Neurocomputing 548 (2023) 126422

Contents lists available at ScienceDirect

Neurocomputing

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

Feature selection for domain adaptation using complexity measures and swarm intelligence

G. Castillo-García *, L. Morán-Fernández, V. Bolón-Canedo

CITIC, Universidade da Coruña, A Coruña, Spain

ARTICLE INFO

Article history:
Received 14 February 2023
Revised 23 May 2023
Accepted 4 June 2023
Available online 8 June 2023
Communicated by Zidong Wang

Keywords:
Transfer learning
Domain adaptation
Feature selection
Data complexity
Particle swarm optimization
Sticky binary particle swarm optimization

ABSTRACT

Particle Swarm Optimization is an optimization algorithm that mimics the behaviour of a flock of birds, setting multiple particles that explore the search space guided by a fitness function in order to find the best possible solution. We apply the Sticky Binary Particle Swarm Optimization algorithm to perform feature selection for domain adaptation, a specific type of transfer learning in which the source and the target domain have a common feature space, a common task, but different distributions. When applying Particle Swarm Optimization, classification error is usually employed in the fitness function to evaluate the goodness of subsets of features. In this paper, we aim to compare this approach with using complexity metrics instead, under the assumption that reducing the complexity of the problem will lead to results that are independent from the classifier used for testing while being less computationally demanding. Therefore, we carried out experiments to compare the performance of both approaches in terms of classification accuracy, speed and number of features selected. We found out that our proposal, although in some cases incurs in a slight degradation of classification performance, it is indeed faster and selects fewer features, making it a feasible trade-off.

1. Introduction

Feature selection [1] is a machine learning technique used to reduce the dimensionality of a dataset. Its goal is to select only the relevant features for a predictive model, leaving the ones that are redundant or do not provide useful information out, which has several benefits. One such benefit is improved interpretability, which is particularly useful when the model's decisions need to be explained to human users. Additionally, feature selection can lead to better model performance and generalization by removing noise from the data and reducing the number of features, which decreases the risk of overfitting the training data. Finally, feature selection can also result in faster training and prediction times due to the decreased dimensionality of the data.

There are several approaches that can be taken to perform feature selection. One option is the use of filter methods, which rely on statistical measures of the features to identify the most relevant ones. These methods are independent of the machine learning algorithm and can be applied to any model. Alternatively, wrapper methods can be used, which involve training the model with different combinations of features and selecting the combination that yields the best classification performance. While more computationally expensive than filter methods, wrapper methods take into account the interaction between features and the model, potentially leading to higher performance. Embedded methods, which are built into the machine learning algorithm, can also be used for feature selection by identifying the most relevant features through a combination of model training and feature selection. Hybrid methods, which combine the strengths of different approaches, are another option for feature selection, such as using a filter method to pre-select features and a wrapper method to fine-tune the selection.

There are many successful cases of feature selection in machine learning. For example, in natural language processing tasks, such as text classification, feature selection can be used to identify the most important words or phrases in a document that are most relevant to the classification task [2]. In computer vision tasks, feature selection can be used to identify the most important pixels or image features that are most relevant to the task at hand [3]. In general, feature selection can be applied to a wide range of machine learning tasks.

Transfer learning, on its side, is a technique that aims to use acquired knowledge from an existing source domain to improve learning performance in a different, yet similar target domain. In our case, we deal with problems where there is a common feature

* Corresponding author.
E-mail addresses: guillermo.castillo@udc.es (G. Castillo-García), laura.moranf@udc.es (L. Morán-Fernández), veronica.bolon@udc.es (V. Bolón-Canedo).

https://doi.org/10.1016/j.neucom.2023.126422
0925-2312/

space in both the source and target datasets. The task is common, but the source and target domains have different distributions. This specific case of transfer learning is known as *domain adaptation*.

The focus of this study is to use feature selection for domain adaptation, with the objective of identifying a common subset of features that optimizes classification performance in both the source and target datasets. Different approaches have been proposed to deal with this problem, among which is the use of Particle Swarm Optimization (PSO) [4], a swarm intelligence based algorithm that mimics the behaviour of a flock of birds, where multiple particles move around the search space trying to find a good solution, guided by a fitness function. In particular, there have been in the literature some attempts to use PSO approaches for domain adaptation [5,6], although these works used the classification accuracy to evaluate the goodness of the subsets of features, which is a very time-consuming approach.

In this paper we perform a comparison between using complexity metrics in the fitness function and using classifiers. The reason behind using data complexity metrics [7] is based on the assumption that a good choice of features should lower the complexity of the data, thus producing results that are independent of the classifier used in a subsequent phase, as well as reducing the computational time required.

The remainder of the paper is organized as follows: In Section 2, we provide a background on transfer learning and the Particle Swarm Optimization algorithm. In Section 3, we describe our proposed solution, including the classifiers and data complexity metrics used. In Section 4, we present the experimental design and datasets used. In Section 5, we present and discuss the results obtained. In Section 6, we present two case studies. Finally, in Section 7, we provide our conclusions and suggest directions for future work.

2. Background

2.1. Transfer learning

Transfer learning is a machine learning technique where a model trained on one task is re-purposed for a different (but related) task or domain. This allows us to reach high performance in our predictive models while lightening the computational cost of training them.

To give a formal definition of the transfer learning task, we must start by giving the definitions for domain and task [8]:

- A **domain** D consists of two components: a feature space $\mathscr{X}$ and a marginal probability distribution $P(X)$, where $X = \{x_1, \ldots, x_n\} \in \mathscr{X}$. This translates into $\mathscr{X}$ being all the features in our dataset and X a sample in that dataset, being x_i the value of feature i for such sample. Generally, when two domains are different, they can have different feature spaces or different marginal probability distributions.
  We can understand the marginal probability distribution $P(X)$ as the composition of the data, i.e. the number of samples belonging to each possible value of each feature. Therefore, we can formalize a domain like this: $D = \{\mathscr{X}, P(X)\}$.
- A **task** T has two components, a label space γ (all the possible labels for the samples) and an objective predictive function $f(\cdot)$. Therefore, we can formalize a task as $T = \{\gamma, f(\cdot)\}$.
  The predictive function $f(\cdot)$ consists of tuples $\{x_i, y_i\}$, where $x_i \in \mathscr{X}$ and $y_i \in \gamma$. This function is used to predict the label $f(x)$ of a new instance x. The predictive function would be perfect if it classified every sample correctly, and the intention is to learn a function that is as close as possible to being perfect.

Knowing this, we can give the definition for transfer learning: given a source domain D_s and a learning task for the source domain T_s, a target domain D_t and a learning task for the target domain T_t, transfer learning aims to help improve the learning of target predictive function $f_t(\cdot)$ in D_t using the knowledge in D_s and T_s, where $D_s \neq D_t$, or $T_s \neq T_t$.

The specific case of transfer learning we are exploring in this study is called *domain adaptation*. In domain adaptation problems, both source and target tasks are the same, but the domains, while having a common feature space, have differing marginal probability distributions.

2.2. Feature selection for transfer learning

The process of feature selection for predictive models consists in extracting a subset of features which are relevant for the given task, removing features which are redundant or do not provide meaningful information, aiming to make the problem simpler and improve performance by reducing noise in the data.

As we previously commented, domain adaptation is a specific kind of transfer learning problem in which the feature space is common both in the source and target data. There is a common task, but the source and target domains have different distributions. Feature selection can be applied to domain adaptation problems to extract a subset of features which minimizes the difference between the two distributions, thus helping to achieve good performance for the given task in both the source and target domain.

Different approaches and algorithms have been proposed to deal with the problem of feature selection for transfer learning, among which we can find Particle Swarm Optimization.

2.3. PSO: Particle Swarm Optimization

Particle Swarm Optimization (PSO) is an optimization algorithm that was first introduced by Kennedy and Eberhart in 1995 [4]. It is a population-based optimization algorithm inspired by the behaviour of swarms in nature, such as bird flocks or fish schools, which sets a swarm of particles that explore the search space in parallel. Each particle is a solution candidate, and consists of a position and velocity. There is also a function used to compute the value for each particle, called fitness function. Each particle records its best value obtained, while the best value reached by any particle is shared among all of them, in order to guide the particle to positions that improve their best value yet.

2.3.1. Sticky Binary Particle Swarm optimization (SBPSO)

PSO was originally proposed to solve continuous problems, but it has been adapted to also work with binary problems. This case is called Binary Particle Swarm Optimization, BPSO. This is the type of problem we face in this paper, as our search space consists of the available features, with two possible states, in our case whether a feature is selected or not.

In BPSO [9], the velocity of a particle is used to determine how likely it is for an element of the position (a feature, in our case) to toggle its value. The problem with BPSO is that it uses the same formula as PSO, which was designed for continuous search spaces, and therefore it does not work well.

Given these problems, Nguyen et al. [9] proposed a new version of the algorithm, coming up with a new and different concept of momentum. In a binary search space, the movement of the particles consists in toggling their elements, their bits. As it is not a continuous space, particles are not able to "keep moving" in a direction. Therefore, momentum is defined as the tendency to stay in the current position, property which they call stickiness. Because of that, this version of the algorithm is called Sticky Binary Particle Swarm Optimization (SBPSO).

In SBPSO, the concept of velocity is redefined using a vector of probabilities with as many elements as bits exist in our search space, where each of them represents the probability for the corresponding bit to toggle its value.

As mentioned before, instead (or as a new form) of momentum, we have stickiness, *stk*, which will be a value between 0 and 1 for each bit. The higher its value, the more likely it is for the bit to keep its value. Each time a bit flips its value, *stk* becomes 1 and starts decaying linearly to 0. Therefore, after a flip happens, it is less likely for that change to be reverted, translating into having some time to "look around in nearby positions".

We also have *maxLife*, the maximum number of iterations that a bit can go through without changing its value, and $currentLife_d$ is the number of iterations that have passed since the value of the bit d last changed. With these two variables, stk_d is computed as follows:

$$stk_d = 1 - \frac{currentLife_d}{maxLife} \tag{1}$$

Then, the probability of a bit toggling its value, p_d is computed like this:

$$p_d = i_s * (1 - stk_d) + i_p * |pbest_d - x_d| + i_g * |gbest - x_d|, \tag{2}$$

where $pbest_d$ is the best value reached by particle d, and *gbest* is the best value obtained globally, by any particle, and x_d is the position of the particle, that is, a vector with the value of each bit. We also have i_s, i_p and i_g, which are weights for each component. The values that we use and that are usually given to these are, respectively, 0.1154, 0.4423 and 0.4423, as indicated in the original paper [9].

Based on the new flipping probability vector computed, the new position of a particle is calculated as follows:

$$x_d^{t+1} = \begin{cases} 1 - x_d^t, & \text{if } random() \leqslant p_d \\ x_d^t, & \text{otherwise} \end{cases} \tag{3}$$

This way, as we mentioned earlier, we replace the velocity and momentum properties of continuous PSO with this concept of stickiness, which can be summed up as the tendency of moving around the search space, allowing particles to move around close positions avoiding big random jumps.

2.4. Data complexity measures

As mentioned in Section 2.3, the fitness function is used in the PSO algorithm to evaluate the goodness of each solution candidate, that is, to compute the value of the position of each particle. In this work, we explore the possibility of using complexity measures in the fitness function.

Complexity metrics in the field of machine learning can be used to asses the difficulty of a classification problem, trying to capture data particularities and identifying correlations with classification performance [10].

Basu and Ho [11] attributed the complexity of a classification problem to three main factors: the ambiguity of the classes, the sparsity and dimensionality of the data, and the complexity of the boundary separating the classes. They further gave three categories into which complexity measures could be divided [12]:

- **Measures of overlap in feature values from different classes**. These measures assess the effectiveness of individual feature dimensions or a combination of multiple dimensions in separating classes by examining the range and spread of values within each class and the overlap among different classes.
  Some measures of overlap that belong to this category are Maximum fisher's discriminant ratio (F1), Length of overlapping region (F2) and Maximum individual feature efficiency (F3), which are the three complexity measures used in this study, and provide the advantage of not relying on any classifier and being thus independent of the classifier used in a subsequent testing phase.
- **Measures of separability of classes**. These measures evaluate how well two classes can be separated by analyzing the class boundary and combining the contributions of individual feature dimensions into a single score, typically a distance metric.
  Examples of measures that fall into this category are Nonlinearity of linear classifier by LP (label powerset) [13] and the Average number of points per dimension.
- **Measures of geometry, topology, and density of manifolds**. These measures provide indirect characterizations of class separability by assuming that a class is composed of one or multiple manifolds that form the probability distribution of the class. The shape, position, and interconnectedness of these manifolds give an indication of how well the classes are separated, but they do not directly measure separability. Complexity measures in this category include the Fraction of points on class boundary and the Error rate of 1NN classifier, among others.

3. Proposed solution

We used Sticky Binary Particle Swarm Optimization to perform feature selection for domain adaptation. Taking an existing approach —using classification accuracy in the fitness function— as a starting point, we propose using complexity measures instead and performed a comparison between both approaches. Our motivation is based on the hypothesis that using complexity metrics will reduce the complexity of both source and target datasets, leading to competitive results which are simpler, faster and independent from the classifier used in a subsequent test phase.

With this aim, we took the solution proposed by Nguyen et al. [5] as a starting point, implemented and modified it to use data complexity metrics.

3.1. Original fitness function

Nguyen et al. [5] proposed the following fitness function:

$$Fitness = sw * srcErr + tw * tarErr + stw * diffST \tag{4}$$

where sw, tw and stw are weights, *srcErr* and *tarErr* are classification errors on source and target data, and *diffST* measures how different the marginal distributions of each data partition are. Let us see more in-depth how each one of these components are implemented.

- **Discriminability on the source domain** (*srcErr*): it is measured by the classification error rate in order to get a high discriminative ability in the source domain. It is calculated by applying 3-fold validation on the source dataset, using one of the folds for test, and the other two for training, and using a classifier (originally k-Nearest Neighbors).
- **Discriminability on the target domain** (*tarErr*): it is computed as the error rate on the labeled instances of the target data, using source data as training set, making use of the same classifier as in the previous component. This is based on the assumption that if the selected features have a good discriminative ability on the source domain, they should also have a high discriminability on the target domain.
- **Difference between marginal distributions** (*diffST*): finally, *diffST* tries to minimize the difference between the marginal distributions. For that Maximum Mean Discrepancy is used,

with Gaussian Radial Basis function as kernel function, as it is able to detect more types of dependence than linear or polynomial kernels.

3.2. Variants of the fitness functions proposed

For testing our hypothesis, we kept the last component, *diffST*, and tested six different variations of *srcErr* and *tarErr*, three using different complexity metrics and three using different classifiers.

For the options that utilize complexity metrics, we replaced *srcErr* and *tarErr* with the values obtained by applying the corresponding complexity metric to the source and target datasets. The three complexity metrics used in this study are Maximum Fisher's Discriminant Ratio (F1), Length of Overlapping Region (F2) and Maximum individual feature efficiency (F3). The values of F1 and F3 range from 0 to 1, whereas F2 gives values between 0 and the number of features in the dataset. Each of these metrics are computed as follows:

- **Maximum Fisher's Discriminant Ratio (F1)**: it measures the overlap between the values of the features in different classes, and it is based on the assumption that if at least one feature is able to discriminate the classes, the dataset can be considered simpler. It is computed as follows [7]:

$$F1 = \frac{1}{1 + max_{i=1}^{m} r_{f_i}}, \tag{5}$$

where r_{f_i} is the discriminant ratio for each feature f_i. Originally, the value of the maximum discriminant ratio is taken, but we use the inverse in order to get values between 0 and 1, having lower values indicate less complexity.
To calculate this metric, we used the multi-class formulation of r_{f_i} given by Orriols-Puig et al. [14]:

$$r_{f_i} = \frac{\sum_{j=1}^{n_c} \sum_{k=1, k\neq j}^{n_c} p_{c_j} p_{c_k} (\mu_{c_j}^{f_i} - \mu_{c_k}^{f_i})^2}{\sum_{j=1}^{n_c} p_{c_j} (\sigma_{c_j}^{f_i})^2}, \tag{6}$$

where p_{c_j} is the proportion of examples in class c_j, $\mu_{c_j}^{f_i}$ is the mean of feature f_i across examples of class c_j, and $\sigma_{c_j}^{f_i}$ is the standard deviation of those values.

- **Length of Overlapping Region (F2)**: this metric is a modification of the volume of overlapping region, which calculates the overlap of the distributions of the feature values within the classes, that is determined by finding, for each feature f_i, its maximum and minimum values in the classes, and the range of the overlapping interval is then calculated, normalized by the range of the values in both classes. Finally, the obtained values are multiplied.
In our case, instead of multiplying, we use the sum, which makes it the length instead of the volume. Using the product makes the value returned decrease greatly as dimensionality increases, resulting in a broad range of possible values that are close to zero, making the fitness function very difficult to balance.
Therefore, F2 can be defined as [7]:

$$\begin{aligned} F2 &= \sum_{i}^{m} \frac{overlap(f_i)}{range(f_i)} \\ &= \sum_{i}^{m} \frac{max\{0, minmax(f_i) - maxmin(f_i)\}}{maxmax(f_i) - minmin(f_i)} \end{aligned} \tag{7}$$

where:

$$\begin{aligned} minmax(f_i) &= min(max(f_i^{c_1}), max(f_i^{c_2})), \\ maxmin(f_i) &= max(min(f_i^{c_1}), min(f_i^{c_2})), \\ maxmax(f_i) &= max(max(f_i^{c_1}), max(f_i^{c_2})), \\ minmin(f_i) &= min(min(f_i^{c_1}), min(f_i^{c_2})), \end{aligned} \tag{8}$$

where $\max(f_i^{c_j})$ and $\min(f_i^{c_j})$ are the maximum and minimum values of each feature in class $c_j \in \{1, 2\}$, respectively.

- **Maximum individual feature efficiency (F3)**: F3 estimates the individual efficiency of each feature in separating the classes, and return the maximum value found among the m features. For each feature, the overlap of the values in the samples that belong to different classes is checked. The more overlap there is, the more ambiguous the classes are in this region, and the problem is therefore more complex. The problem can be considered simpler if there is at least one feature that shows low ambiguity between the classes.
We use the complement of this measure so that lower values equal less complexity. F3 is computed as [7]:

$$F3 = \min_{i=1}^{m} \frac{n_o(f_i)}{n} \tag{9}$$

where $n_o(f_i)$ gives the number of examples that are in the overlapping region for feature f_i. This is computed by the following equation:

$$n_o(f_i) = \sum_{j=1}^{n} I(x_{ji} > maxmin(f_i) \wedge x_{ji} < minmax(f_i)) \tag{10}$$

where I is a function that returns 1 if the argument is true, and 0 otherwise, while $maxmin(f_i)$ and $minmax(f_i)$ are the same ones defined for F2.

In the case of the variants of the fitness function using classifiers, we kept the same process described in Section 3.1 for *srcErr* and *tarErr*. We used k-Nearest-Neighbors (kNN), as was originally proposed in the work of Nguyen et al. [5], and added two classifiers, Support Vector Machine (SVM) and Naive-Bayes (NB):

- **k-Nearest Neighbors (kNN)** [15]: it is a non-parametric, lazy learning algorithm. It does not make any assumptions about the underlying data distribution. It stores all available instances and classifies new data points based on a similarity measure (usually Euclidean distance). When a new data point is encountered, the kNN algorithm looks at the k number of instances that are nearest to it, and classifies the new data point based on the majority class among those neighbors. In our case, we set parameter k to 1, so a new data point will be assigned the class that the closest instance belongs to.
- **Support Vector Machine (SVM)** [16]: it is a supervised learning algorithm that can be used for both classification and regression tasks. It works by finding a hyperplane in a high-dimensional feature space that maximally separates the data points of different classes. A good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class. In case of non-linearly separable data, the algorithm can be used with a kernel trick to transform the input data into a higher-dimensional space in which the classes become separable. In this study, the kernel function that is used is RBF (Radial Basis Function).
- **Naive-Bayes (NB)** [17]: it is a probabilistic classifier based on the Bayes' theorem, which states that the probability of an event (in this case, a data point belonging to a class) can be calculated by considering the prior probability of the event and the likelihood of the event given certain conditions (in this case, the

feature values of the data point). The Naive-Bayes classifier assumes that the features are independent, which allows for an efficient calculation of the likelihood. The classifier then calculates the probability of each class for the given data point and assigns the data point to the class with the highest probability.

3.3. Overall algorithm

Fig. 1 shows the general structure of the proposed methodology. First, we have Src_{train} and Tar_{train} which are the source and target data for training. Both of them are used in the feature selection process, using SBPSO, which is marked in blue. This gives a common feature space for both source and target data, which is applied to the test data ($Src\prime_{test}$ and $Tar\prime_{test}$). A classifier is trained using $Src\prime_{test}$, and then $Tar\prime_{test}$ is used for testing on the trained classifier, producing the corresponding classification performance.

4. Experiment design

We compared the performance of the SBPSO algorithm using the variants of the fitness function previously presented on four different datasets: Gas Sensor, Handwritten Digits, Prostate and TripAdvisor.

4.1. Datasets

The first dataset is Gas Sensor Drift [18], which consists of 13,910 instances with 127 features, and has six kind of gases, which are the classes to predict. The dataset is split in 10 batches. We used the first as source dataset, and the remaining ones as target datasets, which results in nine domain adaptation cases. Drift refers to the gradual and continuous deviation of a sensor's output from its expected or calibrated value over time. Therefore, as the different batches of this dataset are gathered over a period of 36 months, the distribution between them varies, making it a suitable problem for domain adaptation.

Two datasets are used for the Handwritten Digits problem: MNIST [19] and USPS [20]. Both of them consist of samples with 256 features (the pixels of 16x16 images) and 10 different classes, one for each digit. They have different distributions as they are collected from different sources: the USPS dataset is collected from scanned envelops of the US Postal Service, while the MNIST dataset comes from American Census Bureau employees and American high school students. 800 samples of the MNIST dataset were used and 720 of the USPS dataset. In this case, we use one of the datasets as source and the other one as target, which gives us two domain adaptation cases.

Third, we have the Prostate dataset [21], which is a microarray consisting of 134 samples with 12,600 features and two classes. The dataset comes in two splits: one contains 101 samples, which is used as source data, and the other contains 33 samples, used as target data. In this dataset, the source and target distributions differ significantly, as they were taken fom different experiments, and the target dataset has almost 10-fold difference in overall microarray intensity from the training data, making it challenging for machine learning models to reach a high performance.

The last dataset we used is the TripAdvisor [22] dataset, which contains restaurant reviews from different cities, with 209 features and two possible classes. We removed the user and restaurant IDs from the data, as we try to predict if a restaurant will be liked based only on its characteristics and price range. For this we use the datasets corresponding to Madrid and Barcelona cities, thus providing two domain adaptation cases. These datasets originally contain 561,588 and 404,947 respectively, but because of computational time restrictions, we used a random subsample of 1000 instances for each city.

In real-world problems, it is quite common that collecting labeled data is expensive and/or time consuming. It is in this scenario in which transfer learning is really useful, as it enables to use the acquired knowledge about the problem from the source (labeled) domain, to be transferred to the target (partially-labeled or unlabeled) domain. Therefore, we split the data, using 70% for training and the rest for testing, which in the case of the target data, although having the corresponding labels for getting the test accuracy, would act as unlabeled data.

4.2. Experiment settings

We divided the experimental phase in two steps. First, we carried out a comparison of the time per iteration required when using each fitness function. For that, we used only the first two components of the function (assigning 0.5 to each of its corresponding weights, *sw* and *tw*), as the third one is common for the six options, and we ran the algorithm for 50 iterations in each case.

Then we compared the performance in each of the problems. For that, each fitness function was fine tuned for each dataset in order to get the best possible combination of weights, which are shown in Table 1 and Table 2.

Each experiment was repeated 15 times with 15 different seeds, and each one ran up to 3000 iterations, stopping if it did not improve for 300 straight iterations. We also set a minimum percentage of features to select, 15%, in order to counter the tendency to select as few features as possible that can appear when using complexity metrics in some specific cases (the case study in Section 6.1 explores how modifying that parameter can affect performance in those cases).

5. Experimental results

This section presents the results obtained by testing the two types of penalty in the fitness function in the four problems mentioned above.

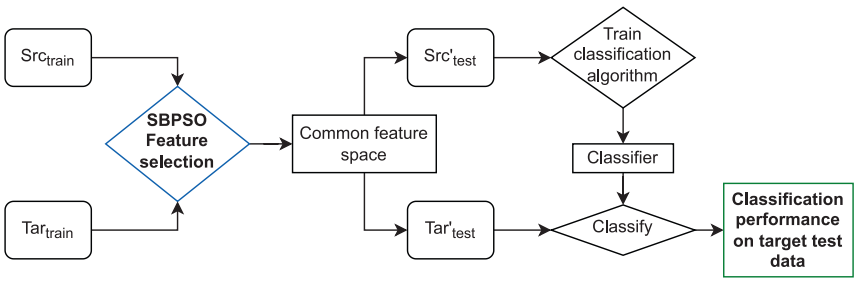


Fig. 1. Diagram of feature selection for domain adaptation using SBPSO.

Table 1
Weights used for each experiment using complexity metrics in the format (sw, tw, stw).

Dataset	F1	F2	F3
Gas Sensor	(0.1, 0.3, 0.6)	(0.4, 0.2, 0.4)	(0.2, 0.4, 0.4)
MNIST - USPS	(0.1, 0.3, 0.6)	(0.2, 0.4, 0.4)	(0.2, 0.2, 0.6)
USPS - MNIST	(0.6, 0.2, 0.2)	(0.5, 0.1, 0.4)	(0.2, 0.6, 0.2)
Prostate	(0.1, 0.3, 0.6)	(0.2, 0.2, 0.4)	(0.1, 0.3, 0.6)
Barcelona - Madrid	(0.2, 0.6, 0.2)	(0.001, 0.004, 0.95)	(0.1, 0.3, 0.6)
Madrid - Barcelona	(0.3, 0.3, 0.4)	(0.075, 0.025, 0.9)	(0.6, 0.2, 0.2)

Table 2
Weights used for each experiment using classifiers in the format (sw, tw, stw).

Dataset	kNN	SVM	NB
Gas Sensor	(0.1, 0.5, 0.4)	(0.3, 0.3, 0.4)	(0.1, 0.6, 0.3)
MNIST - USPS	(0.3, 0.7, 0.0)	(0.2, 0.4, 0.4)	(0.2, 0.4, 0.4)
USPS - MNIST	(0.2, 0.6, 0.2)	(0.2, 0.4, 0.4)	(0.3, 0.3, 0.4)
Prostate	(0.6, 0.2, 0.2)	(0.3, 0.3, 0.4)	(0.3, 0.3, 0.4)
Barcelona - Madrid	(0.3, 0.1, 0.6)	(0.3, 0.3, 0.4)	(0.1, 0.3, 0.6)
Madrid - Barcelona	(0.4, 0.2, 0.4)	(0.33, 0.33, 0.33)	(0.1, 0.3, 0.6)

5.1. Time comparison

First, we carried out a comparison of the time per iteration required when using each fitness function, as described in Section 4.2, and the results are shown in Table 3.

The values shown are the mean obtained. As expected, the fitness functions using complexity metrics are notably faster than the ones using classifiers. It stands out that, as the number of features in the dataset grows, SVM and Naive Bayes become slower. Using the Naive Bayes fitness function with the Prostate dataset, the time required was too high to be a good choice for the PSO algorithm. Because of that and the time limitation it supposes, the Naive Bayes fitness function was not used with the Prostate Dataset in the following experiments. As stated in Section 4.1, we used only 1000 samples from each dataset of the TripAdvisor problem, because it became extremely time consuming when using more samples. Using one third of the data, the average time per iteration was around 4 s for the complexity metrics, but it took 47s with NB, 108s with kNN and 768s with SVM.

After the time per iteration tests, we proceeded to perform a performance comparison, with the settings presented in Section 4.2. For each dataset, we show a table displaying the mean results obtained with each fitness function. We also performed a Friedman statistical test to assess the significance of differences between the performance of the different options, followed by a Nemenyi test. The results of these tests are presented in critical-difference (CD) diagrams, where the best performing options are on the left, and those with non-significant differences are connected by a straight line.

5.2. Performance on Gas Sensor

The results for the first problem, Gas Sensor, are shown in Table 4. It shows the mean results of batches. The complete results for each individual batch are provided in Table A.

As we can see, the general trend is that the best results are obtained using the same classifier both in the fitness function and the test phase. This shows us that, as expected, these results are dependent on the classifier. This dependency is not present using complexity metrics in the fitness function. Fig. 2 shows the CD diagrams for the results obtained when testing in each of the classifiers, and it also reflects this dependency, as the option that uses the same classifier both in the fitness function and on the test phase performs the best in each case, and shows significant difference with the second best performing option.

The variants of the fitness function that use classifiers generally get better results, although the option using F2 gets close in some cases, and shows no critical difference when testing with SVM, and even performs slightly better than SVM and kNN when testing on NB, according to the statistical test. Concerning the number of features selected, complexity metrics select fewer than classifiers in all experiments.

5.3. Performance on Handwritten Digits

For the Handwritten Digits problem, Table 5 shows the results when using USPS as source data and MNIST as target data. In this case, the tendency is similar to the previous dataset, with classifiers generally getting higher accuracy than complexity metrics, but the latter selecting fewer features. We can see two exceptions: on the one hand, the fitness function that uses kNN performs worse than the other classifiers, even when using kNN in the subsequent test phase. On the other hand, F1 selects a surprisingly high number of features, while F2 and F3 do select less than classifiers, as expected.

Regarding the complexity metrics, F3 is the one that performs better. We can see on Fig. 3 that, with every classifier, F3 always shows no critical difference with respect to at least two of the classifiers.

Table 6 shows the other case of Handwritten Digits: using MNIST as source data and USPS as target. The accuracies obtained are higher than in the previous case, but the tendency remains the same, keeping the high number of selected features when using F1.

If we look at the statistical tests in Fig. 4, we find that F3 is also competitive in this case, and when testing on kNN, F1 shows no critical difference with the options that use classifiers.

In both cases of the Handwritten Digits problem, F1 and F3 reach higher performance than F2. We have seen that F3 is competitive in terms of performance, and it has the advantage of selecting a smaller number of variables, making it a potential alternative to traditional classifiers in the penalty function.

5.4. Performance on Prostate

Table 7 shows the results using the Prostate dataset, and Fig. 5 shows the CD graph of the statistical tests.

This case breaks with the tendency seen in the previous problems, as it shows very similar results with all the variants of the fitness function, both in terms of performance and number of features selected. This is a result of this domain adaptation problem being a particularly difficult one. The source and target datasets were extracted from different experiments, and therefore

Table 3
Mean time (in seconds) per iteration for each dataset and fitness function. Lowest results are marked in bold.

Dataset	F1	F2	F3	kNN	SVM	NB
Gas Sensor	0.165	0.171	**0.164**	1.117	1.558	0.529
Handwritten Digits	**0.182**	0.184	0.195	1.098	2.068	1.255
Prostate	**0.311**	0.316	0.345	1.267	2.053	19.618
TripAdvisor	0.182	**0.180**	**0.180**	1.317	0.973	0.539

Table 4
Summary of the resulting accuracy (Acc.) with the Gas Sensor dataset. The best result for each classifier is marked in bold.

Fitness	kNN Acc.	SVM Acc.	NB Acc.	%Features
F1	0.487	0.401	0.262	15.777
F2	0.505	0.463	0.323	15.748
F3	0.479	0.432	0.321	16.663
kNN	**0.689**	0.516	0.301	21.143
SVM	0.602	**0.551**	0.308	20.475
NB	0.583	0.505	**0.377**	25.793

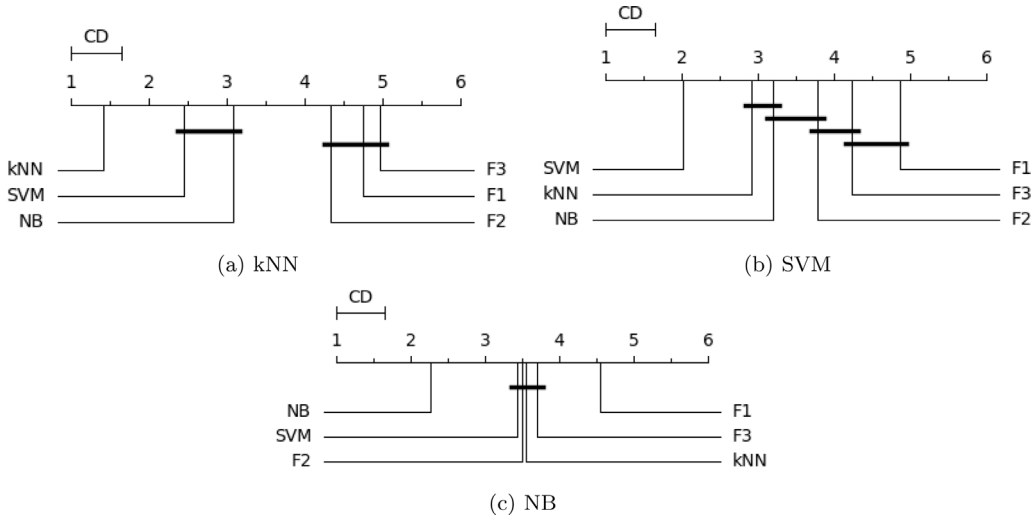


Fig. 2. CD diagram of the results with the Gas Sensor dataset, testing on each classifier.

Table 5
Resulting accuracy (Acc.) with USPS as source data and MNIST as target data. The best result for each classifier is marked in bold.

Fitness	kNN Acc.	SVM Acc.	NB Acc.	%Features
F1	0.257	0.241	0.247	68.945
F2	0.222	0.259	0.215	15.625
F3	0.299	0.300	0.326	27.995
kNN	0.383	0.402	0.351	45.833
SVM	**0.418**	**0.497**	0.343	45.052
NB	0.408	0.456	**0.419**	43.815

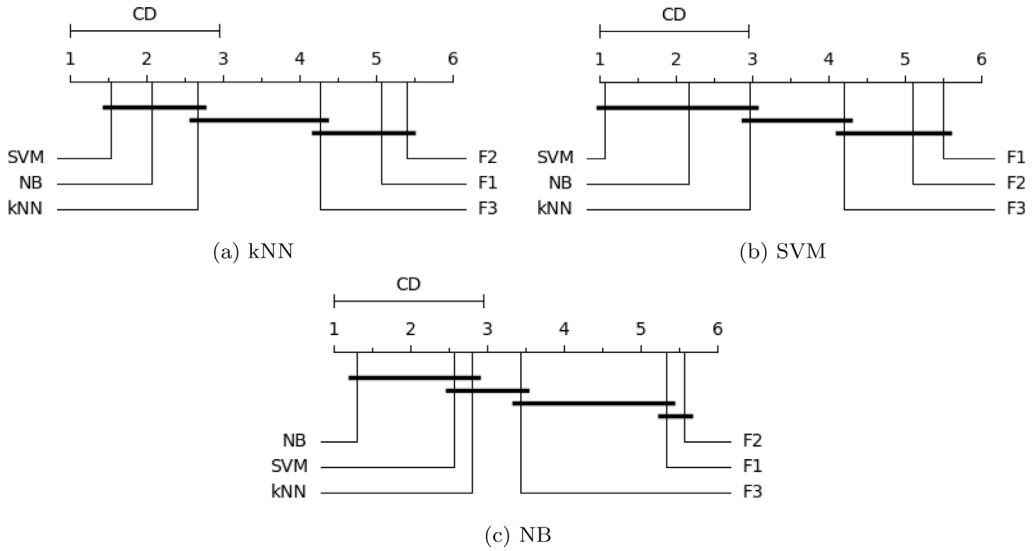


Fig. 3. CD diagram of the results using USPS as source and MNIST as target, testing on each classifier.

Table 6
Resulting accuracy (Acc.) with MNIST as source data and USPS as target data. The best result for each classifier is marked in bold.

Fitness	kNN Acc.	SVM Acc.	NB Acc.	%Features
F1	0.547	0.348	0.455	67.513
F2	0.289	0.242	0.262	15.820
F3	0.533	0.367	0.490	32.682
kNN	**0.600**	0.433	0.528	51.758
SVM	0.574	**0.520**	0.527	48.438
NB	0.584	0.456	**0.537**	51.107

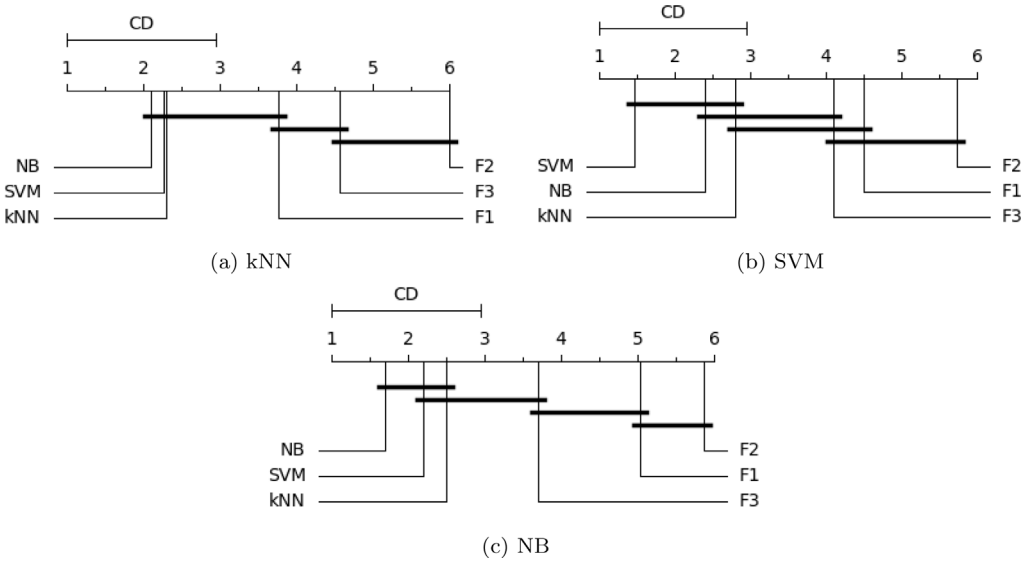


Fig. 4. CD diagram of the results using MNIST as source and USPS as target, testing on each classifier.

Table 7
Resulting accuracy (Acc.) with the Prostate dataset. The best result for each classifier is marked in bold.

Fitness	kNN Acc.	SVM Acc.	NB Acc.	%Features
F1	0.600	**0.300**	0.350	53.48
F2	0.600	**0.300**	0.350	44.56
F3	0.700	**0.300**	0.350	53.30
kNN	**0.717**	**0.300**	**0.367**	50.91
SVM	0.583	**0.300**	0.350	53.26

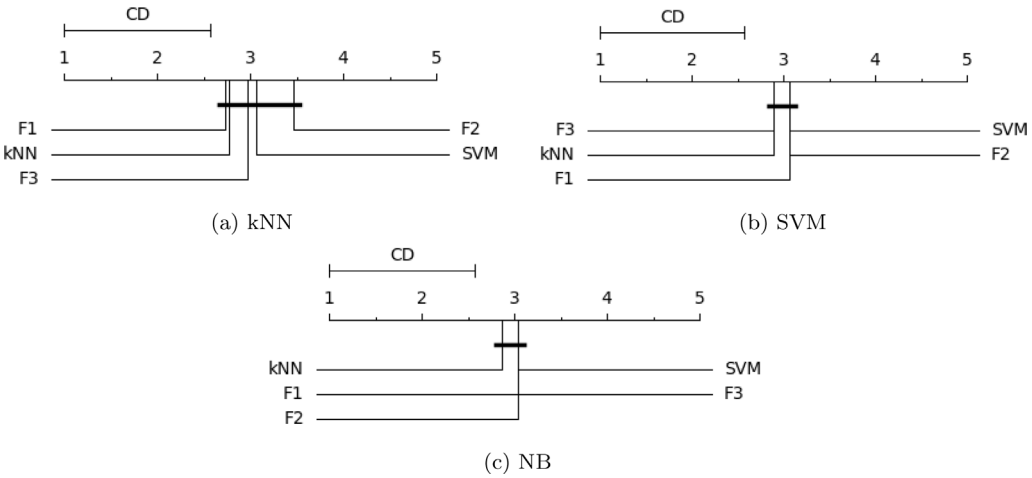


Fig. 5. CD diagram of the results with the Prostate dataset, testing on each classifier.

Table 8
Resulting accuracy (Acc.) testing on SVM with linear kernel with the Prostate dataset. The best result is marked in bold.

Fitness	SVM (linear kernel) Acc.	%Features
F1	0.493	53.44
F2	**0.500**	44.61
F3	0.453	53.07
kNN	0.487	50.79
SVM	0.493	53.17

their class distributions differ greatly: 49%-51% in the source dataset, 26%-74% in the target dataset. This can lead to some classifiers assigning all the samples to one of the classes [23], as it is the case with SVM in our experiments.

Further experimentation to address this specific case proved that using a linear kernel instead of RBF is more appropriate for this dataset, making predictions for different classes instead of predicting all samples as the same class, and therefore obtaining a higher accuracy. These results are showed in Table 8.

As we can see, F2 is the fitness function option that performs best, both in terms of accuracy and number of features selected. Therefore, we can conclude that, for this dataset, even with its high level of difficulty, complexity metrics are as competitive as classifiers.

5.5. Performance on TripAdvisor

Finally, Table 9 shows the results obtained using the Barcelona dataset as source and the Madrid dataset as target, and the CD diagrams of the statistical tests are shown in Fig. 6.

We can clearly see that, in terms of accuracy, F1 and F3 are as competitive, if not better, than the options that use classifiers. As we can see in the statistical tests, F3 is the best option when testing both in the SVM and NB classifiers. With respect to the number of features selected, we see the same tendency as in the Handwritten Digits problem: F1 selects a high number of features, but F2 and F3 select fewer than the classifiers.

The results obtained in the other case, having Madrid as source data and Barcelona as target, are shown in Table 10, and its statistical tests are shown in Fig. 7.

The results are similar to the previous case, being F2 the only option that shows a significant difference with other variants to calculate the penalty function, whereas F1 and F3 are as competitive as classifiers, and the number of features selected also follows the same trend.

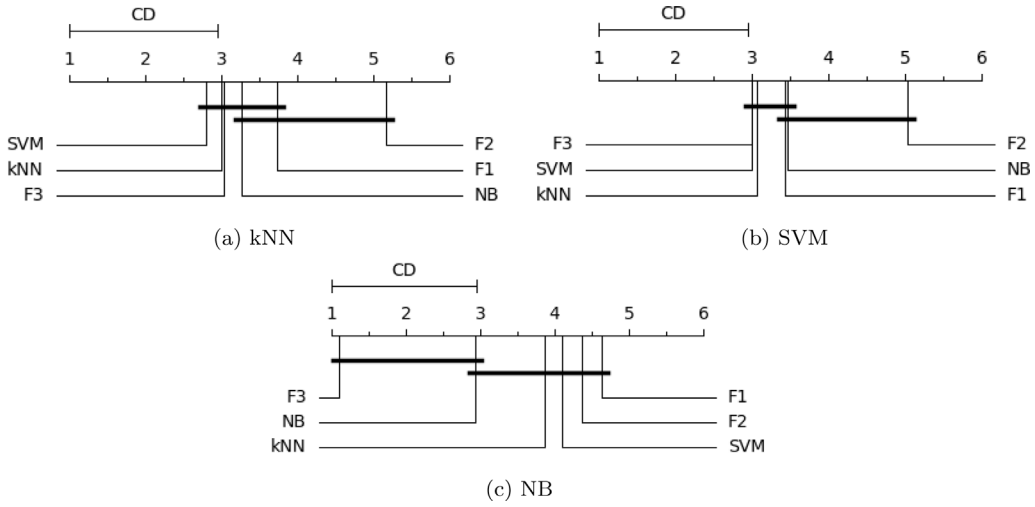
Further analyzing the problem, we find that the dataset is not balanced, with a 86.44% of the labels being 1, while the rest are 0. This supposes a challenge to overcome when learning a model, as there is a risk that it ends up predicting every sample as belonging to the majority label. This is the case with the F2 option, and the reason of it not performing as well as the rest. It also happens in almost every case in the test phase when using NB. As future work, we plan to apply sampling methods in order to solve the imbalance of the data.

6. Case studies

We carried out two case studies to explore two specific aspects of our experiments: seeing how the minimum percentage of features to select affects performance, and the effects of splitting the data differently.

Table 9
Resulting accuracy (Acc.) with the TripAdvisor dataset, using Barcelona as source and Madrid as target. The best result for each classifier is marked in bold.

Fitness	kNN Acc.	SVM Acc.	NB Acc.	%Features
F1	0.987	0.988	0.868	65.27
F2	0.870	0.891	0.887	46.00
F3	0.988	0.989	**0.976**	33.13
kNN	**0.991**	0.990	0.879	67.07
SVM	0.989	**0.991**	0.887	67.53
NB	0.989	0.990	0.878	52.20



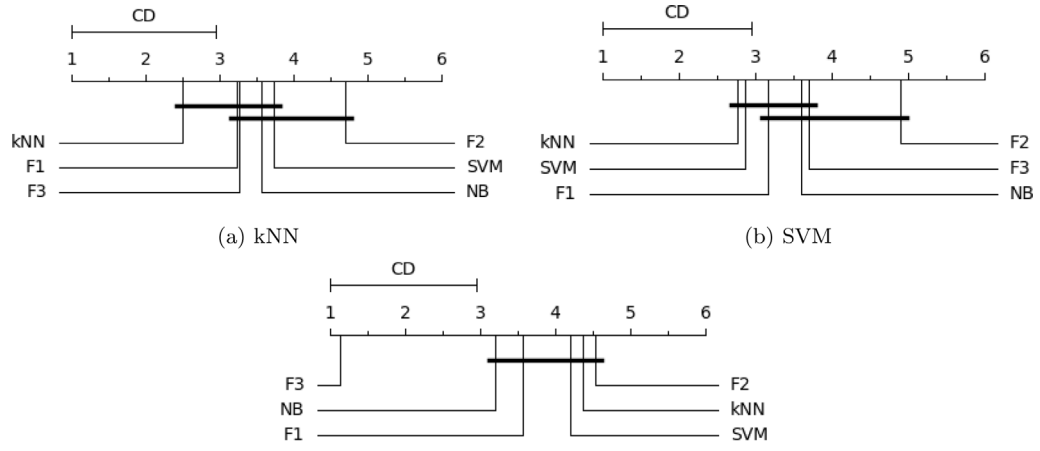
(a) kNN (b) SVM

(c) NB

Fig. 6. CD diagram of the results using Barcelona as source and Madrid as target, testing on each classifier.

Table 10
Resulting accuracy (Acc.) with the TripAdvisor dataset, using Madrid as source and Barcelona as target. The best result for each classifier is marked in bold.

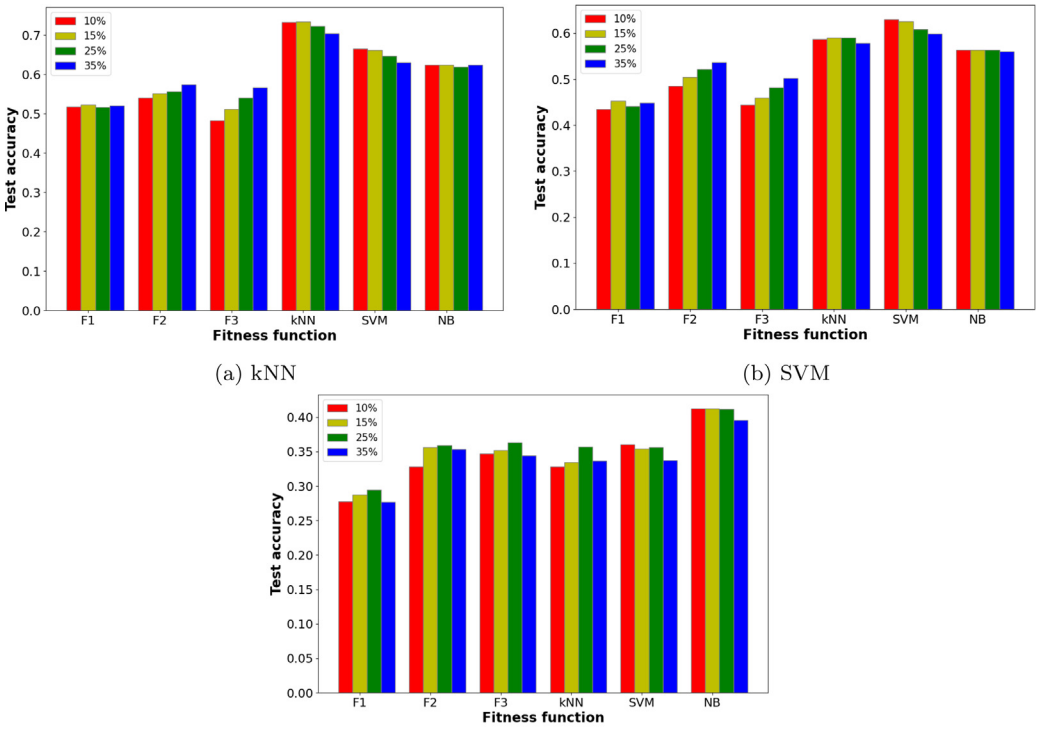
Fitness	kNN Acc.	SVM Acc.	NB Acc.	%Features
F1	0.991	0.993	0.878	64.20
F2	0.815	0.909	0.887	46.93
F3	0.992	0.992	**0.985**	34.07
kNN	**0.996**	**0.994**	0.875	64.33
SVM	0.991	0.993	0.880	63.13
NB	0.990	0.992	0.884	57.27



(a) kNN (b) SVM

(c) NB

Fig. 7. CD diagram of the results using Madrid as source and Barcelona as target, testing on each classifier.



(a) kNN (b) SVM

(c) NB

Fig. 8. Mean accuracy with each fitness function when testing on kNN, SVM and NB, using a minimum of 10%, 15%, 25% and 35% of features. Each graph has a different range of values in the vertical axis.

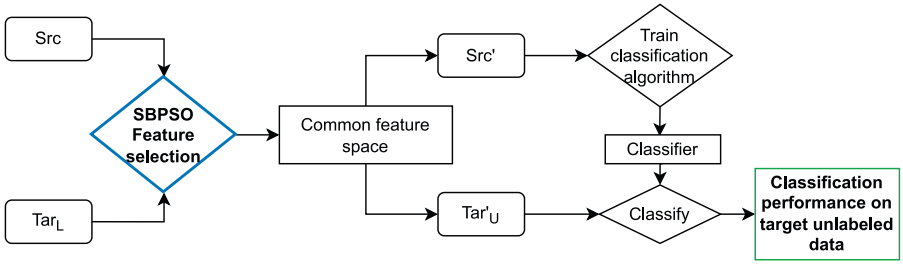


Fig. 9. Diagram of feature selection for domain adaptation using SBPSO, splitting the target data as labeled-unlabeled.

Table 11
Weights used with complexity metrics for each experiment in the format (sw, tw, stw).

Dataset	F1	F2	F3
MNIST - USPS	(0.6, 0.2, 0.2)	(0.5, 0.1, 0.4)	(0.2, 0.6, 0.2)
USPS - MNIST	(0.1, 0.3, 0.6)	(0.2, 0.4, 0.4)	(0.2, 0.2, 0.6)

Table 12
Weights used with classifiers for each experiment in the format (sw, tw, stw).

Dataset	kNN	SVM	NB
MNIST - USPS	(0.2, 0.6, 0.2)	(0.2, 0.4, 0.4)	(0.3, 0.3, 0.4)
USPS - MNIST	(0.3, 0.7, 0.0)	(0.2, 0.4, 0.4)	(0.2, 0.4, 0.4)

6.1. Effect of varying the minimum percentage of features to select

As a tendency to select as few features as possible can arise when using complexity metrics, we performed a case study varying the minimum percentage of features to select to check how it can affect performance. We did an experiment setting the minimum percentage of features to select to 10%, 25% and 35% (apart from the previously tested 15%), and tested on the Gas Sensor dataset, as this was the problem where this tendency appeared when using complexity metrics. We used batches 2, 3, 4, 5, 6, 8 and 9 due to time limitation, but those six batches can give us representative results. The weights of each case of the fitness function are the ones showed in Section 4.2.

To compare the results of each option, Fig. 8 shows bar diagrams with the mean accuracy when classifying with kNN, SVM and NB.

When utilizing the fitness functions that use classifiers, there is no clear conclusion to be extracted, as expected, as classifiers do not show the tendency to select as few features as possible. On the contrary, it does happen with complexity metrics. We can see that, when classifying with kNN and SVM, F2 and F3 improve as we set a higher minimum percentage of features to select, while F1 gets similar results in all cases. When classifying with NB, it seems that setting the percentage of features to 15% or 25% achieves better results than 10% or 35%.

From the results obtained, there is no general rule on what minimum percentage of features to select. It can indeed improve performance depending on the fitness function and classifier used for testing. We can therefore conclude that, depending on the problem, if this tendency to select as few features as possible arises, a trade-off can be made between number of selected features and performance in order to get slightly better results.

6.2. Splitting the data as labeled-unlabeled

In our previous experiments, as explained in Section 4.1, we make training and testing splits in both source and target datasets. In real world problems, we can encounter the situation of having a labeled set of data, that would be our source data, and target data that is entirely or partially unlabeled due to it being expensive to label. This approach can be found in other works [5] [24]. This case study tries to represent this situation, in which would like to take advantage of the already labeled dataset of a similar domain.

This means that, in the experiments carried out, our source data is completely labeled, and is used both in the training and test phases, while for the target data we split it as if one third of the data was labeled, and the rest was unlabeled. In the training phase, we used the labeled target data, and then, on the test phase, we predicted the labels of the unlabeled target data (it acts as "unlabeled", even though we have their real labels and are therefore able to get the real accuracy). The general structure of this method is shown in Fig. 9:

In contrast to the approach detailed in Section 5, this method utilizes the complete source dataset, *Src*, for both performing feature selection and for training the classifier used for predicting new samples in the test phase, using only the selected features in this case ($Src\prime$). Additionally, we used a reduced subset of the target data; it is divided into a labeled subset (Tar_L) used for feature selection and an unlabeled subset used with the selected features ($Tar\prime_U$) for testing.

For these experiments, we used the Handwritten Digits problem, set the minimum percentage of features to select to 15%, and used the weights shown in Table 11 and Table 12.

Table 13 shows the results using USPS as source data and MNIST as target, and Fig. 10 shows CD graph of their statistical tests. Table 14 and Fig. 11 show the corresponding ones for the experiments using MNIST as source data and USPS as target.

Table 13
Resulting accuracy (Acc.) splitting the data as labeled - unlabeled, with USPS as source data and MNIST as target data. The best result for each classifier is marked in bold.

Fitness	kNN Acc.	SVM Acc.	NB Acc.	%Features
F1	0.302	0.236	0.205	67.60
F2	0.172	0.189	0.162	17.81
F3	0.358	0.296	0.289	33.20
kNN	**0.456**	0.403	0.368	45.47
SVM	**0.456**	**0.550**	0.423	45.10
NB	0.450	0.476	**0.478**	44.09

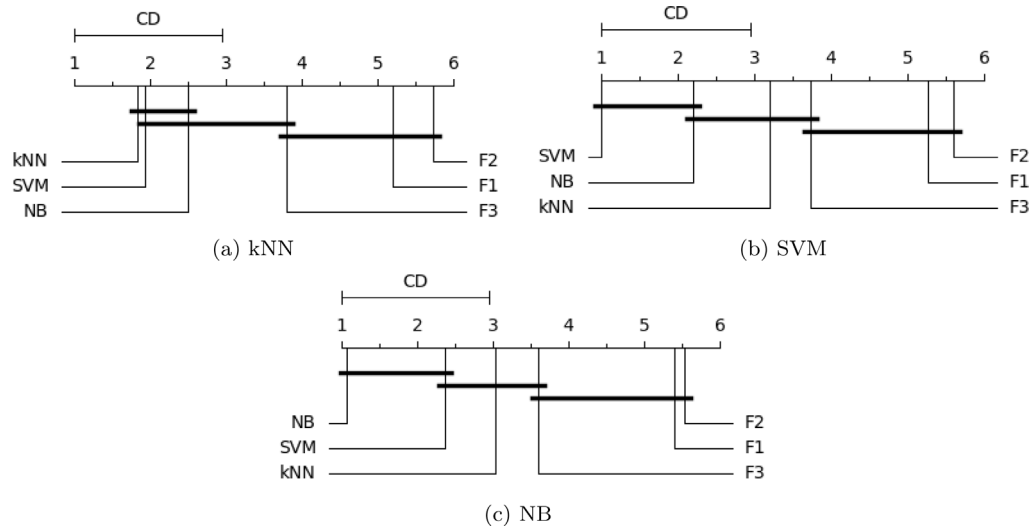


Fig. 10. CD diagram of the results splitting the data as labeled-unlabeled, using USPS as source and MNIST as target, testing on each classifier.

Table 14
Resulting accuracy (Acc.) splitting the data as labeled - unlabeled, with MNIST as source data and USPS as target data. The best result for each classifier is marked in bold.

Fitness	kNN Acc.	SVM Acc.	NB Acc.	%Features
F1	0.614	0.450	0.481	68.39
F2	0.332	0.279	0.299	16.48
F3	0.566	0.451	0.475	31.74
kNN	0.646	0.506	0.545	51.61
SVM	**0.647**	**0.649**	0.585	48.28
NB	0.635	0.545	**0.586**	52.19

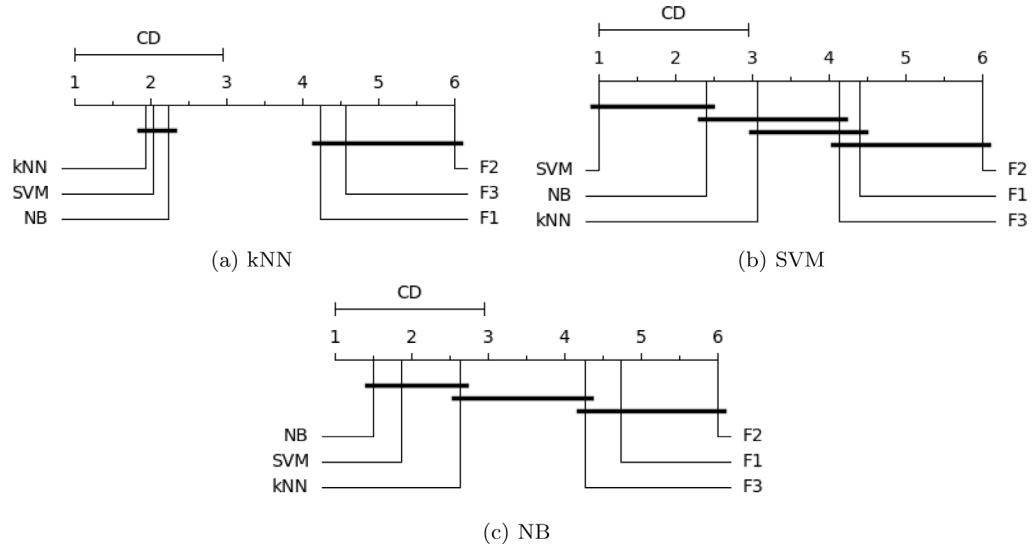


Fig. 11. CD diagram of the results splitting the data as labeled-unlabeled, using MNIST as source and USPS as target, testing on each classifier.

Comparing the results to those presented in Section 5.3, we can see that the accuracy achieved by the options using classifiers in the fitness function generally shows a little improvement. This does not happen in every case using complexity metrics. For example, using USPS as source data, F2 achieves lower performance, but improves when using MNIST as source. It also depends on the case with F1 and F3; it shows no generalized improvement or worsening.

If we compare the CD diagrams of both cases, when using USPS as source data the only improvement shown by complexity metrics is that, when testing on kNN, F3 shows critical difference only with one of the classifiers, while in Section 5.3 there is significant differ-

ence with two. The opposite happens when using MNIST as source data, where in every case there are more cases of significant difference between classifiers and complexity metrics, specially when testing on kNN. In terms of number of features selected, the results are practically the same.

From this case study, we can conclude that, although in real world problems we may not be able to decide how to split the data, as they will be given to us in a specific way, when compared to how we have done it in the rest of the paper, this way of splitting data can lead to an improvement in performance specially when using classifiers, while this improvement is not generalized when using complexity metrics. Therefore, when the data is splitted this way, it may be more advisable to use classifiers rather than complexity measures.

7. Conclusions

Transfer learning is a recent trend in machine learning and a prolific field of research. In this paper we focused on domain adaptation, with the goal of obtaining a common representation of meaningful features both for the source and target data. In particular, we proposed the use of PSO to find the relevant features. We performed a comparison between two approaches, an already existing one, using classification performance in the fitness function to evaluate the subsets of features, and a novel one, using data complexity metrics for this task, under the hypothesis that a subset of data with the correct features results in a less complex dataset.

After carrying out experiments over four problems suitable for transfer learning evaluation, we demonstrated that the use of complexity measures in the fitness function led to a reduction in the computational time and number of features selected. Depending on the problem, complexity metrics also achieve competitive performance, making the trade-off between performance and speed feasible in some cases. Moreover, an added advantage of the use of complexity measures is that it is classifier-independent, not conditioning the choice of a given classifier in a posterior classification stage.

Further research would be recommended in order to explore and compare different approaches. We suggest experimenting with different datasets, exploring new fitness functions, combining different metrics or classifiers, or trying other feature utility metrics, such as correlation or mutual information.

Data availability

Datasets are public

Declaration of Competing Interest

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

Acknowledgments

This work was supported by CITIC, as Research Center accredited by Galician University System, which is funded by "Consellería de Cultura, Educación e Universidade from Xunta de Galicia", supported in an 80% through ERDF Funds, ERDF Operational Programme Galicia 2014–2020, and the remaining 20% by "Secretaría Xeral de Universidades" (Grant ED431G 2019/01). It was also partially funded by Xunta de Galicia/FEDER-UE under Grant ED431C 2022/44; Ministerio de Ciencia e Innovación MCIN/AEI/10.13039/501100011033 and "NextGenerationEU"/PRTR under Grants [PID2019-109238 GB-C22; TED2021-130599A-I00].

Appendix A. Complete Gas Sensor results

Table A.15 shows the average results for each individual batch of the Gas Sensor dataset, which are summarized in Table 4.

Table A.15
Resulting accuracy (Acc.) with each batch of the Gas Sensor dataset. The best result for each batch and classifier is marked in bold.

Dataset	Fitness	kNN Acc.	SVM Acc.	NB Acc.	%Features
Batch 2	F1	0.526	0.339	0.238	15.75
Batch 2	F2	0.642	0.585	0.261	15.75
Batch 2	F3	0.672	0.520	0.227	17.32
Batch 2	kNN	**0.849**	0.540	0.193	18.67
Batch 2	SVM	0.833	**0.671**	0.249	18.45
Batch 2	NB	0.823	0.610	**0.391**	22.61
Batch 3	F1	0.607	0.464	0.250	15.75
Batch 3	F2	0.674	0.596	**0.468**	15.75
Batch 3	F3	0.616	0.555	0.387	18.45
Batch 3	kNN	**0.789**	0.564	0.192	21.93
Batch 3	SVM	0.724	**0.638**	0.392	19.91
Batch 3	NB	0.733	**0.638**	**0.468**	28.80
Batch 4	F1	0.411	0.399	0.360	15.75
Batch 4	F2	0.515	0.494	0.414	15.75
Batch 4	F3	0.390	0.312	0.426	15.97
Batch 4	kNN	**0.688**	0.628	0.500	22.16
Batch 4	SVM	0.565	**0.640**	0.399	20.25
Batch 4	NB	0.568	0.595	**0.527**	24.97
Batch 5	F1	0.656	0.695	**0.586**	15.75
Batch 5	F2	0.656	0.663	0.530	15.75
Batch 5	F3	0.610	0.644	0.564	16.54
Batch 5	kNN	**0.748**	0.738	0.450	16.42
Batch 5	SVM	0.719	**0.741**	0.523	16.42
Batch 5	NB	0.671	0.731	0.535	22.72
Batch 6	F1	0.665	0.399	0.154	15.75
Batch 6	F2	0.580	0.416	0.231	15.75
Batch 6	F3	0.611	0.464	0.261	16.20

(*continued on next page*)

Table A.15 (*continued*)

Dataset	Fitness	kNN Acc.	SVM Acc.	NB Acc.	%Features
Batch 6	kNN	**0.783**	0.431	0.257	22.38
Batch 6	SVM	0.678	**0.584**	0.287	20.13
Batch 6	NB	0.640	0.451	**0.336**	28.57
Batch 7	F1	0.343	0.247	0.166	15.75
Batch 7	F2	0.434	0.308	0.258	15.75
Batch 7	F3	0.411	0.333	0.211	16.93
Batch 7	kNN	**0.587**	0.329	0.188	20.73
Batch 7	SVM	0.516	**0.343**	0.190	18.90
Batch 7	NB	0.440	0.305	**0.279**	25.07
Batch 8	F1	0.290	0.314	0.106	15.75
Batch 8	F2	0.290	0.432	0.303	15.75
Batch 8	F3	0.392	0.458	0.337	16.54
Batch 8	kNN	**0.623**	**0.604**	**0.386**	21.65
Batch 8	SVM	0.508	0.572	0.290	19.03
Batch 8	NB	0.415	0.515	0.286	26.12
Batch 9	F1	0.552	0.518	0.305	16.01
Batch 9	F2	0.447	0.430	0.258	15.75
Batch 9	F3	0.287	0.336	0.303	16.01
Batch 9	kNN	**0.652**	**0.574**	0.320	20.08
Batch 9	SVM	0.486	0.526	0.255	24.80
Batch 9	NB	0.532	0.461	**0.331**	28.87
Batch 10	F1	0.334	0.238	0.189	15.75
Batch 10	F2	0.303	0.246	0.181	15.75
Batch 10	F3	0.323	**0.266**	0.169	16.01
Batch 10	kNN	**0.479**	0.233	**0.223**	26.25
Batch 10	SVM	0.388	0.248	0.184	26.38
Batch 10	NB	0.426	0.239	0.243	24.41

References

[1] I. Guyon, S. Gunn, M. Nikravesh, L. Zadeh, Feature extraction: foundations and applications, Vol. 207, Springer, 2008.
[2] Y. Yang, J.O. Pedersen, A comparative study on feature selection in text categorization, in: Icml, Vol. 97, Citeseer, 1997, p. 35.
[3] V. Bolon-Canedo, B. Remeseiro, Feature selection in image analysis: a survey, Artificial Intelligence Review 53 (4) (2020) 2905–2931.
[4] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proc. of ICNN'95 Int. Conf. on Neural Networks, Vol. 4, IEEE, 1995, pp. 1942–1948.
[5] B. Nguyen, B. Xue, P. Andreae, A particle swarm optimization based feature selection approach to transfer learning in classification, in: Proc. of Genetic and Evolutionary Computation Conf., 2018, pp. 37–44.
[6] H. Dhrif, V. Bolón-Canedo, S. Wuchty, Gene subset selection for transfer learning using bilevel particle swarm optimization, in: 2020 19th IEEE Int. Conf. on Machine Learning and Applications (ICMLA), IEEE, 2020, pp. 1317–1323.
[7] A. Lorena, L. Garcia, J. Lehmann, M. Souto, T. Ho, How complex is your classification problem? a survey on measuring classification complexity, ACM Computing Surveys (CSUR) 52 (5) (2019) 1–34.
[8] S.J. Pan, Q. Yang, A survey on transfer learning, IEEE Transactions on knowledge and data engineering 22 (10) (2009) 1345–1359.
[9] B.H. Nguyen, B. Xue, P. Andreae, A novel binary particle swarm optimization algorithm and its applications on knapsack and feature selection problems, in: Intelligent and Evolutionary Systems, Springer, 2017, pp. 319–332.
[10] L. Morán-Fernández, V. Bolón-Canedo, A. Alonso-Betanzos, Can classification performance be predicted by complexity measures? a study using microarray data, Knowledge and Information Systems 51 (3) (2017) 1067–1090.
[11] T.K. Ho, M. Basu, Complexity measures of supervised classification problems, IEEE transactions on pattern analysis and machine intelligence 24 (3) (2002) 289–300.
[12] M. Basu, T.K. Ho, Data complexity in pattern recognition, Springer Science & Business Media, 2006.
[13] A. Hoekstra, R.P. Duin, On the nonlinearity of pattern classifiers, in: Proceedings of 13th international conference on pattern recognition, Vol. 4, IEEE, 1996, pp. 271–275.
[14] A. Orriols-Puig, N. Macia, T.K. Ho, Documentation for the data complexity library in c++, Universitat Ramon Llull, La Salle 196 (1–40) (2010) 12.
[15] L.E. Peterson, K-nearest neighbor, Scholarpedia 4 (2) (2009) 1883.
[16] W.S. Noble, What is a support vector machine?, Nature biotechnology 24 (12) (2006) 1565–1567
[17] I. Rish, et al., An empirical study of the naive bayes classifier, in: IJCAI 2001 workshop on empirical methods in artificial intelligence, Vol. 3, 2001, pp. 41–46.
[18] A. Vergara, S. Vembu, T. Ayhan, M. Ryan, M. Homer, R. Huerta, Chemical gas sensor drift compensation using classifier ensembles, Sensors and Actuators B: Chemical 166 (2012) 320–329.
[19] L. Deng, The mnist database of handwritten digit images for machine learning research, IEEE Signal Processing Magazine 29 (6) (2012) 141–142.
[20] J.J. Hull, A database for handwritten text recognition research, IEEE Transactions on Pattern Analysis and Machine Intelligence 16 (5) (1994) 550–554, https://doi.org/10.1109/34.291440.
[21] Kent ridge bio-medical dataset repository, https://leo.ugr.es/elvira/DBCRepository/, accessed October 2022.
[22] P. Pérez-Núñez, O. Luaces, J. Díez, B. Remeseiro, A. Bahamonde, Tripadvisor restaurant reviews (dec 2021). doi:10.5281/zenodo.5644892. URL: doi:10.5281/zenodo.5644892.
[23] V. Bolón-Canedo, L. Morán-Fernández, A. Alonso-Betanzos, An insight on complexity measures and classification in microarray data, in: in: 2015 International Joint Conference on Neural Networks (IJCNN) IEEE, 2015, pp. 1–8.
[24] S. Uguroglu, J. Carbonell, Feature selection for transfer learning, in: Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2011, Athens, Greece, September 5–9, 2011, Proceedings, Part III 22, Springer, 2011, pp. 430–442.

Guillermo Castillo-García received his B.S. (2018) in Computer Science from the University of Murcia, and M.S. (2022) in Research in Artificial Intelligence from the Menéndez Pelayo International University. He is currently a Ph.D. student at the Department of Computer Science of the University of A Coruña. His work is focused on machine learning and feature selection.

Laura Morán-Fernández received her B.S. (2015) and Ph.D. (2020) degrees in Computer Science from the University of A Coruña (Spain). She is currently an Assistant Lecturer in the Department of Computer Science and Information Technologies of the University of A Coruña. She received the Frances Allen Award (2021) from the Spanish Association of Artificial Intelligence (AEPIA). Her research interests include machine learning, feature selection and big data. She has co-authored four book chapters, and more than 15 research papers in international journals and conferences.

Verónica Bolón-Canedo received her B.S. (2009), M.S. (2010) and Ph.D. (2014) degrees in Computer Science from the University of A Coruña (Spain). After a postdoctoral fellowship in the University of Manchester, UK (2015), she is currently an Associate Professor in the Department of Computer Science and Information Technologies of the University of A Coruña. Her main current areas are machine learning and feature selection. She is co-author of more than 100 papers on these topics in international conferences and journals.