced as an extra layer into any NN; a mask is included in the input data with l_1- and l_2-regularization, in the same way as the elastic net is defined, and elastic net penalties are applied to the hidden layers. However, according to the authors, for the method to function properly the number of layers has to be reduced, contraindicating its use with convolutional NN (CNN) models. None of those approaches is able to provide personalized information, nonetheless. To address this issue, we used a well-known computer vision technique called saliency.

2.1. Saliency

NNs are viewed as black boxes where, given any input and any desired output, it is possible to obtain an accurate prediction that is somehow close to what we should expect. However, NNs do not provide any kind of transparent explanation about how the system reaches the predicted solution. Saliency, a concept that was first developed in computer vision settings, enables viewing what is happening inside a given NN. More specifically, it is an evaluation of the quality of individual pixels in an image [20].

Semi-supervised and supervised approaches are used to calculate saliency. The earliest works used a semi-supervised approach [20, 21]; thus, for a given NN trained for classification and a given image, a back-propagation routine detects

the pixels that most influence the desired output. More recent approaches are supervised [22, 23], for which training is different. Thus, the NN has the same input and output size and it is also known what the most relevant features are in a given image; e.g., if we are detecting a cat, the important features are pixels in the image of the actual cat. The model is trained so that the predicted output matches a prior segmentation of important features. This segmentation explains why these NNs are also called semantic segmentation networks. They are alternatively called attention models [24, 25] when a recurrent NN is used at the end to evaluate the quality of features.

While supervised techniques achieve better results than semi-supervised approaches, they have the major drawback that it is necessary to know the most relevant features for each instance a priori in order to successfully train the model. Unlike what happens with image datasets, this information is not always available for other environments, e.g., DNA microarrays. Furthermore, supervised techniques can only be used for classification problems and are not suitable for other approaches, such as regression. For this reason we used a semi-supervised saliency technique.

2.2. The proposed model

For our model, we used a generalization of an idea proposed by Simonyan et al. [20]. Let $\mathbf{X} \in \mathcal{R}^{N \times R}$ be our input data, with N and R representing the number of samples and of features, respectively; and let $\tilde{\mathbf{Y}} = f(\mathbf{X}; \mathbf{\Theta}) \in \mathcal{R}^{N \times C}$ be our classification model (for the purposes of explanation only; later we explain how this approach can be applied to regression problems). Which type of model we use is irrelevant, as long as it can be trained using a loss minimization function (NN, CNN, SVM, etc). C is the number of different classes to evaluate, and $\mathbf{\Theta}$ are the classifier weights, which are adjusted during the training procedure.

For explanatory purposes, we can assume that $f(\mathbf{X}; \mathbf{\Theta})$ is the result of ap-

plying the softmax function to a one-layer model ($\Theta \in \mathcal{R}^{C \times R}$). Thus:

$$\sum_{c=1}^{C} y_c^{(i)} = 1, \tag{1}$$

where

$$y_c^{(i)} = \text{softmax}(\theta_c^T \mathbf{X}^{(i)}). \tag{2}$$

$y_c^{(i)}$ is the probability that instance i belongs to class c. θ_c is the c-th column of Θ.

To train this model, we minimize a loss function $\ell(\mathbf{\Theta}; f, \mathbf{X}, \mathbf{Y})$, where $\mathbf{Y} \in \mathcal{R}^{N \times C}$ is one-hot encoding for the class. Since we are using the softmax function as our output, our minimization function is the categorical cross-entropy, defined as

$$\ell(\mathbf{\Theta}; f, \mathbf{X}, \mathbf{Y}) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_c^{(i)} \log \Big(f(\mathbf{x}^{(i)}; \mathbf{\Theta})_c \Big) \tag{3}$$

2.2.1. Classic saliency

To know which the features most contribute to activating the class c, the solution proposed by Simonyan et al. [20] is to evaluate the gradient of $y_c^{(i)}$ with respect to the input, i.e., in mathematical terms:

$$\sigma_c^{(i)} = \left| \frac{\partial y_c^{(i)}}{\partial \mathbf{X}^{(i)}} \right|. \tag{4}$$

Intuitively, this gradient indicates how we should modify the input instance in order to maximize its belongingness to class c. Since this technique only works for classification problems, however, we need to generalize the method to make it suitable for our purposes.

2.2.2. Generalization

Instead of applying a gradient function for each class c, we use an approach that is similar to updating weights during training. As the loss function usually yields higher gradients whenever there is severe misclassification during training, we propose defining a gain function ($g(\mathbf{\Theta}; f, \mathbf{X}, \mathbf{Y})$) that will ensure

high gradients when samples are correctly classified. Thus, our saliency function σ is defined as:

$$\sigma(\tilde{y}^{(i)}, y^{(i)}) = \left| \frac{\partial g(\tilde{y}^{(i)})}{\partial \mathbf{x}^{(i)}} \right|, \tag{5}$$

where $\tilde{\mathbf{y}} = f(\mathbf{X}; \boldsymbol{\Theta})$ is our model's predicted output for instance i.

Below we describe how to create this gain function g depending on the loss function being minimized.

3. The gain function

Since our aim was to develop a saliency system that can work with multiple types of problems, we explain how to create our gain function g for three different scenarios: one for regression and two for classification (NNs and SVMs). This gain function must be defined in a way such that correctly classified samples obtain high saliency scores, with incorrect classifications indicated by values close to zero. This explains why loss functions cannot be directly applied.

3.1. Regression

We first describe the regression gain function as being more intuitive. For simplification purposes, we assume our model is trained using the mean square error (MSE) loss:

$$\ell_{MSE}(\tilde{\mathbf{Y}}, \mathbf{Y}) = -\frac{1}{N} \sum_{i=1}^{N} \left(\tilde{\mathbf{y}}^{(i)} - \mathbf{y}^{(i)} \right)^2, \tag{6}$$

where $\tilde{\mathbf{Y}} = f(\mathbf{X}; \boldsymbol{\Theta})$ is our model's predicted output. Choosing a different loss function makes no difference since all regression losses have the same structure: a 0 value if the prediction is absolutely correct, increasing in value as the prediction moves away from the expected result.

Our gain function must behave in the opposite way, however, i.e., it must yield high values for accurate predictions and values close to zero for poor predictions. Thus, our solution is to use the inverse of the MSE loss function:

$$g_{MSE}(\tilde{\mathbf{Y}}, \mathbf{Y}) = \frac{\alpha}{\ell_{MSE}(\tilde{\mathbf{Y}}, \mathbf{Y}) + \epsilon}, \tag{7}$$

where α is a multiplication factor and $\epsilon > 0$ is a factor that avoids division by zero. By default, we set $\alpha = 1$ and $\epsilon = 10^{-3}$.

3.2. Classification

Although Eq. 7 can potentially also be used as the gain function in classification problems, we found it to be less than suitable because of its behavior with total misclassifications ($\tilde{y}_c^{(i)} = 0$ and $y_c^{(i)} = 1$, for instance). Remember that since we are using the saliency function to build an FS algorithm, we do not want any information on total misclassifications; in other words, in these instances we want our gain function to return 0 values, which is a requirement not satisfied by the gain function in Eq. 7. We therefore developed two different gain functions for two different classification losses: a cross-entropy gain function and a hinge loss gain function.

3.2.1. Cross-entropy gain function

The cross-entropy loss function (Eq. 3) is frequently used with NNs for classification, including CNNs. Since our gain function should operate in reverse, our solution is as follows:

$$g_{CE}(\tilde{\mathbf{Y}}, \mathbf{Y}) = -\frac{\alpha}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_c^{(i)} \log\left(1 - \hat{y}_c^{(i)}\right), \tag{8}$$

where

$$\hat{y}_c^{(i)} = \min\left\{1 - \epsilon, \quad \tilde{y}_c^{(i)}\right\}, \tag{9}$$

in order to avoid a zero-logarithm, with α and ϵ having the same behavior as in Eq. 7. This function will ensure that no saliency results whenever total misclassification occurs ($\log(1 - 0) = \log(1) = 0$). Note that we do not want to kill the gradient when clipping the value in Eq. 9, so the gradient should remain unaltered ($\nabla \hat{y}_c^{(i)} = 1$).

This approach is similar to that of Simonyan et al. [20], differing in just one point: while that method decomposes the last layer network to obtain the

saliency, in our approach the model is directly applied to a gain function, making it suitable for use in other machine learning approaches, like regression. An advantage of our approach is that it returns close-to-zero gradients whenever there is a misclassification. Our algorithm, unlike that of Simonyan et al. [20], is therefore able to indicate that there are no relevant features in a misclassification.

3.2.2. Hinge loss function

The hinge loss function is often used to train SVMs. In a multiclass problem, it can be defined as:

$$\ell_H(\tilde{\mathbf{Y}}, \mathbf{Y}) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_c^{(i)} \max(0, 1 - \tilde{y}_c^{(i)}) + (1 - y_c^{(i)}) \max(0, 1 + \tilde{y}_c^{(i)}). \quad (10)$$

Thus, correct class predictions will have values higher than 1, whereas incorrect class predictions will have values lower than -1.

Note that the function does not have a gradient when values are higher than 1 in correct classes (and similarly for -1 in incorrect classes), which means that a predicted output with value 2 would have the same information as a predicted output with value 2000. The gain function was thus modified as follows:

$$g_H(\tilde{\mathbf{Y}}, \mathbf{Y}) = -\frac{\alpha}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_c^{(i)} \log\left(1 - \breve{y}_c^{(i)}\right), \quad (11)$$

where

$$\breve{y}_c^{(i)} = \min\left\{1 - \epsilon, \quad \frac{\min(1, \max(-1, \tilde{y}_c^{(i)})) + 1}{2}\right\}. \quad (12)$$

Again, we do not kill the gradient after clipping ($\nabla \breve{y}_c^{(i)} = 1$).

4. Saliency-based feature selection

Our SFS approach, a ranker-based FS method that returns an ordered vector of all features based on their importance, is described in Fig. 3 and Algorithm 1.

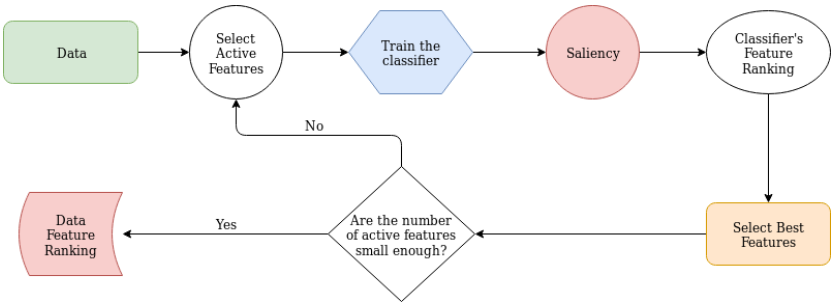

Figure 3: SFS algorithm flow chart.

Data: $\mathbf{X}, \mathbf{Y}, f, \ell, \mathbf{\Theta}, \gamma, \epsilon,$ reps

Result: feature ranking $\mathbf{r}$

$n_f \leftarrow R$ `// $n_f$ is the number of Alive features`

$\mathbf{r} \leftarrow [1 \dots n_f]$;

while $n_f > \epsilon > 1$ **do**

  $\hat{\mathbf{X}} \leftarrow \mathbf{X}$;

  $\hat{\mathbf{X}}[:, \mathbf{r}[n_f + 1 : R]] \leftarrow 0$;

  $\sigma_{fs} \leftarrow \text{zeros}(n_f)$;

  for $rep \leftarrow 1$ **to** C **do**

    Initialize $f(\hat{\mathbf{X}}; \mathbf{\Theta})$;

    Train $f(\hat{\mathbf{X}}; \mathbf{\Theta})$ given $\mathbf{Y}$;

    $\tilde{\mathbf{Y}} \leftarrow f(\hat{\mathbf{X}}; \mathbf{\Theta})$;

    $\sigma_{fs} \leftarrow \sigma_{fs} + \text{GetSaliency}(\tilde{Y}, Y, \sigma)$;

  end

  index $\leftarrow \text{argsort}(\sigma_{fs}, \text{descend})$;

  $\mathbf{r}[1 : n_f] \leftarrow \mathbf{r}[\text{index}]$;

  $n_f \leftarrow int(n_f * \gamma)$;

end

Algorithm 1: SFS algorithm pseudocode

The proposed algorithm contains just three hyperparameters: $\epsilon \geq 1$, a stopping criterion; $1 \geq \gamma > 0$ to control the number of alive features kept in the following iteration; and *reps* to control the number of times a model is trained so as to avoid overfitting.

We trained the model f with all the features in the feature set, computed saliency, then summed up and sorted all the features so as to obtain a feature ranking $\mathbf{r}$. The least relevant features were then discarded and the operation was repeated until the stopping criterion applied.

Function `GetSaliency`$(\tilde{Y}, Y, \sigma)$**:**

$\sigma \leftarrow 0$;

$C \leftarrow$ Number of classes in Y;

for $c \leftarrow 1$ **to** C **do**

$\sigma_c \leftarrow \sum_{i_c=1}^{N_c} \sigma(\tilde{\mathbf{y}}^{(i_c)}, \mathbf{y}^{(i_c)})$;

$\sigma \leftarrow \sigma + \frac{\sigma_c}{\|\sigma_c\|_1}$;

end

return σ

end

Algorithm 2: Saliency function for classification

How saliency is computed differs depending on whether we are dealing with a classification or a regression problem. As shown in Algorithm 2, for classification we compute and normalize saliency for each class and then sum up all the features. For regression we simply sum up all the saliency scores, as described in Algorithm 3.

Function `GetSaliency`$(\tilde{Y}, Y, \sigma)$**:**

$\sigma \leftarrow \sum_{i=1}^{N} \sigma(\tilde{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)})$;

return σ

end

Algorithm 3: Saliency function for regression

The complexity of the algorithm is variable, as it completely depends on the

Table 1: NIPS 2003 Feature Selection Challenge datasets

Name	# instances (train, test)	# features	# relevant features	% relevant features	pos/neg ratio
Arcene	(88, 112)	10000	7000	0.7	1.0
Dexter	(300, 300)	20000	9947	0.5	1.0
Dorothea	(800, 350)	100000	50000	0.5	0.11
Gisette	(6000, 1000)	5000	2500	0.5	1.0
Madelon	(2000, 600)	500	20	0.04	1.0

γ parameter. In the best case scenario, when $\gamma = 0$, complexity is linear in the number of instances ($\mathcal{O}(N)$), whereas in the worst case scenario, when $\gamma \approx 1$, complexity depends on the number of variables ($\mathcal{O}(RN)$), given that we only remove one feature in each loop.

5. Ablation case study

Below we demonstrate how the γ and *reps* parameters affect the behavior of our saliency algorithm and also show how decoupling to obtain the feature ranking from the model used for classification can affect our approach. We first describe the datasets used to test our methodology.

5.1. Datasets

5.1.1. NIPS 2003 Feature Selection Challenge

We used the five synthetic datasets – Arcene, Dexter, Dorothea, Gisette and Madelon – proposed for the NIPS 2003 Feature Selection Challenge,[1] designed for the sole purpose of measuring the quality of feature selection algorithms for classification. Table 1 lists the specific characteristics of each dataset. Although the datasets contain just two classes each, they are challenging because of specific traits, namely, they contain few training examples (Arcene), unbalanced data (Dorothea) or low-relevance features (Madelon).

[1] http://clopinet.com/isabelle/Projects/NIPS2003/

5.1.2. MNIST, FASHION-MNIST, CIFAR-10 and CIFAR-100

The NIPS 2003 datasets had too few example to be suitable for testing our approach in a big data scenario, so we also selected four classic computer vision datasets, as follows:

- MNIST, the classic computer vision classification challenge [26], contains some 50,000 handwritten digits (10 classes) stored in 28×28 grayscale resolution.
- FASHION-MNIST, a dataset of Zalando clothing images [27], contains 60,000 images (10 classes) stored in 28×28 grayscale resolution.
- CIFAR-10 and CIFAR-100 [28], containing some 50,000 tiny RGB images ($32 \times 32 \times 3$) of objects such as cars, truck, planes, etc belonging to 10 and 100 different classes, respectively.

5.1.3. Regression

Unlike classic information-based FS algorithms [29], our SFS algorithm is able to perform FS for regression problems. For testing purposes we used two big data datasets:

- *Relative Location of CT Slices on Axial Axis*[2][30]. This dataset, whose aim, as indicated by the name, is to uncover the relative location of computed tomography (CT) images on the axial axis, contains 53,500 images for 74 different patients. Each image is described by two histograms, for a total of 385 features.
- *Energy Molecule* [3] [31]. The aim of this dataset is to train machine learning techniques to quickly compute atomization energy, as the corresponding simulations are computationally time-consuming. The dataset contains ground state energies for 16,242 molecules, each with 1,275 features.

[2] https://archive.ics.uci.edu/ml/datasets/Relative+location+of+CT+slices+on+axial+axis
[3] https://www.kaggle.com/burakhmmtgl/energy-molecule

5.2. Testing algorithm parameter effects

5.2.1. The reps parameter

It is almost impossible to train the same machine learning model more than once and expect to achieve the exact same result each time, as random initialization in a model's weights or random permutations in the training set cause small differences in output. Thus, we first evaluated how the number of training repetitions could affect the output. Our objective was to check if it would be necessary to train the model more than once at each step, computing the saliency ranking as the mean average of all repetitions.

We trained our algorithm using the NIPS 2003 datasets, setting $\gamma = 0$ and trying different numbers of repetitions. We trained a 3-layer fully-connected NN (150, 100 and 50 nodes per layer) using batch normalization (BN) [32] and $ReLU(x) = max(0, x)$ activation. The softmax function was used as output and Eq. 3 was used as the loss function. We included l2 weight decay regularization with factor 0.001 and trained the model for 100 epochs using the Adam optimizer [33]. To deal with unbalanced data, we replicated the number of examples until we achieved balance between classes. All models were created using the Keras framework[4] and TensorFlow [34] as back-end.

From Fig. 4 it can be seen that the number of repetitions affected accuracy in most datasets, and that this effect depended also on the number of features. A Friedman test showed no significant differences between the models when the number of repetitions was greater than two, except for the Madelon dataset, with a small number of relevant features. We also found significant differences between these models and the model with just one repetition. On that basis, we would recommend using more than two repetitions to ensure better performance.

5.2.2. Effect of γ parameter

To analyze the behavior of the γ parameter, we set $reps = 1$, while keeping the remaining parameters the same as for the previous test. Fig. 5 shows that

[4] https://keras.io/

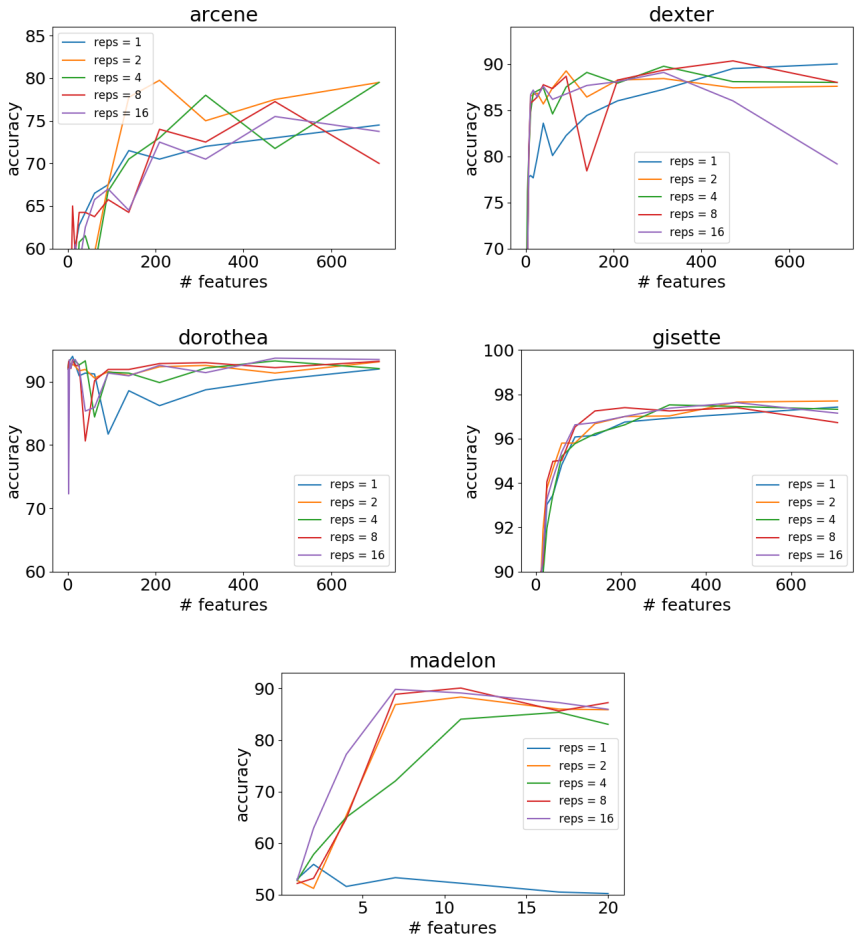

Figure 4: Effect of the number of training repetitions with our algorithm. The number of repetitions significantly affects the outcome.

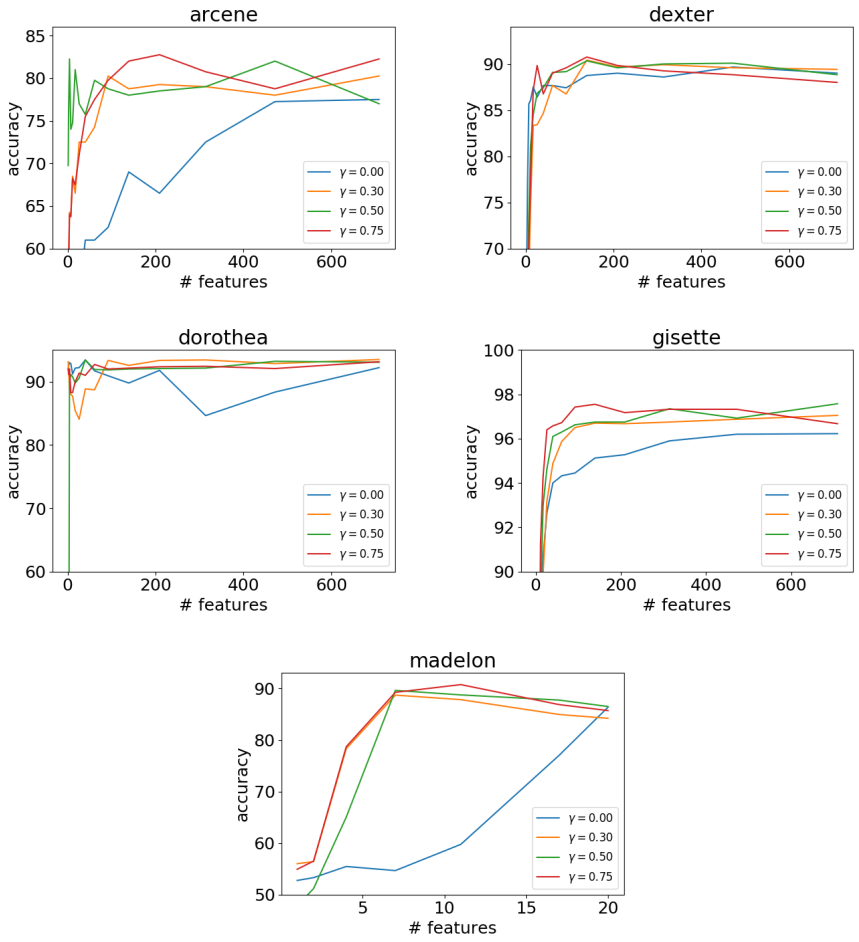

Figure 5: Effect of the hyperparameter γ with our algorithm. Using $\gamma = 0$ led to poor results when using a small number of features. However, there was no significant performance improvement when $\gamma > .3$.

the accuracy of our algorithm improved as the γ value increased. This occurred because when features were removed, redundancy was also reduced, causing the other features to become more important. Although increasing the γ value helped our algorithm achieve better scores, the Friedman test results indicate that there were no significant differences when we set $\gamma \geq 0.3$, except for the Madelon dataset. Furthermore, Wilcoxon testing indicated that there was a significant difference between the $\gamma = 0$ and $\gamma = 0.75$ models for the Arcene, Madelon and Gisette datasets, but no significant difference for the Dexter and Dorothea datasets. In the latter datasets, this may be because of the high feature-to-sample ratios.

5.2.3. Effect of ranker-classifier decoupling

Our FS algorithm is an embedded model: both selection and classification/regression tasks can be performed at the same time. An embedded system selects the features that achieve good results in the same model used for either classification or regression. However, the problem is that this might lead to the selection of features that are only valid for the machine learning model used to obtain the ranking.

A question we needed to answer, therefore, was: how good is our selection?. We consequently tested our proposal separating the problem into two different tasks: ranking and classifying. By using different models for each task, we tested how dependent the ranking was on the model used to obtain it. We used the four kernel implementations provided by sklearn's SVC dataset [35], for parameters $C = 1$, $degree = 3$ and $coef0 = 1$. Our algorithm meta-parameters were set to $\gamma = 0.975$ and $reps = 1$. We did not need any more repetitions, as the SVM-training sequential minimal optimization (SMO) algorithm [36] is very stable. Table 2 shows the results obtained for the NIPS 2003 datasets.

The answer to the question posed above was therefore as follows:

1. Only in 35% of cases was the best result obtained by using the same algorithm for both ranking and classification. Since we are using four different models, this percentage is close to random.

Table 2: Decoupling ranker and classifier accuracy results. The number of features used to achieve the best score and the best score achieved by each classifier are indicated in brackets and bold, respectively. For multiple rankers achieving the best score, the ranker with the fewest features is highlighted.

Dataset	SVM Ranker	SVM Classifier (# of features)			
		Linear	Poly	RBF	Sigmoid
Arcene	Linear	83.0 (3189)	**88.0** (9750)	85.0 (48)	**75.0** (327)
	Poly	84.0 (5578)	**88.0** (9750)	81.0 (1088)	73.0 (438)
	RBF	83.0 (1269)	**88.0** (9750)	**88.0** (487)	74.0 (3622)
	Sigmoid	**84.0** (1145)	**88.0** (9750)	81.0 (1088)	72.0 (3715)
Dexter	Linear	**94.3** (1419)	93.3 (99)	**93.7** (105)	**93.0** (102)
	Poly	93.6 (7071)	91.0 (66)	90.7 (33)	90.3 (68)
	RBF	94.0 (7071)	92.0 (93)	92.7 (37)	91.7 (40)
	Sigmoid	93.7 (3384)	**93.7** (66)	92.3 (54)	92.0 (64)
Dorothea	Linear	94.9 (222)	94.6 (216)	94.6 (79)	**95.1** (11318)
	Poly	**94.9** (69)	**95.1** (88)	**94.9** (65)	94.9 (150)
	RBF	94.6 (26)	94.6 (25)	94.6 (23)	94.9 (27)
	Sigmoid	94.3 (71)	94.6 (75)	94.9 (96)	94.9 (85)
Gisette	Linear	**98.3** (1080)	98.3 (714)	98.1 (351)	97.6 (351)
	Poly	98.1 (168)	98.2 (360)	98.2 (470)	97.9 (240)
	RBF	98.2 (275)	98.2 (581)	98.2 (412)	98.0 (412)
	Sigmoid	98.0 (315)	**98.3** (483)	**98.3** (351)	**98.0** (351)
Madelon	Linear	58.2 (218)	71.7 (94)	73.7 (271)	**57.0** (374)
	Poly	61.0 (9)	**76.2** (18)	88.0 (11)	52.3 (500)
	RBF	60.5 (5)	70.3 (96)	89.3 (12)	53.7 (3)
	Sigmoid	**62.0** (4)	71.3 (108)	**90.8** (17)	52.3 (500)

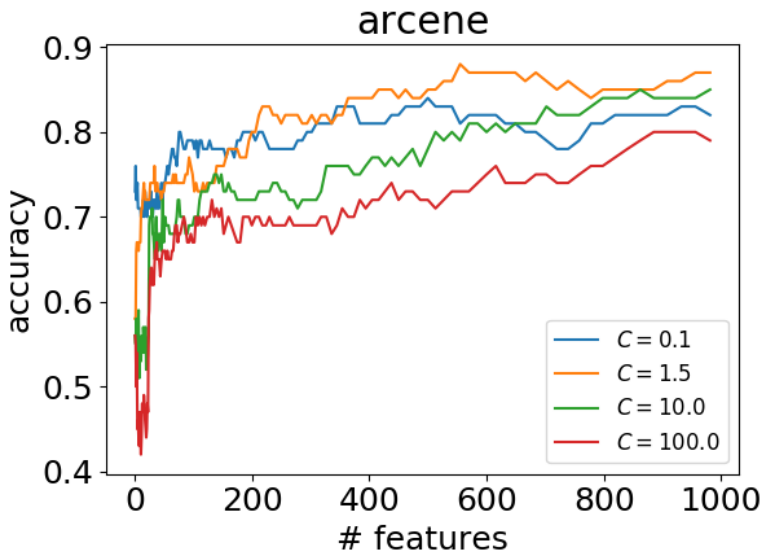

Figure 6: Effect of overfitting during the FS step. As classifier overfitting increases, FS quality decreases.

2. In three datasets (Arcene, Dorothea and Gisette), the best ranker was the same for three different classifiers. This would suggest that the selected features were not substantially affected by the type of classifier. Thus, a good ranker model performs well irrespective of the classifier used.

We can therefore conclude that it is more important to have a good classifier for FS than to try and use the same model for both ranking and classifying.

5.2.4. Effect of overfitting in the FS step

As we have seen in the previous subsection, good classifier selection is crucial to good FS, so we also needed to evaluate the impact of model overfitting. We report results only for the Arcene dataset, as it yielded very meaningful results. The best results were achieved using SVM-RBF for both ranking and classifying, with $C = 1.5$ as the meta-parameter (see Table 2). In our experiment we therefore used the same classifier configuration, checking how the result varied as we modified the C parameter during the FS step and setting our algorithm

Table 3: Instance-level results. For each sample, the saliency function successfully provides information about which features are most relevant.

$\mathbf{X} = \{x_a, x_b, x_c, x_d, x_e\}$	$\mathbf{Y}$	significant variables	saliency($\mathbf{X}$)
$\{-0.5, -0.5, -0.5, -0.5, 0.5\}$	1	$\{x_a, x_b, x_c\}$	$\{\mathbf{9.0}, \mathbf{5.9}, \mathbf{8.7}, 0.7, 0.8\}$
$\{-0.5, 0.5, -0.5, 0.5, 0.5\}$	-1	$\{x_a, x_b, x_c\}$	$\{\mathbf{7.4}, \mathbf{6.8}, \mathbf{6.3}, 2.5, 0.1\}$
$\{0.5, 0.5, -0.5, -0.5, -0.5\}$	1	$\{x_a, x_d, x_e\}$	$\{\mathbf{6.3}, 0.9, 0.1, \mathbf{5.1}, \mathbf{5.7}\}$
$\{0.5, 0.5, 0.5, -0.5, 0.5\}$	-1	$\{x_a, x_d, x_e\}$	$\{\mathbf{9.9}, 2.0, 2.4, \mathbf{3.7}, \mathbf{5.7}\}$

meta-parameters at their minimum values ($\gamma = 0$ and $reps = 1$). Fig. 6 shows that the best results were achieved when using the same C value as used in the classification step. While using a lower value did not substantially affect the result, overfitting (i.e., high C values) drastically affected the quality of the result. Thus, we strongly recommend using classifiers in the FS step that prevent overfitting, to avoid the introduction of noise in the FS ranking system.

5.3. Benefits of instance-level information

Besides dimensionality reduction, our algorithm provides instance-level information, i.e., it shows which features are most relevant for each specific case. This is particularly interesting in terms of explainable models, as it provides insights as to why a model makes certain decisions.

Using a toy example to demonstrate how our algorithm provides insightful instance-level information, let $\mathbf{X} = \{x_a, x_b, x_c, x_d, x_e\}$ be a random uniform datum ($\mathbf{X} \in [-1, 1]$) and let

$$\mathbf{Y} = \begin{cases} \text{sign}(x_b) \times \text{sign}(x_c) & \text{if } x_a < 0 \\ \text{sign}(x_d) \times \text{sign}(x_e) & \text{otherwise} \end{cases} \tag{13}$$

be the expected output. The output will always depend on three variables: one fixed variable (x_a) and two variables that depend on the fixed variable value. While all five variables are, by definition, needed to successfully train the classifier, each instance only takes into account three of them.

Table 3 shows the instance-level information obtained for an NN trained with a single hidden layer. The higher values correspond to features used to

create the expected output.

6. SFS compared with other FS methods

In order to test our proposed SFS, we compared it to the most representative FS methods currently available, namely: (a) Lasso and Elastic Net implementations in the sklearn package [37, 38]; (b) the MIM revision [39], also implemented in the sklearn package; (c) the ReliefF algorithm [40] implemented in the skrebate package [41]; and (d) the DFS algorithm [19], which uses our own Keras implementation. To ensure a fair comparison, we used DFS mask values as the feature rankings for the DFS. Note that this approach was not mentioned in the original paper [19], as the model was only trained with the new mask and constraint (like Lasso and Elastic Net). However, experimental results show that this approach achieves better results (the code with our implementations is available at GitHub[5]).

6.1. NIPS 2003 Feature Selection Challenge

As the number of instances was relatively low in all the NIPS 2003 datasets, we followed the same approach as explained in the ablation study described above; i.e., we used four SVM variants to test algorithm performance. In the case of the DFS algorithm, we used the 3-layer NN (also described above) as the FS algorithm. We also tried the same network with our proposed approach.

6.1.1. Arcene dataset

From Table 4 it can be observed that the best accuracy results were achieved using the polynomial kernel, but using practically all the features (in the most favourable case, 81.6%). Our algorithm achieved equivalent accuracy results but using an RBF kernel and only 19 features (i.e., only 4.87% of all the features in the dataset).

Table 4: Accuracy results for the Arcene dataset. The number of features that achieved the best score is indicated in brackets. 'Baseline' refers to classifier accuracy without feature removal. 'Same' in the ranker column means that the same SVM model was used as both ranker and classifier. The best ranker-classifier combinations are indicated in bold.

FS Method	Ranker	SVM Classifier (# of features)			
		Linear	Poly	RBF	Sigmoid
Baseline (all features)	—	83.0	87.0	72.0	69.0
LASSO [37, 38]	—	70.0	—	—	—
Elastic Net [37, 38]	—	76.0	—	—	—
MIM [39]	—	83.0 (84)	**88.0** (8164)	81.0 (25)	69.0 (1817)
ReliefF [40]	—	83.0 (10000)	**88.0** (8818)	75.0 (600)	71.0 (2231)
DFS [8]	NN	86.0 (1145)	87.0 (8810)	77.0 (302)	72.0 (4439)
SFS	Same ($\gamma = 0$)	84.0 (4328)	87.0 (10000)	83.0 (600)	74.0 (3715)
	Best ($\gamma = 0$)	85.0 (318)	87.0 (9036)	86.0 (513)	78.0 (2601)
	Same ($\gamma = 0.975$)	83.0 (3189)	**88.0** (9760)	**88.0** (487)	72.0 (3715)
	Best ($\gamma = 0.975$)	84.0 (1145)	**88.0** (9750)	**88.0** (487)	84.0 (1145)
	NN ($\gamma = 0.9$, reps $= 3$)	83.0 (1599)	87.0 (5869)	**88.0 (19)**	75.0 (465)

Table 5: Accuracy results for the Dexter dataset. The number of features that achieved the best score is indicated in brackets. 'Baseline' refers to classifier accuracy without feature removal. 'Same' in the ranker column means that the same SVM model was used as both ranker and classifier. The best ranker-classifier combinations are indicated in bold.

FS Method	Ranker	SVM Classifier (# of features)			
		Linear	Poly	RBF	Sigmoid
Baseline (all features)	—	93.7	89.0	89.0	89.0
LASSO [37, 38]	—	89.0	—	—	—
Elastic Net [37, 38]	—	91.3	—	—	—
MIM [39]	—	93.7 (20000)	92.0 (166)	91.0 (152)	90.7 (1533)
ReliefF [40]	—	93.7 (20000)	91.7 (31)	91.0 (68)	90.0 (111)
DFS [8]	NN	93.7 (20000)	90.7 (93)	90.3 (2979)	90.3 (3746)
SFS	Same ($\gamma = 0$)	94.0 (939)	92.0 (56)	91.3 (52)	91.0 (4710)
	Best ($\gamma = 0$)	94.0 (939)	92.7 (52)	91.3 (52)	91.0 (4710)
	Same ($\gamma = 0.975$)	**94.3 (1419)**	91.0 (66)	92.7 (37)	92.0 (64)
	Best ($\gamma = 0.975$)	**94.3 (1419)**	93.7 (7071)	94.0 (7071)	93.7 (3384)
	NN ($\gamma = 0.9$, reps $= 3$)	94.0 (2493)	91.7 (111)	91.7 (114)	91.0 (124)

Table 6: Accuracy results for the Dorothea dataset. The number of features that achieved the best score is indicated in brackets. 'Baseline' refers to classifier accuracy without feature removal. 'Same' in the ranker column means that the same SVM model was used as both ranker and classifier. The best ranker-classifier combinations are indicated in bold.

FS Method	Ranker	SVM Classifier (# of features)			
		Linear	Poly	RBF	Sigmoid
Baseline (all features)	—	93.1	90.3	9.7	92.3
LASSO [37, 38]	—	93.4	—	—	—
Elastic Net [37, 38]	—	93.7	—	—	—
MIM [39]	—	94.0 (1677)	94.9 (112)	94.9 (77)	94.9 (77)
ReliefF [40]	—	94.6 (1030)	94.3 (79)	94.3 (67)	94.3 (7)
DFS [8]	NN	94.6 (59)	95.1 (61)	**95.4 (57)**	95.1 (53)
SFS	Same ($\gamma = 0$)	94.0 (3995)	**95.4** (283)	95.1 (138)	94.9 (94)
	Best ($\gamma = 0$)	94.3 (94)	**95.4** (283)	**95.4** (381)	94.9 (94)
	Same ($\gamma = 0.975$)	94.9 (222)	95.1 (88)	94.6 (23)	94.9 (27)
	Best ($\gamma = 0.975$)	94.9 (69)	95.1 (88)	94.9 (65)	95.1 (11318)
	NN ($\gamma = 0.9$, reps $= 3$)	94.3 (193)	94.3 (85)	94.9 (381)	94.6 (402)

6.1.2. Dexter dataset

As can be observed in Table 5, the issue with the polynomial kernel in the Arcene dataset also emerged for the Dexter dataset, although this time with the linear kernel. Again, our algorithm outperformed the other techniques, independently of the SVM kernel selected. This time our approach not only considerably reduced the number of features used (7.1% of features in our case, compared to 100% for the other methods) but also slightly improved the accuracy result.

6.1.3. Dorothea dataset

Table 6 shows that the best result for Dorothea was achieved using the DFS algorithm with an RBF kernel. Our approach achieved the same accuracy with both polynomial and RBF kernels, but requiring more features (57 for the DFS algorithm and 283 for our algorithm, 0.057% and 0.3% of the full set of features, respectively).

[5] https://github.com/braisCB/SFS

Table 7: Accuracy results for the Gisette dataset. The number of features that achieved the best score is indicated in brackets. 'Baseline' refers to classifier accuracy without feature removal. 'Same' in the ranker column means that the same SVM model was used as both ranker and classifier. The best ranker-classifier combinations are indicated in bold.

FS Method	Ranker	SVM Classifier (# of features)			
		Linear	Poly	RBF	Sigmoid
Baseline (all features)	—	97.7	97.5	96.9	95.7
LASSO [37, 38]	—	97.4	—	—	—
Elastic Net [37, 38]	—	97.4	—	—	—
MIM [39]	—	97.7 (2212)	97.8 (1997)	97.6 (902)	96.7 (645)
ReliefF [40]	—	98.2 (3083)	98.2 (1803)	97.7 (1850)	97.1 (1587)
DFS [8]	NN	98.2 (168)	98.1 (3503)	**98.3** (130)	97.5 (163)
SFS	Same ($\gamma = 0$)	98.1 (551)	98.1 (333)	98.0 (423)	97.8 (645)
	Best ($\gamma = 0$)	98.1 (299)	98.3 (662)	98.1 (275)	97.8 (412)
	Same ($\gamma = 0.975$)	98.3 (1080)	98.2 (360)	98.2 (412)	98.0 (351)
	Best ($\gamma = 0.975$)	98.3 (1080)	98.3 (483)	98.3 (351)	98.0 (351)
	NN ($\gamma = 0.9$, reps $= 3$)	98.0 (1397)	**98.4 (291)**	98.2 (333)	97.8 (324)

6.1.4. Gisette dataset

Table 7 shows that our algorithm achieved the highest score using the NN as ranker, although DFS obtained similar accuracy with fewer features (130 for DFS versus 291 for our approach, 2.6% and 5.8% of the full set of features, respectively). Compared with either MIM or ReliefF, our algorithm systematically selected fewer features.

6.1.5. Madelon dataset

As indicated by Table 8, the ReliefF algorithm achieved the best score. Noteworthy, however, was the fact that our algorithm achieved the same score, although using more features (1.8% and 5% of the total number of features, respectively).

To sum up, compared with state-of-the art algorithms, our algorithm achieved the same or even slightly improved FS accuracy results for all five NIPS 2003 datasets. Regarding the number of features used, our SFS algorithm greatly reduced the number of features needed, while in other cases in which existing

Table 8: Accuracy results for the Madelon dataset. The number of features that achieved the best score is indicated in brackets. 'Baseline' refers to classifier accuracy without feature removal. 'Same' in the ranker column means that the same SVM model was used as both ranker and classifier. The best ranker-classifier combinations are indicated in bold.

FS Method	Ranker	SVM Classifier (# of features)			
		Linear	Poly	RBF	Sigmoid
Baseline (all features)	—	53.0	67.7	68.7	52.3
LASSO [37, 38]	—	58.3	—	—	—
Elastic Net [37, 38]	—	59.7	—	—	—
MIM [39]	—	62.5 (5)	72.5 (128)	80.2 (15)	57.2 (3)
ReliefF [40]	—	62.7 (4)	74.7 (36)	**91.5 (9)**	53.2 (111)
DFS [8]	NN	62.5 (8)	72.0 (40)	90.8 (11)	53.0 (1)
SFS	Same ($\gamma = 0$)	59.2 (56)	73.5 (42)	83.9 (32)	54.7 (76)
	Best ($\gamma = 0$)	62.3 (1)	73.5 (42)	83.9 (32)	62.3 (58)
	Same ($\gamma = 0.975$)	58.2 (218)	76.2 (18)	89.3 (12)	52.3 (500)
	Best ($\gamma = 0.975$)	62.0 (4)	76.2 (18)	90.8 (17)	57.0 (374)
	NN ($\gamma = 0.9$, reps $= 3$)	62.8 (9)	77.8 (16)	**91.5** (25)	52.8 (87)

methods achieve an important reduction, our approach needed approximately the double of features.

6.2. Regression

We conducted two experiments to test behavior of our SFS algorithm for a regression problem using (as mentioned previously) the Relative Location of CT Slices on Axial Axis and the Energy Molecule datasets. Our approach was similar to that described in Section 5.2.1, i.e., a 3-layer NN (150, 100 and 50 nodes), BN and the *ReLu* activation function, differing only in the output (now just one node) and the loss function (MSE). Cross-validation was five-fold.

Table 9 shows results for the CT dataset. We compared our SFS algorithm with the DFS algorithm, using an input mask with $l_1 = 5 \cdot 10^{-4}$ as the weight penalty. We also used our SFS method adding the DFS mask at the input (the SFS+DFS configuration). The *reps* parameter was set to 2 in all experiments. The SFS and SFS+DFS, both with a 3-layer CNN, achieved the best results. For the SFS alone, 192 features (the full set) were needed, whereas for the SFS+DFS

Table 9: SFS mean absolute error performance for a regression problem and the Relative Location of CT Slices on Axial Axis dataset.

FS Method	# of features			
	19	38	96	192
DFS [8]	4.18	**2.84**	2.32	2.39
SFS ($\gamma = 0$)	6.55	3.70	2.71	**2.24**
SFS ($\gamma = 0.9$)	4.18	3.16	2.60	2.31
SFS+DFS ($\gamma = 0$)	4.41	2.94	2.25	2.40
SFS+DFS ($\gamma = 0.9$)	**4.08**	2.87	**2.30**	2.27

Table 10: SFS mean absolute error performance for a regression problem and the Energy Molecule dataset.

FS Method	# of features			
	63	127	318	637
DFS [8]	**0.139**	0.136	0.137	0.142
SFS ($\gamma = 0$)	0.292	0.215	0.146	0.141
SFS ($\gamma = 0.9$)	0.145	**0.132**	**0.131**	**0.136**
SFS+DFS ($\gamma = 0$)	0.145	0.139	0.143	0.149
SFS+DFS ($\gamma = 0.9$)	**0.139**	0.136	**0.132**	**0.138**

combination, only 96 features (50%) were needed for equivalent accuracy.

Table 10 shows, for the same configuration, the results for the Energy Molecule dataset. SFS with $\gamma = 0.9$ achieved almost the best score using just 127 features (10% of the whole dataset), while SFS and SFS+DFS achieved the best scores for all the different configurations.

6.3. Results for big data scenarios

One of the main advantages of the SFS algorithm is that it can be used in big data environments, as it was developed for state-of-the-art architectures like CNNs. We further tested its behavior with four different datasets using the WRN-16-4 wide residual network [42] as classifier.

Table 11: MNIST accuracy results using WRN-16-4 as the classifier.

FS Method	Ranker	# of features			
		39	78	196	392
DFS [8]	3-layer CNN	95.73	98.56	99.32	99.46
DFS [8]	WRN-16-4	92.92	97.36	98.72	98.98
SFS	3-layer CNN ($\gamma = 0$)	89.78	95.66	99.14	99.53
SFS	3-layer CNN ($\gamma = 0.9$)	**97.08**	98.49	99.13	**99.56**
SFS	WRN-16-4 ($\gamma = 0$)	88.18	94.98	98.70	99.41
SFS	WRN-16-4 ($\gamma = 0.9$)	96.76	**98.62**	99.10	99.30
SFS+DFS	3-layer CNN ($\gamma = 0$)	95.60	98.47	**99.38**	99.48
SFS+DFS	WRN-16-4 ($\gamma = 0$)	92.92	97.36	98.72	98.98

We tested two different ranker configurations: WRN-16-4 and a standard 3-layer CNN consisting of two convolutional layers with 16 and 32 channels and, after each convolution, a 2×2 max-pooling layer. Finally, two fully connected layers were used, one containing 1,024 nodes, and the other sized according to the number of labels in the dataset (100 for CIFAR-100, 10 for the other datasets). Both BN [32] and the *ReLu* activation function were applied right after all hidden layers, and the softmax function was applied to the output. The Adam optimizer [33] was used to train the model.

Used for both networks was categorical cross-entropy as the loss function and a weight penalty $l_2 = 5 \cdot 10^{-4}$.

Our experiments were conducted with the image databases described earlier, i.e., MNIST, Fashion-MNIST, CIFAR-10 and CIFAR-100.

6.3.1. MNIST

No data augmentation techniques were used to train the models. The 3-layer CNN was trained for 40 epochs, while the WRN-16-4 required 80 epochs. As can be observed from the results in Table 11, our approach achieved the best scores in terms of balancing accuracy and the number of features used. Note also that the accuracy of the classifier improved greatly on using a high γ value.

Table 12: Fashion-MNIST accuracy results using WRN-16-4 as the classifier.

FS Method	Ranker	# of features			
		39	78	196	392
DFS [8]	3-layer CNN	78.85	85.5	**90.45**	92.41
DFS [8]	WRN-16-4	72.58	76.52	87.05	**92.61**
SFS	3-layer CNN ($\gamma = 0$)	67.85	81.86	89.33	92.36
SFS	3-layer CNN ($\gamma = 0.9$)	**82.63**	**86.33**	90.09	**92.60**
SFS	WRN-16-4 ($\gamma = 0$)	64.17	73.53	85.60	92.48
SFS	WRN-16-4 ($\gamma = 0.9$)	77.99	82.58	88.16	91.72
SFS+DFS	3-layer CNN ($\gamma = 0$)	79.81	**86.29**	**90.44**	**92.59**
SFS+DFS	WRN-16-4 ($\gamma = 0$)	65.09	73.87	85.86	92.40

Table 13: CIFAR-10 accuracy results using WRN-16-4 as the classifier.

FS Method	Ranker	# of features			
		153	307	768	1536
DFS [8]	3-layer CNN	67.43	**79.92**	87.71	90.69
DFS [8]	WRN-16-4	61.51	71.19	83.94	89.47
SFS	3-layer CNN ($\gamma = 0$)	61.00	72.49	84.31	89.31
SFS	3-layer CNN ($\gamma = 0.9$)	63.52	77.96	89.60	91.5
SFS	WRN-16-4 ($\gamma = 0$)	53.03	60.42	85.55	90.44
SFS	WRN-16-4 ($\gamma = 0.9$)	64.15	79.27	**89.85**	**91.58**
SFS+DFS	3-layer CNN ($\gamma = 0$)	**68.13**	79.03	88.04	91.06
SFS+DFS	WRN-16-4 ($\gamma = 0$)	54.05	68.65	82.47	89.54

Table 14: CIFAR-100 accuracy results using WRN-16-4 as the classifier.

FS Method	Ranker	# of features			
		153	307	768	1536
DFS [8]	3-layer CNN	34.55	**49.68**	57.92	62.22
DFS [8]	WRN-16-4	28.24	40.77	56.98	**67.42**
SFS	3-layer CNN ($\gamma = 0$)	24.53	34.14	52.67	63.45
SFS	3-layer CNN ($\gamma = 0.9$)	30.74	44.55	61.49	66.96
SFS	WRN-16-4 ($\gamma = 0$)	24.66	37.86	56.66	66.39
SFS	WRN-16-4 ($\gamma = 0.9$)	27.88	44.45	**62.83**	67.22
SFS+DFS	3-layer CNN ($\gamma = 0$)	**36.86**	46.64	60.46	63.55
SFS+DFS	WRN-16-4 ($\gamma = 0$)	25.08	37.29	54.07	64.94

Later we discuss the intuition behind this effect.

6.3.2. Fashion-MNIST

As data augmentation we used random horizontal flips, along with horizontal and vertical random shifts (up to 4 pixels). As this dataset is more complex than MNIST, we increased the number of training epochs to 80 for the 3-layer CNN and to 130 for the WRN-16-4. Table 12 shows our result, indicating that, overall, our SFS approach with $\gamma = 0.9$ achieved the best scores.

6.3.3. CIFAR-10

We used the same data augmentation and training configuration as for Fashion-MNIST, but increased the random shifts to 5 pixels. As confirmed by Table 13, use of DFS helped the model achieve better results when the number of retained features was low. In contrast, a high γ value was useful with higher numbers of features (more than 25% of the full set of features).

6.3.4. CIFAR-100

Using the same training configuration as for CIFAR-10, we obtained the results in Table 14. For this dataset, the result for our approach was not clearly

better than the result for DFS, possibly because of an overfitting problem (although training accuracy was close to 100%, test accuracy never reached 70%).

6.4. Classifier reliability

To our knowledge, ours is the first FS algorithm, outside tree-based techniques, that can provide information on feature importance. The main advantage of our approach is that it can explain classifier decisions. Regarding classifier reliability, if we look at the MNIST results (Table 11), we can see that the best results were achieved using the 3-layer CNN rather than the WRN-16-4 model as ranker. This result was not expected because the latter is considered a better classifier than the former. Thus, our intuition regarding this effect is as follows:

1. Max-pooling. We think the main problem with our algorithm was that it did not manage feature correlation unless the training algorithm did. In the case of a CNN model, local correlations can be detected using pooling layers. Unlike the WRN-16-4 classifier, the 3-layer CNN contained 2 max-pooling layers, which may have helped our algorithm achieve better scores.
2. Over-fitting. Training set accuracy was always optimal using the WRN-16-4. This may have led our algorithm to poor generalization behavior, as explained in relation to the ablation study (see Fig. 6).

We thus conclude that the quality of the classifier greatly affected the behavior of our algorithm. Furthermore, the MNIST results indicate that test accuracy is not a good metric to deterime the suitability of a classifier for use as a ranker in the SFS algorithm. We therefore conducted another experiment, taking advantage of saliency properties. As previously defined, since our saliency function is the gradient of the gain function with respect to the input, it measures how we have to modify our input to increase the probability of belongingness to the desired class. We therefore used our saliency function to do exactly the opposite. The idea is to answer the simple question: how much do I have to change a sample to change the classifier's output?

Table 15: Adversarial images. Given an original image X, the minimal perturbation needed to force a trained classifier to predict each label with more than 95% certainty.

	X	$X=0$	$X=1$	$X=2$	$X=3$	$X=4$	$X=5$	$X=6$	$X=7$	$X=8$	$X=9$
WRN-16-4 w/o Input Noise											
WRN-16-4 w/ Input Noise											

This issue of adversarial examples as inputs to an NN that result in incorrect output has attracted great attention in recent years. Several techniques have been developed to increase the reliability of classifiers, including the fast gradient sign method (FGSM) [43], FGSM variants [44] and projected gradient descent (PGD) [45], which use the saliency gradient to evaluate how much an input sample must be modified to cheat the classifier. In our case, we extended these works to the evaluation of classifier reliability, focusing not so much on how an image has to be modified, but whether the generated image can cheat the human eye.

Our experiment is depicted in Table 15. Given original images and a trained classifier, we used the saliency output to make perturbations in the images aimed at cheating the classifier and obtaining wrong predictions with high certainty (greater than 95%). The first row shows the perturbations needed in a WRN-16-4 model, achieving 99.69% accuracy in the test set.

The resulting images were not substantially visually different from the original, as only random noise was added to the original; in other words, ten different images that looked extremely similar were able to achieve completely different predictions in our WRN-16-4 model.

Since introducing white noise could easily fool our WRN-16-4 model, we retrained it, but this time introducing some random Gaussian noise in the training images. The quality of the classifier decreased, as we only obtained 99.37% accuracy in the test set. However, on looking at the images (Table 15, second row), we can see that the perturbed images look completely different from the originals and so it should be easy to establish classifier prediction by just exam-

ining the input. Therefore, we conclude that the second model is more reliable even though it achieves lower accuracy in the training set.

This very interesting effect has implications for the training algorithm. As we can easily obtain images that can fool our model, we can make adjustments to our training set that improve the reliability of our model. This is a very important factor in decision support systems, as conclusions are based on more solid explanations. It also has implications for medical analysis, as, for any given sample, we can demonstrate a conclusion to doctors and explain how input should be modified to modify predictions. Together with a dictionary of potential treatments and their effect on the input parameters, it could also lead to treatment recommendations.

7. Conclusion

Our novel saliency-based FS approach (the SFS algorithm), contrary to classic approaches, ranks the importance of each feature at the instance level rather than in relation to the entire dataset. Experimental state-of-the-art results under different configurations for challenging datasets show that our algorithm is suitable for use in any kind of classification or regression problem. In contrast with classic information-based FS techniques, the reduced complexity of our SFS algorithm, which can be computed simultaneously with classification or regression training, allows it to be used in high-dimension datasets.

7.1. Contributions

Summarizing the contributions of our work in terms of the strengths and the disadvantages of the SFS algorithm, advantages are as follows:

1. *Versatility.* The SFS algorithm can be used in many different scenarios, for different target objectives (classification or regression) and with different model architectures (NNs, SVMs, etc).
2. *Robustness.* The SFS algorithm achieves state-of-the-art results with both small and big data datasets, while its ability to work with different architectures makes it a powerful FS technique in multiple scenarios.

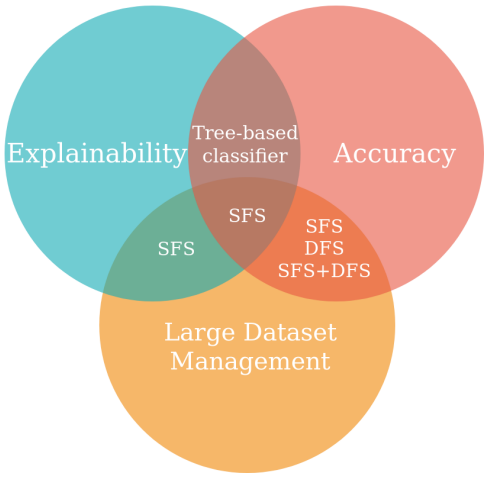

Figure 7: Feature selection algorithm choice depending on needs.

3. *Instance-level information.* The SFS algorithm provides detailed information for each sample introduced in the model that is very useful in terms of explainability.

Disadvantages are as follows:

1. *Failure to detect redundancy.* As the SFS algorithm focuses on the gradient of the gain function, it cannot detect redundancy. This problem is partially solved when using a CNN architecture, but only for the detection of locally spatial correlations (other redundancies cannot be detected).
2. *Wrapper problems.* The SFS algorithm has the same problems as wrapper methods, namely, it can be slow (using $\gamma \approx 1$), there is a risk of over-fitting and selection is dependent on the model used (for either classification or regression).
3. *Instance-level information in small datasets.* Although the SFS algorithm can provide instance-level information under all circumstances, in terms of explainability issues it possibly functions best when applied to large datasets (when the number of features is small a tree-based method will output more explainable information).

Fig. 7 is an FS algorithm recommendation graphic based on the three re-

quirements of an FS algorithm: explainability, accuracy and big data capability. Our SFS algorithm is recommended as the best state-of-the-art wrapper for big data environments, while a tree-based algorithm will provide a better explanation if the number of features is small.

7.2. Future research

As future research, we aim to use our SFS technique to define a metric to evaluate a model's robustness, i.e., to measure how hard it is to fool a classifier or a regression architecture, so as to avoid having to rely on visual cues. We also plan to test adversarial images as part of an explainable model in a real scenario like medical information. Other lines of future research are to fine-tune weak points in the SFS algorithm, e.g., improve explainability in small datasets through SFS combined with tree-based methods in a hybrid model and resolve the redundancy problem, e.g., by including an additional step aimed at detecting correlation between variables.

8. Acknowledgements

This research was financially supported in part by the Spanish Ministerio de Economía y Competitividad (research project TIN2015-65069-C2-1-R), by the Xunta de Galicia (research projects ED431C 2018/34 and Centro Singular de Investigación de Galicia, accreditation 2016-2019) and by the European Union (European Regional Development Fund). We also gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research. Brais Cancela acknowledges the support of the Xunta de Galicia under its postdoctoral program.

References

[1] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, S. U. Khan, The rise of "big data" on cloud computing: Review and open research issues, Information systems 47 (2015) 98–115.

[2] S. Abe, Feature selection and extraction, in: Support Vector Machines for Pattern Classification, Springer, 2010, pp. 331–341.

[3] I. Guyon, A. Elisseeff, An introduction to feature extraction, in: Feature extraction, Springer, 2006, pp. 1–25.

[4] A. Romero, C. Gatta, G. Camps-Valls, Unsupervised deep feature extraction for remote sensing image classification, IEEE Transactions on Geoscience and Remote Sensing 54 (3) (2016) 1349–1362.

[5] Y. Chen, H. Jiang, C. Li, X. Jia, P. Ghamisi, Deep feature extraction and classification of hyperspectral images based on convolutional neural networks, IEEE Transactions on Geoscience and Remote Sensing 54 (10) (2016) 6232–6251.

[6] P. Krishnan, K. Dutta, C. Jawahar, Deep feature embedding for accurate recognition and retrieval of handwritten text, in: Frontiers in Handwriting Recognition (ICFHR), 2016 15th International Conference on, IEEE, 2016, pp. 289–294.

[7] I. Guyon, A. Elisseeff, An introduction to variable and feature selection, Journal of machine learning research 3 (Mar) (2003) 1157–1182.

[8] Q. Zou, L. Ni, T. Zhang, Q. Wang, Deep learning based feature selection for remote sensing scene classification., IEEE Geosci. Remote Sensing Lett. 12 (11) (2015) 2321–2325.

[9] Y. Han, Y. Yang, Y. Yan, Z. Ma, N. Sebe, X. Zhou, Semisupervised feature selection via spline regression for video semantic recognition, IEEE Transactions on Neural Networks and Learning Systems 26 (2) (2015) 252–264.

[10] J. Novaković, Toward optimal feature selection using ranking methods and classification algorithms, Yugoslav Journal of Operations Research 21 (1).

[11] V. Bolon-Canedo, N. Sanchez-Marono, A. Alonso-Betanzos, Feature selection for high-dimensional data, Springer, 2015.

[12] I. Guyon, J. Weston, S. Barnhill, V. Vapnik, Gene selection for cancer classification using support vector machines, Machine learning 46 (1-3) (2002) 389–422.

[13] H. Faris, A.-Z. Ala'M, A. A. Heidari, I. Aljarah, M. Mafarja, M. A. Hassonah, H. Fujita, An intelligent system for spam detection and identification of the most relevant features based on evolutionary random weight networks, Information Fusion 48 (2019) 67–83.

[14] M. Mafarja, I. Aljarah, A. A. Heidari, A. I. Hammouri, H. Faris, A.-Z. Ala'M, S. Mirjalili, Evolutionary population dynamics and grasshopper optimization approaches for feature selection problems, Knowledge-Based Systems 145 (2018) 25–45.

[15] M. T. Ribeiro, S. Singh, C. Guestrin, Why should i trust you?: Explaining the predictions of any classifier, in: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, ACM, 2016, pp. 1135–1144.

[16] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, ImageNet: A Large-Scale Hierarchical Image Database, in: CVPR09, 2009.

[17] R. Ibrahim, N. A. Yousri, M. A. Ismail, N. M. El-Makky, Multi-level gene/mirna feature selection using deep belief nets and active learning, in: Engineering in Medicine and Biology Society (EMBC), 2014 36th annual international conference of the IEEE, IEEE, 2014, pp. 3957–3960.

[18] C. Feng, M. Cui, B.-M. Hodge, J. Zhang, A data-driven multi-model methodology with deep feature selection for short-term wind forecasting, Applied Energy 190 (2017) 1245–1257.

[19] Y. Li, C.-Y. Chen, W. W. Wasserman, Deep feature selection: theory and application to identify enhancers and promoters, Journal of Computational Biology 23 (5) (2016) 322–336.

[20] K. Simonyan, A. Vedaldi, A. Zisserman, Deep inside convolutional networks: Visualising image classification models and saliency maps, arXiv preprint arXiv:1312.6034.

[21] A. Mahendran, A. Vedaldi, Understanding deep image representations by inverting them, in: Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on, IEEE, 2015, pp. 5188–5196.

[22] R. Zhao, W. Ouyang, H. Li, X. Wang, Saliency detection by multi-context deep learning, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 1265–1274.

[23] D. Zhang, D. Meng, J. Han, Co-saliency detection via a self-paced multiple-instance learning framework, IEEE transactions on pattern analysis and machine intelligence 39 (5) (2017) 865–878.

[24] V. Mnih, N. Heess, A. Graves, et al., Recurrent models of visual attention, in: Advances in neural information processing systems, 2014, pp. 2204–2212.

[25] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, Y. Bengio, Show, attend and tell: Neural image caption generation with visual attention, in: International Conference on Machine Learning, 2015, pp. 2048–2057.

[26] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324.

[27] H. Xiao, K. Rasul, R. Vollgraf, Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms (2017).

[28] A. Krizhevsky, G. Hinton, Learning multiple layers of features from tiny images.

[29] V. Bolón-Canedo, N. Sánchez-Maroño, A. Alonso-Betanzos, A review of feature selection methods on synthetic data, Knowledge and information systems 34 (3) (2013) 483–519.

[30] F. Graf, H.-P. Kriegel, M. Schubert, S. Pölsterl, A. Cavallaro, 2d image registration in ct images using radial image descriptors, in: International Conference on Medical Image Computing and Computer-Assisted Intervention, Springer, 2011, pp. 607–614.

[31] B. Himmetoglu, Tree based machine learning framework for predicting ground state energies of molecules, The Journal of chemical physics 145 (13) (2016) 134101.

[32] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, arXiv preprint arXiv:1502.03167.

[33] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980.

[34] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al., Tensorflow: A system for large-scale machine learning., in: OSDI, Vol. 16, 2016, pp. 265–283.

[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.

[36] J. Platt, Sequential minimal optimization: A fast algorithm for training support vector machines.

[37] S.-J. Kim, K. Koh, M. Lustig, S. Boyd, D. Gorinevsky, An interior-point method for large-scale l_1-regularized least squares, IEEE journal of selected topics in signal processing 1 (4) (2007) 606–617.

[38] J. Friedman, T. Hastie, R. Tibshirani, Regularization paths for generalized linear models via coordinate descent, Journal of statistical software 33 (1) (2010) 1.

[39] B. C. Ross, Mutual information between discrete and continuous data sets, PloS one 9 (2) (2014) e87357.

[40] I. Kononenko, Estimating attributes: analysis and extensions of relief, in: European conference on machine learning, Springer, 1994, pp. 171–182.

[41] R. J. Urbanowicz, R. S. Olson, P. Schmitt, M. Meeker, J. H. Moore, Benchmarking relief-based feature selection methods, arXiv preprint arXiv:1711.08477.

[42] S. Zagoruyko, N. Komodakis, Wide residual networks, arXiv preprint arXiv:1605.07146.

[43] I. J. Goodfellow, J. Shlens, C. Szegedy, Explaining and harnessing adversarial examples, arXiv preprint arXiv:1412.6572.

[44] A. Kurakin, I. Goodfellow, S. Bengio, Adversarial machine learning at scale, arXiv preprint arXiv:1611.01236.

[45] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, A. Vladu, Towards deep learning models resistant to adversarial attacks, arXiv preprint arXiv:1706.06083.