Mach Learn (2018) 107:209–246
https://doi.org/10.1007/s10994-017-5681-1

Data complexity meta-features for regression problems

Ana C. Lorena[1] · **Aron I. Maciel**[1] · **Péricles B. C. de Miranda**[2] · **Ivan G. Costa**[3] · **Ricardo B. C. Prudêncio**[2]

Received: 10 May 2016 / Accepted: 4 October 2017 / Published online: 21 December 2017

Abstract In meta-learning, classification problems can be described by a variety of features, including complexity measures. These measures allow capturing the complexity of the frontier that separates the classes. For regression problems, on the other hand, there is a lack of such type of measures. This paper presents and analyses measures devoted to estimate the complexity of the function that should fitted to the data in regression problems. As case studies, they are employed as meta-features in three meta-learning setups: (i) the first one predicts the regression function type of some synthetic datasets; (ii) the second one is designed to tune the parameter values of support vector regressors; and (iii) the third one aims to predict the performance of various regressors for a given dataset. The results show the suitability of the new measures to describe the regression datasets and their utility in the meta-learning tasks considered. In cases (ii) and (iii) the achieved results are also similar or better than those obtained by the use of classical meta-features in meta-learning.

Editors: Pavel Brazdil and Christophe Giraud-Carrier.

✉ Ana C. Lorena
aclorena@unifesp.br

Aron I. Maciel
aimaciel@unifesp.br

Péricles B. C. de Miranda
pbcm@cin.ufpe.br

Ivan G. Costa
ivan.costa@rwth-aachen.de

Ricardo B. C. Prudêncio
rbcp@cin.ufpe.br

1 Instituto de Ciência e Tecnologia, Universidade Federal de São Paulo, Unidade Parque Tecnológico, São José dos Campos, SP 12247-014, Brazil

2 Centro de Informática, Universidade Federal de Pernambuco, Cidade Universitária, Recife, PE 50740-560, Brazil

3 IZKF Research Group Bioinformatics, RWTH Aachen University, Aachen, Germany

Keywords Meta-learning · Meta-features · Complexity measures

1 Introduction

The design of a learning system is a challenging task due to a variety of important issues that have to be addressed like: algorithm selection, optimization of hyperparameters, choice of data processing techniques, among others. Such issues have been dealt with different approaches ranging from the use of optimization meta-heuristics to supervised learning techniques (Thornton et al. 2013; Pappa et al. 2014). Many of these approaches have been referred in the literature as Meta-Learning (MTL) (Brazdil et al. 2008). In this paper, we focus on a MTL setup which relates a set of descriptive meta-features of problems to the performance of one or more algorithms on their solution (or alternatively, hyperparameter configurations, data processing techniques, among others). In this setup, knowledge is extracted from meta-datasets built from experiments performed with different algorithms on previous problems. These algorithms can then be selected or ranked for new problems based on their meta-features. Additionally, MTL approaches have been adopted in the literature as components of more complex hybrid methods in the design of learning systems (Wistuba et al. 2016; Leite et al. 2012; de Miranda et al. 2014; Gomes et al. 2012).

An important aspect in MTL is the definition of the meta-features used to describe the problem at hand. There is a large selection of meta-features adopted in the MTL literature to describe learning problems. They can be roughly classified as: (i) simple measures, statistical and information-theoretic measures, which are extracted from the learning datasets; (ii) landmarking measures, which are descriptors extracted from simple learning models when applied to the dataset; and (iii) model-based features, such as the size of an induced decision tree (Soares 2008). Alternatively, in the case of classification problems, some previous work have demonstrated the value of employing data complexity measures as meta-features (Cavalcanti et al. 2012; Leyva et al. 2015; Garcia et al. 2016; Morán-Fernández et al. 2017). They allow capturing the complexity of the decision boundary that discriminates the classes (Ho and Basu 2002). Examples include measuring the degree of linearity of the problem and the volume of feature overlapping along different classes.

In Cavalcanti et al. (2012), for instance, the authors state that the behavior of nearest neighbor classifiers is highly affected by data complexity. They experimentally show that a combination of complexity measures can be used to predict the expected accuracy of this classification technique. In Leyva et al. (2015), the authors proved the effectiveness of the data complexity measures in recommending algorithms for instance selection. In Garcia et al. (2015), in turn, authors discussed how label noise affects the complexity of classification problems and showed that many of the data complexity measures are able to characterize the presence of noise in a classification dataset. More recently, Morán-Fernández et al. (2017) discussed how the complexity measures values can be employed to predict the expected classification performance of different classifiers commonly employed in microarray data analysis.

Meta-features have been also proposed for the characterization of regression problems. Most of them are similar to the meta-features proposed for classification problems (Kuba et al. 2002). However, until recently no work proposed measures to estimate the complexity of a regression problem. In Maciel et al. (2016), we introduced complexity measures for regression problems. These novel measures cover concepts similar to those from the original work of Ho and Basu (2002), which were transfered for the regression scenario. They quantify aspects as the linearity of the data, the presence of informative features and the smoothness of the target attribute. In Maciel et al. (2016), the proposed measures were also evaluated in a simple case study regarding their ability to distinguish easy from complex problems using simulated data. Several of the measures were able to separate the simpler problems from problems of medium to high complexity.

In this paper we extend our previous work in various aspects. Firstly, we expand the experimental evaluation with synthetic datasets. For this, we use distinct functions to generate datasets of varying complexities, which include polynomials of increasing degrees and trigonometric functions. The range of values of the complexity measures for the different functions are then analyzed and compared. We also use the complexity measures as meta-features for inducing classifiers able to predict the regression function type of the designed datasets. The results show that many of the measures, as well as their combination, are able to distinguish the different types of problems according to their complexity degree. Nonetheless, as noise is introduced into all datasets, this discrimination power is lost.

We also show the practical use of the measures in real datasets by employing them as meta-features in two MTL tasks. The first task is to recommend the parameter values for the Support Vector Regression (SVR) (Basak et al. 2007) technique in new regression problems. The second MTL setup is designed to predict the expected Normalized Mean Squared Error (NMSE) of particular regressors for new regression problems. In both cases, our complexity measures achieved results superior or comparable to those of other meta-features commonly employed in MTL. The achieved results demonstrate the suitability of our complexity measures as generic datasets descriptors in MTL.

Finally, we include a detailed description and efficient implementation of the complexity measures. This includes illustrative examples of the measures operation, algorithms and their asymptotic computational complexities. We have also proposed computationally efficient alternatives for some of the measures. R codes of all complexity measures proposed are made available at https://github.com/aclorena/ComplexityRegression.git.

This paper is structured as follows: Sect. 2 defines MTL and discusses the use of complexity measures as meta-features. Section 3 presents our complexity measures for regression problems. Section 4 evaluates the ability of these measures in distinguishing the complexity of some synthetic regression problems. Section 5 presents the experimental results achieved in the recommendation of SVR parameter values. Section 6 presents the MTL study to predict the NMSE of various regressors. Section 7 concludes this paper.

2 Meta-learning and data complexity

Meta-Learning (MTL) aims to relate features of learning problems to the performance of learning algorithms. MTL receives as input a set of meta-examples built from a set of learning problems and a pool of candidate algorithms. Usually each meta-example stores the features describing a problem (the meta-features) and a target attribute indicating the best candidate algorithm once evaluated for that problem. A meta-model (or meta-learner) is then learned to predict the best algorithm for new problems based on their meta-features. There are different variations of this basic meta-learning task, including for instance to rank the candidate algorithms, to predict the performance measure of a single algorithm (a meta-regression problem) and to recommend values for the hyperparameters of an algorithm based on the problem features (see Brazdil et al. 2008 for a deeper view of meta-learning).

A common issue of previous work in meta-learning is the need to define a (good) set of meta-features. These features have to be informative and discriminative enough to identify different aspects of the learning problems that can bias algorithm performance. Since MTL is problem dependent, researchers have tried to adopt sets of meta-features that are at same time suitable and informative for each application. In many cases, there is no clear justification for the meta-features adopted in each work, which makes it difficult defining which features are actually relevant and useful to be adopted in new applications. Possibly, a reason for this is the lack of systematic studies to specific aspects that can be measured in a learning problem and then the corresponding meta-features that are more informative.

One of these aspects is the difficulty of a learning problem, which can be expressed in terms of complexity measures of a dataset. Ho and Basu (2002) have proposed a set of measures of classification complexity which have been used as meta-features in various recent work (Cavalcanti et al. 2012; Smith et al. 2014; Leyva et al. 2015; Garcia et al. 2016; Morán-Fernández et al. 2017). These measures can be roughly divided into three main categories: (i) measures of overlapping of feature values; (ii) measures of the separability of the classes; (iii) and geometry, topology and density measures.

The feature overlapping group contains measures that quantify whether the dataset contains at least one feature that is able to fully separate the examples within the classes. If one of such features is present, the problem can be considered simple according to this perspective. In Orriols-Puig et al. (2010) the collective effect of more features in separating the data is also taken into account.

The measures from group (ii) quantify how the classes are separated. One approach for such is to verify if the classes are linearity separable. Linearly separable problems are considered simpler than problems where non-linear boundaries are required. Other measures try to estimate the length of the decision boundary and the overlapping of the classes.

Group (iii) contains measures that capture the inner structure of the classes. The idea is that if examples of the same class are densely distributed, the problem can be considered simpler than when the examples from a same class are sparsely distributed or occupy many manifolds.

Most MTL approaches are in principle independent on the base-learning task of interest (e.g., classification or regression). However, most of the experimental studies were based on the classification problem. As a consequence, there is a lack of meta-features proposed specifically for describing regression problems.

Few previous articles, which employed meta-learning in regression problems, adopted meta-features that resemble complexity measures of the target attribute. However, none of these work explicitly addressed the regression complexity. Examples of such meta-features are classical statistics for continuous attributes (e.g., coefficient of variation, skewness, kurtosis, etc.), which measure symmetry and dispersion of the target attribute in a dataset (Kuba et al. 2002; Soares et al. 2004; Amasyali and Erson 2009; Gomes et al. 2012; Loterman and Mues 2012). In Soares and Brazdil (2006), the authors proposed the use of Kernel Target Alignment (KTA) (Cristianini et al. 2002) as a specific meta-feature to recommend hyper-parameters for support vector regressors. KTA measures the degree of agreement between a kernel and the target attribute in a problem. This meta-feature was also adopted in Gomes et al. (2012). Meta-features based on the landmarking approach were adopted in Kuba et al. (2002), Soares et al. (2004) and Amasyali and Erson (2009), in which simple regression algorithms such as linear regression and decision stumps were adopted as landmarkers. As described next, our work contributes with the delineation and formalization of complexity measures as meta-features for regression problems. The presented set of complexity measures considers different perspectives of data complexity that were not dealt with in any previous work concerning regression problems.

3 Complexity of regression problems

The complexity of a regression problem can be attributed to various factors, some of them similar to those of classification problems. For instance, a regression problem can be complex if it is ill defined and/or described by features that are not informative enough. Data sparsity can also make a problem that is originally complex deceptively simple or vice-versa.

The distribution of the target attribute can indicate whether the regression problem is simple or not too. If the outputs of neighbor points are distributed smoothly, usually this means that simpler functions can be fitted to them than if they are spread in the space.

Finally, the complexity of the objective function which relates the inputs to the target attribute is intrinsic to the problem. Our complexity measures try to quantify the former factors.

The previous interpretations are focused on a concept of intrinsic or absolute complexity, but the complexity of a problem can also be regarded as relative. For instance, whilst approximating non-linear functions can be considered simple for properly tuned multilayer Neural Networks, the same does not hold for a naïve Adaline linear regressor. By relating the values of the proposed complexity measures to the performance achieved by some specific learning technique, as done in MTL, we are also able to assess the relative complexity of a problem.

Our measures are intended to estimate the complexity of regression problems from different perspectives. They are divided into the following categories:

- *Feature correlation measures*: capture the relationship of the feature values with the outputs;
- *Linearity measures*: estimate whether a linear function can fit data, that is, if they are linearly distributed;
- *Smoothness measures*: estimate the smoothness of the function that must be fitted to the data;
- *Geometry, topology and density measures*: capture the spatial distribution and the structure of the data.

To define the complexity measures, let X denote the data matrix, which is composed of n data points described by d predictive features. Each line from X corresponds to a data item $\mathbf{x}_i$, where $i = 1, \ldots, n$, while a column j denotes a feature vector $\mathbf{x}^j$, where $j = 1, \ldots, d$. Each data item has also a label $y_i \in \Re$ and the composition of all labels forms a label vector $\mathbf{y}$. In the definition of some of the measures, we also use the statistical model of a Multiple Linear Regression, which can be expressed as:

$$f(\mathbf{x}) = \beta_0 + \beta_1 x^1 + \cdots + \beta_d x^d + \varepsilon, \tag{1}$$

where the β_i values are the coefficients of the linear function and the ε is a residual or error.

To exemplify the computation of the measures in this section, we will use the two illustrative datasets shown in Fig. 1: Dataset1 (Fig. 1a) and Dataset2 (Fig. 1b). Both datasets contain $n = 1000$ examples and $d = 2$ features with values in the [0, 1] range. The Spearman correlation ρ of each one of the predictive features (x_1, x_2 and x_3) generated to the output y is shown in parenthesis. Dataset1 contains one feature linearly related to the output y (x_1—Fig. 1c),

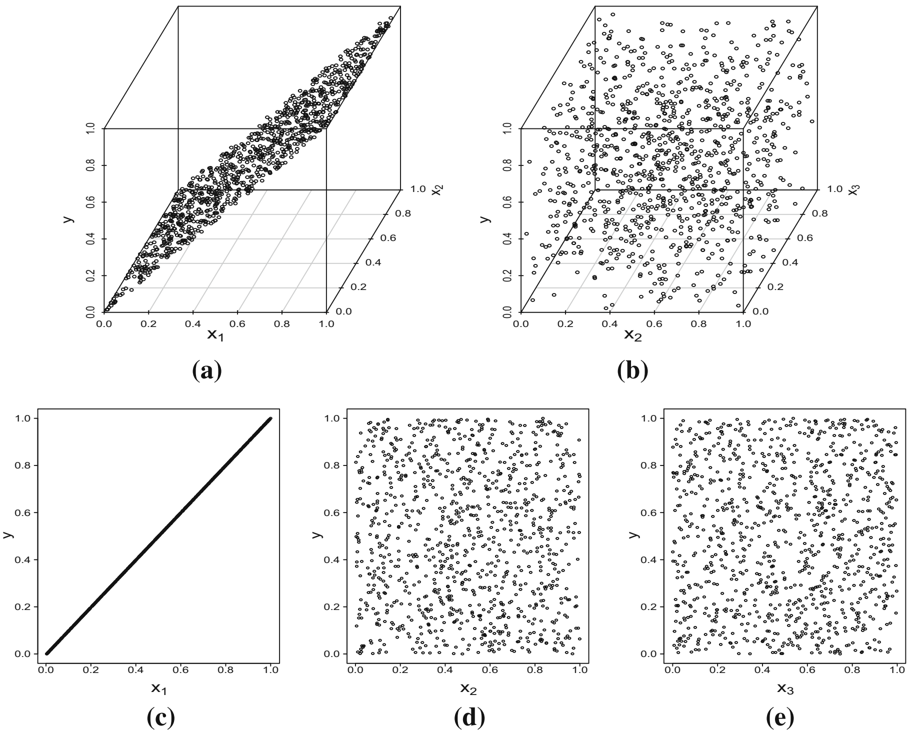


Fig. 1 Examples of regression datasets: while in Dataset2 both features (x_2, x_3) show a random distribution to the output y, in Dataset1 one of the features (x_1) is linearly related to the output y. **a** Dataset1. **b** Dataset2. **c** Feature x_1 ($\rho = 1$). **d** Feature x_2 ($\rho = -0.03$). **e** Feature x_3 ($\rho = 0.02$)

while its second feature (x_2—Fig. 1d) shows a random distribution. Dataset2 is composed by two features with random distribution: x_2 (Fig. 1d) and x_3 (Fig. 1e). This makes Dataset1 simpler than Dataset2 concerning the regression function to be fitted. Although these examples are trivial and quite naïve, they allow illustrating the main concepts captured by each of the measures introduced next.

3.1 Feature correlation measures

This category contains measures that consider the correlation of the values of the features to the outputs. If at least one feature is highly correlated to the output, this indicates that simpler functions can be fitted to the data. Most of the measures from this category are univariate and analyze each feature individually, disregarding their potential correlation or joint effect. This is justified by the fact that we are concerned with simple measures that give an indicative of the problem complexity. Often a combination of more measures will be required for a proper problem characterization.

3.1.1 Maximum feature correlation to the output (C_1)

In C_1 first one has to calculate the absolute value of the Spearman correlation between each feature and the outputs. The absolute value is taken because both extremes of the correlation measure, which range between [−1, 1], are strong (either inversely or directly). C_1 is the maximum correlation value obtained among all the features, as represented in Eq. 2.

$$C_1 = \max_{j=1,\ldots,d} |\rho(\mathbf{x}^j, \mathbf{y})|, \tag{2}$$

where ρ is the Spearman correlation. The Spearman correlation was chosen because it is non-parametric and does not assume any particular distribution between the outputs $\mathbf{y}$ and the predictive feature vectors from the dataset. For each feature, this measure requires the computation of the Spearman correlation coefficient ($\mathrm{O}(n \cdot \log n)$), which results in a total asymptotic complexity of $\mathrm{O}(d \cdot n \cdot \log n)$ for C_1 computation.

Higher values of C_1 indicate simpler problems, where there is at least one feature strongly correlated with the output. For Dataset1, the C_1 value is 1, since feature x_1 has maximum correlation to the target attribute. For Dataset2, the C_1 value is 0.03, indicating no relationship of any of the predictive features to the target attribute.

3.1.2 *Average feature correlation to the output (C_2)*

C_2 is similar to C_1, but computes the average of the correlations of all features to the output, as opposed to just taking the maximum value among them. Therefore, it is a rough measure of the relationship of all the features to the output. C_2 is defined as:

$$C_2 = \sum_{j=1}^{d} \frac{|\rho(\mathbf{x}^j, \mathbf{y})|}{d}. \tag{3}$$

As with C_1, this measure has a total asymptotic computational complexity of $\mathrm{O}(d \cdot n \cdot \log n)$. Higher values of C_2 indicate simpler problems. For Dataset1, the C_2 value is 0.52, whilst it is 0.03 for Dataset2. This measure is impacted by the amount of features with poor association with the output variable.

3.1.3 *Individual feature efficiency (C_3)*

C_3 calculates, for each feature, the number of examples that must be removed from the dataset until a high correlation value to the output is achieved. Here we have adopted $|\rho(\mathbf{x}^j, \mathbf{y})| > 0.9$ as a threshold for a high correlation. It then calculates the number of examples that are removed, divided by the total number of examples. The intuition is to capture the explanatory power of each one of the features by verifying how many of the original examples they can relate to the output with a high confidence. This computation is done for each individual feature and the minimum of the values found is returned. This corresponds to the feature most related to the output. Therefore, if one predictive feature of a dataset is highly related to the output, this measure interprets it as a simple regression problem. C_3 can be expressed as:

$$C_3 = \min_{j=1}^{d} \frac{n^j}{n}, \tag{4}$$

where n^j is the number of examples removed for obtaining a high correlation value between the j-th feature vector and the target attribute.

A naïve implementation of C_3 would have a cubic complexity on the number of samples. We propose therefore an efficient implementation of C_3 which takes advantage of the fact the Spearman correlation works on differences of rankings. Instead of re-estimating the rankings

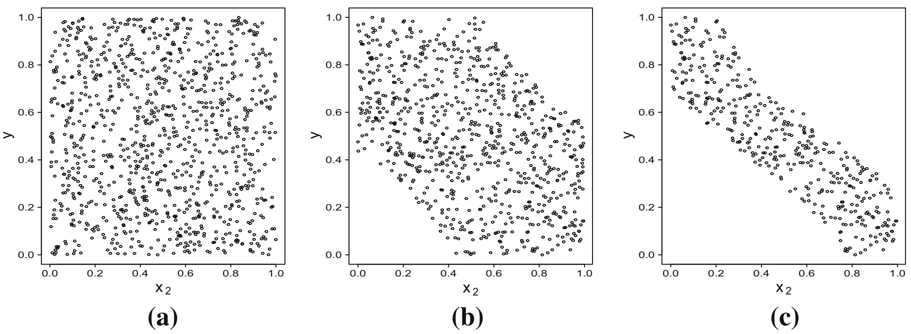


Fig. 2 Example of C_3 operation for feature x_2 (Fig. 1d). **a** Initial dataset. **b** Intermediate step. **c** Final dataset

after the removal of each observation, we simply update the difference of the rankings. The proposed algorithm has the worst time complexity of $\mathrm{O}(d \cdot n^2)$ and is shown in the Appendix.

Lower values of C_3 indicate simpler problems. If there is at least one feature with correlation higher than 0.9 to the output, all examples are preserved in the dataset and the C_3 value is the minimum (null). This is the case of Dataset1, where feature x_1 has a maximum correlation of 1 to the output. In the case of Dataset2, on the other hand, 601 of the 1000 initial examples have to be removed so that feature x_2 shows a correlation higher than 0.9 to the target attribute and 604 examples must be removed in the case of feature x_3. Therefore, the C_3 value for Dataset2 is 0.601, indicating a higher complexity (as more examples need to be removed to achieve a high correlation, the more complex is the problem).

The execution of C_3 in feature x_2 (Fig. 1d) is illustrated in Fig. 2. Initially this feature is distributed as shown in Fig. 2a, with a correlation of -0.03 to the output variable y. For obtaining an absolute correlation higher than 0.45, 207 examples are removed and the remaining examples are plotted in Fig. 2b. At the end of the processing, 601 examples are removed for obtaining an absolute correlation superior to 0.9 and the remaining examples are distributed as shown in Fig. 2c. As illustrated, C_3 operation for a particular feature can also be regarded as measuring the proportion of examples that need to be removed from the dataset in order to simplify the relation of this predictive variable to the target attribute.

3.1.4 Collective feature efficiency (C_4)

C_4 starts by identifying the feature with highest correlation to the output. All examples with a small residual value ($|\varepsilon_i| \leq 0.1$) after a linear fit between this feature and the target attribute are removed. Then, the most correlated feature to the remaining data points is found and the previous process is repeated until all features have been analyzed or no example remains. Different from C_3, examples removed at a round are disregarded in next rounds of C_4. C_4 returns the ratio of examples for which the residuals absolute values of the linear fit are larger than the threshold 0.1 and is expressed by Eq. 5.

$$C_4 = \frac{\sharp\{\mathbf{x}_i || \epsilon_i| > 0.1\}_{T_l}}{n}, \tag{5}$$

where the numerator gives the number of examples that remain in the dataset at the end of C_4 operation. T_l is the dataset from which this number is calculated and l denotes the number of iterations performed by the algorithm. The l value can range between 1 and d. The algorithm

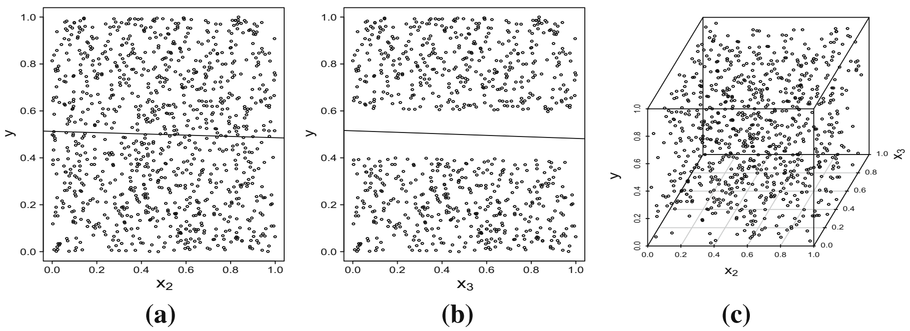


Fig. 3 Computing C_4 for Dataset2. **a** Linear fit with x_2. **b** Linear fit with x_3 on remaining points. **c** Final dataset

for computing this measure is described in the Appendix. It has a total asymptotic complexity of $O(d^2 + d \cdot n \cdot \log n)$.

Higher values of C_4 indicate more complex problems. The C_4 value for Dataset1 is 0, since using feature x_1 a linear fit with null error is obtained and therefore all examples are removed from the dataset in one unique round of the algorithm. For Dataset2, the C_4 value is 0.816, since 816 of the 1000 initial examples remain at the end of the rounds. Figure 3 shows the execution of C_4 algorithm on Dataset2. The initial feature analyzed is x_2, which shows a higher correlation to y. A linear fit is found between this feature and the target attribute (Fig. 3a). The examples with small residual values are then removed (they correspond to 181 examples), as shown in Fig. 3b. The next feature analyzed is x_3. The linear fit between x_3 and y considering the remaining 819 examples is also shown in Fig. 3b. After this fit, 3 examples have a small residual value and are removed, leaving 816 examples in the dataset. The final dataset is illustrated at Fig. 3c. Therefore, the algorithm removes from the dataset all examples regarded as simple by each predictive feature (those that are linearly related to the output variable when taking each feature into account).

3.2 Linearity measures

These measures capture whether a linear function provides a good fit to the problem. If this is the case, the problem can be considered simpler than one in which a non-linear function is required. Both of the measures from this section employ the residuals of a multiple linear regressor. Concerning computational complexity, both measures here show an asymptotic complexity of $O(n \cdot d^2)$, which comes from the linear regression estimation.

3.2.1 Mean absolute error (L_1)

L_1 averages the absolute values of the residues of a multiple linear regressor. Lower values indicate simpler problems, which can be fitted by a linear function. L_1 can be expressed as:

$$L_1 = \sum_{i=1}^{n} \frac{|\varepsilon_i|}{n} \tag{6}$$

The L_1 value for Dataset1 is null, since a linear hyperplane fits this data. For Dataset2, the L_1 value is 0.26.

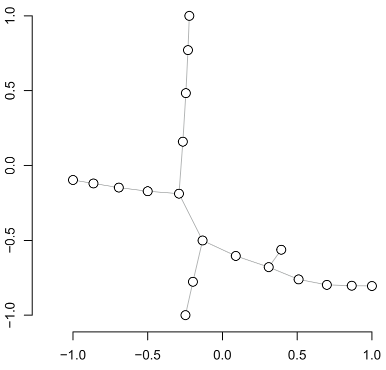


Fig. 4 MST generated from an hypothetical bi-dimensional input dataset

3.2.2 Residuals variance (L_2)

L_2 averages the square of the residuals from a multiple linear regression. Smaller values indicate simpler (linear) problems. For instance, L_2 is null for Dataset1 and it equals 0.09 for Dataset2.

$$L_2 = \sum_{i=1}^{n} \frac{\varepsilon_i^2}{n} \tag{7}$$

3.3 Smoothness measures

In regression problems, the smoother the function to be fitted to the data, the simpler it shall be. Larger variations in the inputs and/or outputs, on the other hand, usually indicate the existence of more intricate relationships between them. The measures from this category measure this aspect and also allow capturing the shape of the curve to be fitted to the data.

3.3.1 Output distribution (S_1)

Like the *Fraction of Points in the Boundary* (N_1) measure from Ho and Basu (2002), first a *Minimum Spanning Tree* (MST) is generated from input data, as shown in Fig. 4 for a hypothetical bi-dimensional dataset. Each data item corresponds to a vertex of the graph, while the edges are weighted according to the *Euclidean distance* between the examples in the input space. The MST will greedily connect examples nearest to each other. Next S_1 monitors whether the examples joined in the MST have similar output values. Lower values indicate simpler problems, where the outputs of similar examples in the input space are also next to each other. S_1 can be expressed as:

$$S_1 = \frac{1}{n} \sum_{i:j \in MST} |y_i - y_j|, \tag{8}$$

where the sum is taken for all vertices i and j that are adjacent in the MST. The S_1 measure will average the outputs of points connected in the MST. If the outputs of similar entries are next to each other, the measure will return lower values. This will be indicative of simpler problems. For Dataset1, the computed S_1 measure is 0.01, while it is equal to 0.35 for Dataset2. We do not show the MST for these datasets here, as their large number of examples impairs the visualization of the tree.

For building the graph from the data it is necessary to compute the distance matrix between all pairs of elements, which requires $O(n^2 \cdot d)$ operations. Afterwards, using the *Prim's algorithm* for estimating the MST requires $O(n^2)$ operations. Therefore, the asymptotic complexity of S_1 is $O(n^2 \cdot d)$.

3.3.2 Input distribution (S_2)

S_2 first orders the data points according to their output values y_i and then computes the *Euclidean distance* between pairs of examples that are neighbors. S_2 complements S_1 by measuring how similar in the input space are data items with similar outputs. Lower values indicate simpler problems, where similar points in the input space share similar outputs. S_2 can be expressed by Eq. 9, given that $y_1 \leq y_2 \leq \cdots \leq y_n$, that is, that the examples are already ordered according to their output values.

$$S_2 = \frac{1}{n}\sum_{i=2}^{n} ||\mathbf{x}_i - \mathbf{x}_{i-1}||_2, \tag{9}$$

Figure 5a presents an example of how S_2 proceeds for an hypothetical one-dimensional dataset, whose examples are already ordered according to y. Two neighboring examples (with similar y values) are highlighted: the doubled circle and the black circle. Their distance in the input space is represented by a dashed line. This value is averaged in S_2 for all pairs of neighbor examples. For Dataset1, the S_2 value is 0.32, while for Dataset2 it is 0.51. The algorithm for S_2 computation is presented in the Appendix. It has an asymptotic complexity of $\mathrm{O}(n \cdot (d + \log n))$.

3.3.3 Error of a nearest neighbor regressor (S_3)

S_3 calculates the mean squared error of a *1-nearest neighbor regressor* (NN), using *leave-one-out*. It is similar to the *error rate of a Nearest Neighbor Classifier* (N_3) measure from Ho and Basu (2002). The 1-NN regressor looks for the training example $\mathbf{x}_i$ most similar to the new example and assigns to it the same output y_i. S_3 measures how the examples are close together and high values imply that there are many gaps in the input space. S_3 can be expressed as:

$$S_3 = \frac{1}{n}\sum_{i=1}^{n} (NN(\mathbf{x}_i) - y_i)^2, \tag{10}$$

where $NN(\mathbf{x}_i)$ represents the 1-nearest neighbor prediction for $\mathbf{x}_i$. In the one-dimensional dataset from Fig. 5b, for instance, the nearest neighbor of the black circle in the input space is marked with a double circle. The black circle would then receive the same output as the double circle, which results in the error represented by a dashed line. Lower S_3 values will be achieved for simpler problems. For instance, the S_3 value for Dataset1 is null, while it is 0.18 for Dataset2. The algorithm for S_3 computation is presented in the Appendix. It has an asymptotic computational cost of $\mathrm{O}(d \cdot n^2)$.

3.4 Geometry, topology and density measures

These measures capture the distribution and density of the examples in the input/output space.

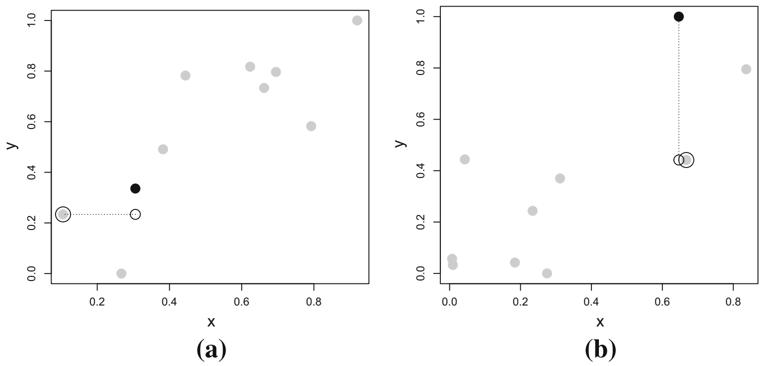


Fig. 5 **a** Distance between two examples with similar outputs in a hypothetical one-dimensional dataset. **b** *Nearest neighbor* output distance for a new example

3.4.1 Non-linearity of a linear regressor (L_3)

L_3 is based on the *Non-linearity of a Linear Classifier* (L_3) measure from Ho and Basu (2002), with adaptations to the regression scenario. For a given dataset, it first selects pairs of examples with similar outputs and creates a new test point by randomly interpolating them. Here both input and output features are interpolated, while in Ho and Basu (2002) random examples of the same class have their input features interpolated.

A linear regressor is trained using the original data and has its *Mean Squared Error* (MSE) measured in the new points. L$_3$ measures how sensitive the regressor is to the new points. If the original training points are distributed smoothly, their interpolated variants will be next to the original data items and the MSE will be consequently low. More formally,

$$L_3 = \frac{1}{l}\sum_{i=1}^{l}\left(f(\mathbf{x}'_i) - y'_i\right)^2, \tag{11}$$

where l is the number of interpolated examples $\mathbf{x}'_i$ generated and y'_i are their labels. Lower values of L_3 indicate simpler problems. In our implementation, random data points are generated between all pairs of the original examples which have neighbor outputs, so that $l = n - 1$. The L_3 algorithm is presented in the Appendix. The total cost of the algorithm is $\mathrm{O}(n(d^2 + \log n))$.

L_3 operation is shown in Fig. 6 for a hypothetical one-dimensional dataset. A linear function is fitted to the original dataset, as shown in Fig. 6a. Random test points between examples with similar outputs are generated, as in Fig. 6b. The error of the linear regressor found is then estimated for the new data points, as shown in Fig. 6c.

The L_3 value for Dataset1 is 0, while it is 0.09 for Dataset2. Figure 7a and b shows the new datasets formed by interpolating pairs of examples with neighbor outputs for Datasets 1 and 2, respectively, from which the MSE values are calculated. As expected, the interpolated version of Dataset1 preserves its linear relationship to the output variable y.

3.4.2 Non-linearity of nearest neighbor regressor (S_4)

S_4 employs the same procedure as before, but using a *nearest neighbor regressor* instead in the output predictions. The final result would then be:

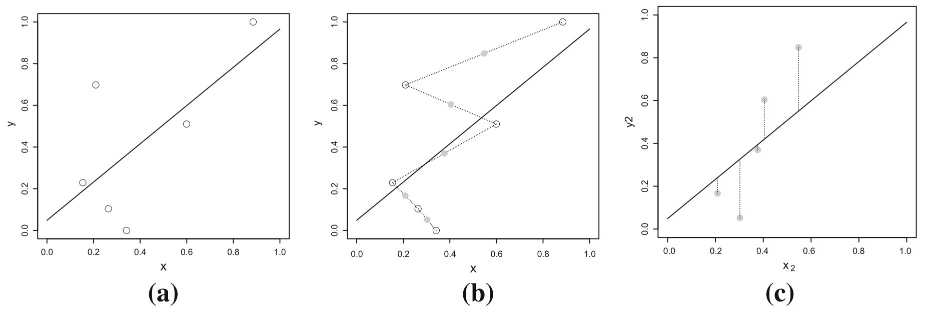


Fig. 6 Example of L_3 operation on a hypothetical one-dimensional dataset. **a** Linear fit in the original dataset. **b** Generation of new data points. **c** Error of linear regressor on new data points

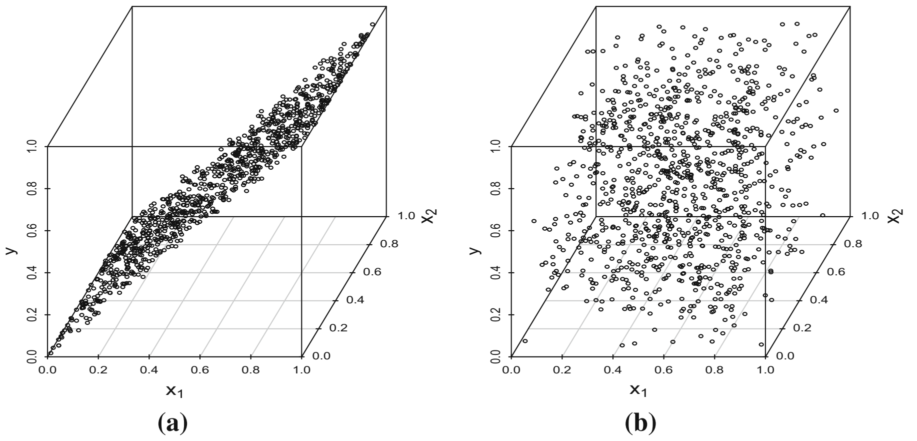


Fig. 7 Datasets obtained by random interpolation of examples with similar outputs from Dataset1 and Dataset2. **a** New points from Dataset1. **b** New points from Dataset2

$$S_4 = \frac{1}{l}\sum_{i=1}^{l}\left(NN(\mathbf{x}_i') - y_i'\right)^2 \qquad (12)$$

The computation of the measure has an asymptotic cost of $O(n \cdot d \cdot \log n)$, using a KD-tree implementation of the NN algorithm. The S_4 value for Dataset1 is null and it is equal to 0.17 in the case of Dataset2.

3.4.3 Average number of examples per dimension (T_2)

T_2 is defined as the average number of examples per dimension. T_2 (Eq. 13) gives an indicative on data sparsity and is the same as defined in Ho and Basu (2002). Lower values indicate sparser and more complex datasets.

$$T_2 = \frac{n}{d} \qquad (13)$$

This simple statistic has complexity linear to the size of variables and samples [$O(n + d)$]. For both Dataset1 and Dataset2, the value of T_2 is 250, since both datasets contain the same number of examples and predictive features.

Table 1 Summary of the measures

Category	Acronym	Min.	Max.	Asymptotic cost	Complexity
Feature correlation	C_1	0	1	$O(d \cdot n \cdot \log n)$	$\downarrow$
	C_2	0	1	$O(d \cdot n \cdot \log n)$	$\downarrow$
	C_3	0	1	$O(d \cdot n^2)$	$\uparrow$
	C_4	0	1	$O(d \cdot (d + n \cdot \log n))$	$\uparrow$
Linearity	L_1	0	-	$O(n \cdot d^2)$	$\uparrow$
	L_2	0	-	$O(n \cdot d^2)$	$\uparrow$
Smoothness	S_1	0	-	$O(d \cdot n^2)$	$\uparrow$
	S_2	0	-	$O(n \cdot (d + \log n))$	$\uparrow$
	S_3	0	-	$O(d \cdot n^2)$	$\uparrow$
Geometry topology and density	L_3	0	-	$O(n \cdot (d^2 + \log n))$	$\uparrow$
	S_4	0	-	$O(n \cdot d \cdot \log n)$	$\uparrow$
	T_2	≈ 0	-	$O(n + d)$	$\downarrow$

3.5 Summary of the measures

Table 1 summarizes the presented measures, presenting their category, acronyms, minimum (Min.) and maximum (Max.) values (when such values are unbounded, a "-" symbol is used), worst-case asymptotic computational cost and relation to the complexity (Complexity) of the regression problem. In the latter case, while the symbol $\uparrow$ denotes that the higher the value of the measure, the higher the complexity of the problem (a direct relation), the $\downarrow$ symbol denotes the opposite (an indirect) relation.

As can be observed in Table 1, the majority of the measures are unbounded. Data should be normalized between [0, 1] before the application of these measures (except from T_2) so that upper values close to 1 can be obtained. Normalization is required for all measures that compute some distance between examples, such as S_1 to S_4. Regarding the worst-case asymptotic computational cost, all measures can be computed in polynomial times on the number of features and examples.

4 Experiments with synthetic datasets

For evaluating the ability of the previous measures in quantifying the complexity of regression problems, we first performed experiments on a set of synthetic datasets. The idea is to verify how the measures behave with regression functions of increasing complexity.

4.1 Datasets

Each synthetic dataset has $n = 500$ entries $\mathbf{x}_i \in \Re^d$, whose d feature values are randomly chosen in [0, 1] according to an uniform distribution. The number of features was varied as $d \in \{1, 2, 5, 10\}$. The $y_i \in \Re$ output values are calculated so that the underlying function relating data is varied between:

– *Polynomial with varying degrees* as given by Eq. 14. The β_{ij} values are randomly chosen in the [0, 1] interval, but remain the same for all n data points that are generated for a

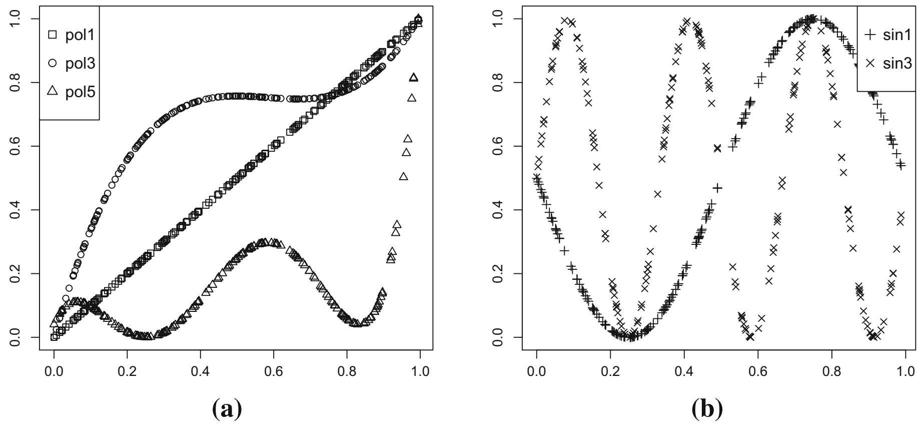


Fig. 8 Examples of synthetic datasets generated ($\sigma = 0$). **a** Polynomials. **b** Sine waves

given dataset. The polynomial degree (p) values were varied as 1, 3 and 5. When $p = 1$, we have a perfect linear relationship between the input variables and the output values. The higher the degree, the higher the complexity of the regression problem, as illustrated in Fig. 8.

$$\begin{aligned} f(\mathbf{x}) =& \beta_0 + \beta_{11}(x^1) + \cdots + \beta_{1p}(x^1)^p + \\ & \beta_{21}(x^2) + \cdots + \beta_{2p}(x^2)^p + \\ & \vdots \\ & \beta_{d1}(x^d) + \cdots + \beta_{dp}(x^d)^p \end{aligned} \tag{14}$$

– *Sine waves with varying frequencies* as expressed by Eq. 15. In this equation, a is the amplitude of the senoid, which we set as 1, b is the frequency and ψ is the phase of the curve, which is set randomly in the $[0, \pi]$ interval. The b values are varied as 1 and 3. The sine wave has more cycles for higher b values, making the regression function more complex. Examples of datasets generated are shown in Fig. 8.

$$\begin{aligned} f(\mathbf{x}) =& a\sin(2\pi bx^1 + \psi_1) + \\ & a\sin(2\pi bx^2 + \psi_2) + \\ & \vdots \\ & a\sin(2\pi bx^d + \psi_d) \end{aligned} \tag{15}$$

We also generated noisy versions of all synthetic datasets by introducing residuals or errors to the outputs. Thereby, a residual ε_i is added to all examples. They are assumed to follow the $N(0, \sigma)$ distribution. The higher the σ, the higher the residuals and the higher is the underlying complexity of the dataset. Three σ values were tested: 0 (no noise), 0.5 and 1.

For each d value, function type and σ variation, 50 datasets were produced. We have, therefore, a total of 3000 synthetic datasets. For all datasets, the outputs are normalized in the [0, 1] interval after generation.

4.2 Individual values of the measures

Figure 9 presents boxplots of the values of the proposed measures across each type of regression problem generated. Results for the T_2 measure are not shown, since there is no variation despite of the regression function used. For each measure, there are five groups of three boxplots. The groups correspond to the regression functions, namely: Polynomial of degrees 1, 3 and 5 (pol1, pol3 and pol5) and Sine with frequencies of 1 and 3 (sin1 and sin3). Within each function type, there are three different boxplots, which correspond to different σ values in the residuals variation: 0, 0.5 and 1. The boxplots for datasets with no errors are colored in red, those for $\sigma = 0.5$ are colored in green and those for $\sigma = 1.0$ are colored in blue. The values of the measures were normalized in the [0, 1] range so that all the plots could be in a same scale.

According to the results shown in Fig. 9, when no noise is introduced to the datasets (boxplots shown in red), almost all measures show variations in their values according to the complexity of the functions. In general, the correlation measures were more effective in separating the types of functions. Taking C_3, for instance, we can see that its values increase for more complex functions, which is in accordance to the expected behavior of this measure for more complex problems (easier problems will show lower C_3 values).

In some cases the values of the measures for the polynomials of higher degrees are similar or indicate a superior complexity of these functions when compared to the simpler senoid (ex. C_1, C_2, S_1, S_3). Since they are functions of different categories, they may be comparably simpler or more complex depending on the aspect to be measured.

For many cases the median values of the measures are similar for the polynomials of degrees 3 and 5 (ex. C_4, L_1, L_2, L_3, S_4). There are also cases where the values for these functions overlap to those of sin1 senoid. Nonetheless, in all cases the simpler problems (linear) can be easily distinguished from the more complex function (senoid with frequency of 3). This is less evident for the S_2 measure, for which the boxplots overlap more.

Performing a Friedman statistical test for assessing the individual ability of each measure in differentiating pairs of function types, there are specific cases where this distinction cannot be made. Table 2 shows the results of the Nemenyi pos-test at 95% of confidence level. When the values of a measure do not allow to differentiate a given pair of functions, a dot is placed at the corresponding row and column from Table 2. Red dots denote the results for $\sigma = 0.0$, whilst green dots correspond to the $\sigma = 0.5$ noise level and the blue dots denote the $\sigma = 1.0$ noise level. It is possible to notice, for instance, that for all σ values the values of the measures C_4, L_1, L_2, L_3 and S_4 do not allow to differentiate the pol3 and pol5 function types. This is in accordance to our previous observation that the median values of the measures are quite similar for the polynomials of degrees 3 and 5.

It is also possible to notice in Table 2 that while many of the measures are able to separate the pairs of functions in the noiseless scenario ($\sigma = 0$), when noise is added to the outputs, all measures are negatively impaired. Noise effect was more harmful in the case of measures C_4, L_1, L_2, L_3, S_1, S_3 and S_4. In these cases, the values of the measures overlap for all types of functions. For instance, S_1 builds a Minimum Spanning Tree from input data, whose structure can be very different face of small data changes, as those introduced by noise. Measures relying on the residuals of a regression function, such as C_4, L_1, L_2 and L_3 are naturally affected by the noise variation procedure adopted in the generation of the datasets. On the other hand, the correlation measures C_1, C_2 and C_3 can be regarded as quite robust in differentiating most of the pairs of functions, despite of the noise level.

It is also possible to notice in Fig. 9 that some measures show a high variance, as indicated by their longer boxplots. This happens specially for some specific combinations of measures and function types, as for the correlation measures and function pol1 and for the linearity

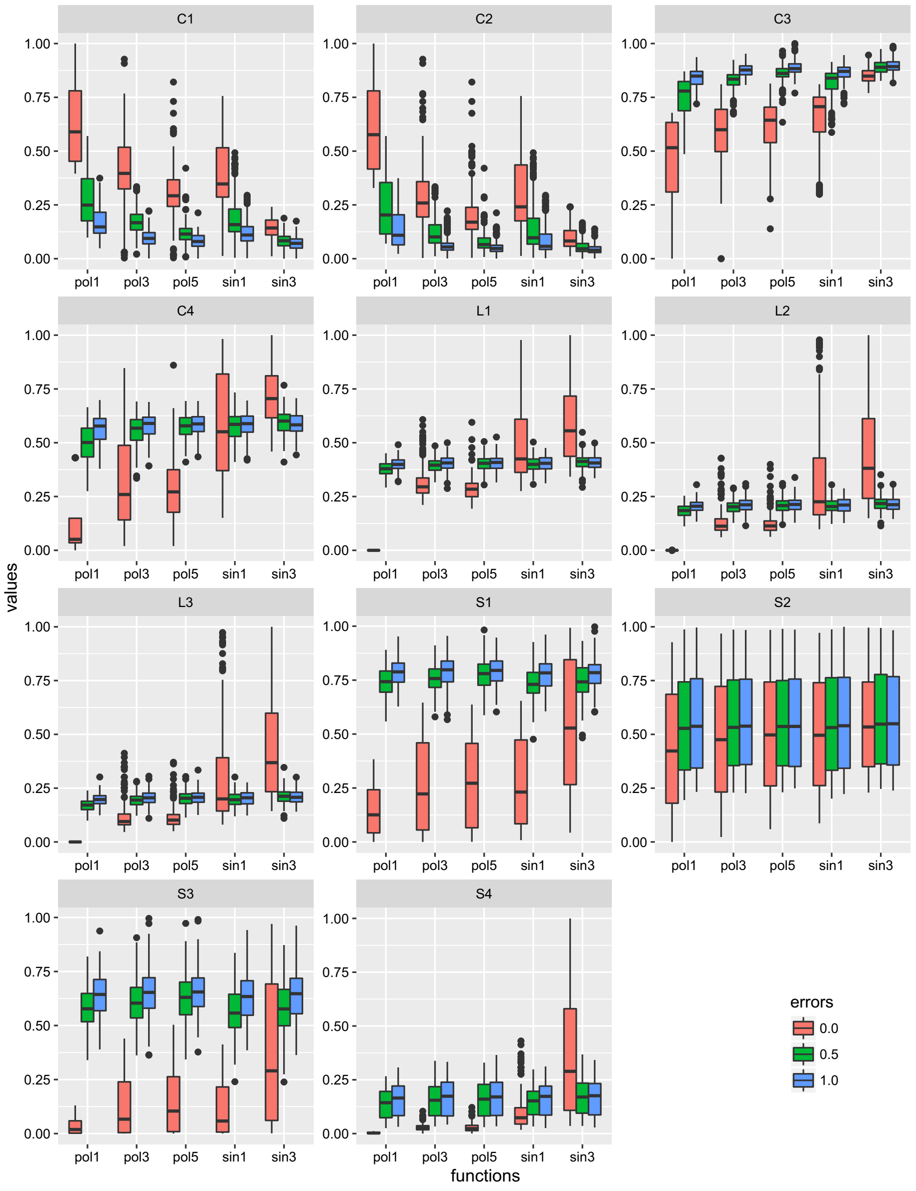


Fig. 9 Boxplots of measures values for the synthetic datasets generated

and smoothness measures and function sin3. This variability is due to the impact of the data dimension in the results of these measures, since each boxplot includes the results for datasets with distinct dimensions. Appendix Fig. 14a shows the boxplots of the C_1 values for the pol1 function, separated by the dimensions of the datasets (1, 2, 5 and 10). The same type of boxplot is presented in Appendix Fig. 14b for the sin3 function and the S_3 values. Data dimension affects specially the correlation-based measures. For instance, for a one-dimensional dataset with no output noise the only feature will show maximum correlation

Table 2 Statistical test results: pairwise comparison of functions according to the values of each complexity measure (Color table online)

	σ	C_1	C_2	C_3	C_4	L_1	L_2	L_3	S_1	S_2	S_3	S_4
pol1–pol3	0.0											
	0.5								🟢		🟢	
	1.0				🔵	🔵	🔵		🔵		🔵	
pol1–pol5	0.0											
	0.5											
	1.0				🔵	🔵	🔵	🔵	🔵	🔵	🔵	🔵
pol1–sin1	0.0											
	0.5								🟢		🟢	🟢
	1.0				🔵	🔵	🔵	🔵	🔵	🔵	🔵	🔵
pol1–sin3	0.0											
	0.5								🟢		🟢	
	1.0				🔵	🔵	🔵		🔵		🔵	
pol3–pol5	0.0				🔴	🔴	🔴	🔴				🔴
	0.5				🟢	🟢	🟢	🟢	🟢	🔵	🟢	🟢
	1.0	🔵		🔵	🔵	🔵	🔵	🔵	🔵	🔵	🔵	🔵
pol3–sin1	0.0		🔴								🔴	
	0.5	🟢	🟢	🟢		🟢	🟢	🟢	🟢	🟢		🟢
	1.0		🔵	🔵	🔵	🔵	🔵	🔵	🔵	🔵	🔵	🔵
pol3–sin3	0.0											
	0.5								🟢		🟢	
	1.0				🔵	🔵	🔵		🔵		🔵	
pol5–sin1	0.0								🔴	🔴		
	0.5				🟢	🟢	🟢	🟢				
	1.0				🔵	🔵	🔵	🔵	🔵	🔵	🔵	🔵
pol5–sin3	0.0											
	0.5					🟢	🟢	🟢				🟢
	1.0	🔵	🔵		🔵	🔵	🔵	🔵	🔵		🔵	🔵
sin1–sin3	0.0											
	0.5								🟢		🟢	
	1.0				🔵	🔵	🔵	🔵	🔵		🔵	🔵

Red dots indicate that there was a significant difference for $\sigma = 0.0$, green dots represent differences for $\sigma = 0.5$ and blue dots indicate significant differences for $\sigma = 1.0$

to the output. When more dimensions are added, the influence of each feature is reduced (Appendix Fig. 14).

Another noticeable result is the fact that all those measures relying on a linear regressor (C_4, L_1, L_2 and L_3) indicate that sin1 and sin3 are more complex without noise than with noise. This also happens for S_4 and function sin3. Whilst it is clear that these regression problems are non linear, this is a case where these individual measures fail to assess the complexity introduced by noise.

Despite the previous observation, overall the results obtained for the synthetic datasets evidences the ability of the proposed measures in revealing the complexity of the regression

Table 3 Error rates of classifiers: regression function prediction

Classifier	$\sigma = 0$		$\sigma = 0.5$		$\sigma = 1$	
	Aver.	Std.	Aver.	Std.	Aver.	Std.
RF	0.06	0.03	0.38	0.04	0.57	0.04
MLP	**0.03**	0.02	**0.31**	0.04	**0.55**	0.04
SVM	0.13	0.03	0.40	0.05	0.62	0.03
3NN	0.06	0.02	0.45	0.04	0.65	0.02

Best results achieved per noise level are highlighted in bold

problems generated. But we expect that a combination of the measures will better characterize the complexity of a problem, since they look at different perspectives regarding data complexity. This aspect is analyzed next.

4.3 Combination of the measures: prediction of function type

For evaluating the combined effect of the proposed measures, we performed a simple MTL experiment for distinguishing the synthetic datasets according to their complexity level. The objective is to predict the regression function that should be fitted to the data. For each noise level ($\sigma = 0$, 0.5 and 1), a meta-dataset is built. Each synthetic dataset corresponds to an input example and is described by the complexity measures values. These examples are labeled according to their underlying regression function, namely pol1, pol3, pol5, sin1 or sin3. Therefore, we have three meta-datasets with 12 input features (T_2 was included in these experiments), 1000 examples and five classes (with 200 examples per class/function type). The majority error rate for each dataset is 0.8.

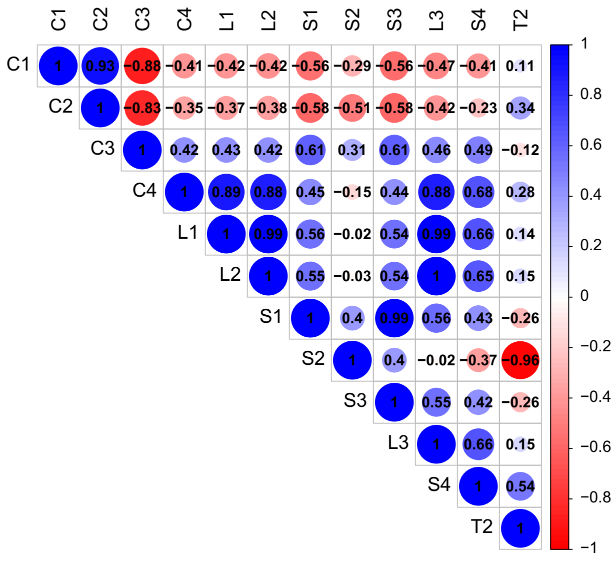
Using the Weka tool with default parameter values, Random Forests (RF), Multilayer Perceptron Neural Networks (MLP), Support Vector Machines (SVM) and 3-nearest neighbor (3NN) classifiers were trained on these datasets, according to the 10-fold cross-validation methodology. The SMO algorithm was employed in the SVM induction. The average and standard deviation of the error rates achieved by the former classification techniques are shown in Table 3.

According to the results shown in Table 3, all classification techniques were successful in separating the noiseless datasets according to their function type. MLP achieved the lowest average error rate, while SVM obtained the worst results. More errors are verified for functions pol3 and pol5, which usually get confused in these cases. As noise is added to the base datasets, the recognition performance of all techniques is diminished (although it still remains above the majority error rate performance). This simple experiment evidences the joint ability of the measures to describe the synthetic datasets generated.

4.4 Correlation and PCA analysis

Figure 10 presents the pairwise correlation of all measures for the synthetic datasets. The darker the color (either blue or red), the stronger the correlation is (direct or inverse, respectively).

As expected, measures based on similar concepts show a high absolute correlation. This is the case of: C_1–C_2–C_3, which are all based on the Spearman correlation of each individual feature to the output; L_1–L_2–L_3, which are related to data linearity; and S_1 and S_3, which measure the smoothness of the regression function by regarding neighbor input examples. S_2 is the feature with most distinct values compared to the other measures, although it is highly correlated to T_2, which assesses data sparseness. In fact, S_2 computes the Euclidean distance



	C1	C2	C3	C4	L1	L2	S1	S2	S3	L3	S4	T2
C1	1	0.93	−0.88	−0.41	−0.42	−0.42	−0.56	−0.29	−0.56	−0.47	−0.41	0.11
C2		1	−0.83	−0.35	−0.37	−0.38	−0.58	−0.51	−0.58	−0.42	−0.23	0.34
C3			1	0.42	0.43	0.42	0.61	0.31	0.61	0.46	0.49	−0.12
C4				1	0.89	0.88	0.45	−0.15	0.44	0.88	0.68	0.28
L1					1	0.99	0.56	−0.02	0.54	0.99	0.66	0.14
L2						1	0.55	−0.03	0.54	1	0.65	0.15
S1							1	0.4	0.99	0.56	0.43	−0.26
S2								1	0.4	−0.02	−0.37	−0.96
S3									1	0.55	0.42	−0.26
L3										1	0.66	0.15
S4											1	0.54
T2												1

Fig. 10 Correlation plot of the measures

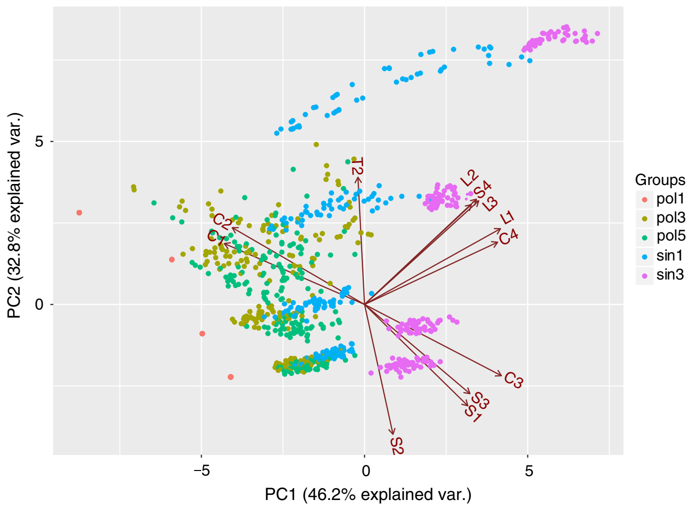


Fig. 11 Scatter plot with PCA values and loadings for all datasets with $\sigma = 0$

between input examples with similar outputs, which tends to be affected by data sparseness. Since our datasets have always the same number of examples, despite the number of input features, the results obtained for S_2 may be an artifact related to the increased data sparseness as more dimensions are added.

We also performed a PCA analysis using the measures values calculated for all synthetic datasets. Herewith, the first four components explain 95% of data variance (Appendix Table 10). Figure 11 shows a scatter plot with the two first components and the variable loadings for datasets with $\sigma = 0$. The first component, responsible for 46% of data variance, separates data from Sinusoidal datasets (high PC1 values) from lower polynomials (low PC1 values). All features, but T_2 and S_2, contribute to this component, which captures the complexity of the datasets. Two features which captures the sparsity of the datasets, T_2 and S_2, dominate PC2. Indeed, this PC separates data generated by distinct functions in four groups, as a reflex of the four numbers of input variables used in the data simulation. Of note, pol1 datasets per dimension are all located in the same position, as they all have similar complexity values. For datasets with $\sigma = 0.5$, we observe a similar but more diffuse distribution of the datasets (Appendix Fig. 15). The distribution of the datasets gets very diffused and the groups of functions overlap more in the case of $\sigma = 1.0$ (Appendix Fig. 16).

5 Experiments with real datasets: parameter selection

Some regression techniques require the user to define the values of one or more parameters. This is the case of the Support Vector Regression (SVR) technique. While MTL has proved to be a suitable and useful strategy for recommending the parameter values of SVR, its results are affected by the quality of the meta-features employed to describe the base datasets (Soares et al. 2004).

In this case study, we investigate the use of the proposed measures to recommend the best configurations of the following parameters of SVR: the γ parameter of RBF kernel, the regularization parameter C and the ϵ parameter. The choice of RBF kernel is due to its flexibility compared to other kernels (Keerthi and Lin 2003). The γ parameter has an important influence in learning performance since it controls the linearity of the induced SVR (Soares et al. 2004). The parameter C is also important for learning performance since it controls the complexity (flatness) of the regression function derived by the SVR (Basak et al. 2007). The ϵ parameter denotes a trade-off between model complexity and the amount of error allowed per training instance (Basak et al. 2007). Thus, if ϵ is greater than the range of the target values, a good result can not be expected. On the other hand, if ϵ is zero, the model can overfit.

The three previous parameters affect the complexity of the regression model induced in SVR, and therefore its generalization performance for a given problem. The best configuration of parameters will depend on the complexity of the learning problem at hand, i.e., the complexity of the model has to match the complexity of the problem. Therefore, we expect that the complexity measures will be useful to identify how difficult is a problem and thus to recommend a suitable configuration of parameters for the SVR.

In this case study, a meta-base was generated from the application of SVRs to 39 different regression problems collected from public data repositories. An instance-based meta-learner was employed to recommend the best parameter configuration for new problems based on their meta-features.

5.1 Meta-base

To generate meta-examples, we collected 39 datasets corresponding to 39 different regression problems from different domains, available at the UCI Repository (Bache and Lichman 2013). The list of regression problems adopted in our work is presented in Table 4, along with their number of examples (in parenthesis). The minimum and maximum values of the number of examples in the adopted datasets are 13 and 1, 200, respectively. Concerning the number of predictive attributes, the minimum and maximum values are 2 and 60 attributes. The diversity of domains and characteristics is convenient to MTL.

Table 4 Regression problems adopted for meta-example generation and their respective number of examples (in parenthesis)

Auto price (159)	Ele-1 (495)	MBA grade (61)
Auto MPG6 (392)	Ele-2 (1052)	Mortgage (1049)
Auto MPG8 (392)	Elusage (55)	Pollution (60)
Baseball (337)	Fertility diagnosis (100)	Pwlinear (200)
Baskball (96)	Forest fires (517)	Pyrim (75)
Bodyfat (252)	Friedman (1200)	Sensory (576)
Bolt (40)	Fruitfly (125)	Servo (167)
Cloud (108)	Gascons (27)	Stock (950)
Concrete (1030)	Housing (506)	Strike (625)
CPU (209)	Laser (993)	Treasury (1049)
Dee (365)	Longley (16)	Triazines (187)
Detroit (13)	Lowbwt (189)	Vineyard (52)
Diabetes numeric (43)	Machine CPU (209)	Yacht hydrodynamics (308)

Each meta-example is related to a single regression problem and stores: (i) the values of its meta-features (as those described in Sect. 3); and (ii) the best configuration of parameters evaluated for the problem. The best configuration stored in a meta-example is defined from the exhaustive evaluation of $3,192$ configurations of parameters: the parameter γ assumed 19 different values (from 2^{-15} to 2^{3}, increasing in powers of two), the parameter C assumed 21 different values (from 2^{-5} to 2^{15}, increasing in powers of two) and the parameter ϵ assumed 8 values (from 2^{-8} to 2^{-1}, increasing in powers of two). For each of combination, a 10-fold cross-validation experiment was performed to evaluate the SVR performance. The configuration of parameters with the lowest *Normalized MSE* (NMSE) was then stored in the meta-example. In these experiments, we deployed the *Scikit-Learn* package (Pedregosa et al. 2011) to implement the SVRs and to perform the cross-validation procedure.

5.2 Meta-learner

Given a new input regression problem, the Meta-Learner retrieves the most similar meta-example, according to the lowest distance between their meta-features. The distance function (*dist*) implemented in the prototype was the unweighted L_1-Norm, defined as:

$$dist(\mathbf{x}, \mathbf{x}_i) = \sum_{j=1}^{m} \frac{|x^j - x_i^j|}{\max_i\left(x_i^j\right) - \min_i\left(x_i^j\right)}, \tag{16}$$

where m is the number of meta-features, $\mathbf{x}$ is the new dataset under analysis and $\mathbf{x}_i$ are the examples from the meta-base. This is the same distance function adopted in Gomes et al. (2012) in a relate study where MTL is applied to SVR parameter tuning.

The meta-learner suggests for the new problem the configuration of parameters stored in the retrieved meta-example. The main assumption here is that the best configuration of parameters in a problem can also achieve good results in a similar problem.

5.3 Results

The proposed meta-features were evaluated on the recommendation of SVR parameters by following a leave-one-out methodology described as follows. At each step of leave-one-out, one meta-example is left out to be adopted as the test problem, and the remaining 38 meta-examples are kept in the meta-base to be selected by the meta-learner. Once the most similar meta-example is identified in the meta-base, its solution (configuration of parameters) is evaluated in the input problem and the NMSE obtained is stored.

The recommendation results achieved by using the proposed meta-features were compared with four other different approaches. Initially, we compared the proposal results with results achieved by the SVR using default parameters. The default parameter values are: $C = 1.0$, $\gamma = \frac{1}{d}$, $\epsilon = 0.1$, where d is the number of input attributes in the regression problem. We also investigate how close the results achieved by our proposal reached the solutions found by a *Grid search* (exhaustive approach). We point out that all used approaches are implementations of the *Scikit-Learn* package (Pedregosa et al. 2011). Finally, we compared our results with the combination of two sets of meta-features originally proposed by Soares et al. (2004) and Soares and Brazdil (2006). The combination of these meta-features was also validated in Gomes et al. (2012) for SVR parameter selection.

Table 5 shows the NMSE values reached by each approach in each regression problem. Lowest NMSE values among the Default, Soares and Proposal solutions are highlighted in boldface for each dataset. The Grid performs an exhaustive search and will obtain the best results most of the times. The table also presents, in the last row, the average ranking of the performance of all techniques. Therein, the Grid algorithm was the best performing technique, with an average ranking performance of 1.33. The use of the default parameters, on the other hand, was the worst performing configuration. The default parameter results compared to the others highlights the importance of a proper parameter tuning in order to maximize the predictive performance of the SVR technique.

According to the results shown in Table 5, the NMSE values reached by the proposal were in general better than those achieved by the SVR using default parameters as defined in Pedregosa et al. (2011). More precisely, the proposal won in 38 out of the 39 regression problems and lost in only 1 dataset (*Sensory*). The grid search exhaustively considers all parameter combinations, finding, this way, the best solution among those presented in the set of possible solutions. Our intention on comparing the recommendation results achieved by the proposal with the optimal results found by the grid search is to verify whether the proposal can recommend solutions near to the optimal. Considering 95% of confidence level between the results of the approaches according to the Wilcoxon statistical test, it can be observed that there is a similarity between these results in 30 out of 39 problems. Finally, the proposal achieved better NMSE values than the meta-features employed in Soares et al. (2004) in most of the problems (18 wins, 16 draws and 5 losses). To perform a fair comparison between both approaches, we also considered 95% of confidence level in the Wilcoxon statistical test to analyze the results. Thus, 5 wins, 2 losses and 32 draws were accounted to the proposal regarding the counterpart. The meta-features from Soares et al. (2004) considered both commonly used meta-features for the characterization of regression problems and meta-features specifically designed for SVR, that are based on the Kernel matrix. Thus, good results are expected by adopting this baseline. Our experiments show that our proposed generic meta-features are competitive compared to this strong baseline and can be useful to characterize regression problems in the SVR domain.

A Friedman statistical test detects that the results of the techniques compared can not be regarded as identical, at 95% of confidence level. The Nemenyi post-test points no differences

Table 5 NMSE values reached by the approaches

Dataset	Default	Soares	Proposal	Grid
Auto price	0.118	0.012	**0.011**	0.008
AutoMPG6	0.056	0.005	0.005	0.005
AutoMPG8	0.056	0.005	0.005	0.005
Baseball	0.134	0.044	**0.015**	0.013
Baskball	0.047	**0.017**	0.018	0.017
Bodyfat	0.039	0.001	0.001	0.001
Bolt	0.141	0.021	**0.016**	0.008
Cloud	0.116	0.005	0.005	0.004
Concrete	0.050	**0.007**	0.009	0.005
Cpu	0.196	0.002	**0.001**	0.000
Dee	0.050	0.010	**0.009**	0.008
Detroit	0.098	0.026	**0.024**	0.016
Diabetes numeric	0.044	0.030	**0.028**	0.026
Ele-1	0.101	0.007	0.007	0.006
Ele-2	0.134	0.000	0.000	0.000
Elusage	0.090	0.021	**0.015**	0.014
Fertility diagnosis	0.252	0.720	**0.107**	0.100
Forestfires	0.230	0.003	0.003	0.003
Friedman	0.035	0.002	0.002	0.002
Fruitfly	0.087	0.040	0.040	0.038
Gascons	0.105	0.003	**0.002**	0.001
Housing	0.055	**0.010**	0.040	0.010
Laser	0.102	**0.001**	0.002	0.000
Longley	0.125	0.003	**0.002**	0.001
Lowbwt	0.031	0.015	0.015	0.014
Machine cpu	0.190	0.011	**0.004**	0.003
Mbagrade	0.060	0.048	**0.045**	0.044
Mortgage	0.081	0.000	0.000	0.000
Pollution	0.042	0.019	**0.016**	0.013
PwLinear	0.041	0.006	**0.005**	0.005
Pyrim	0.074	0.013	0.013	0.011
Sensory	**0**.028	0.067	0.031	0.026
Servo	0.151	0.008	0.008	0.008
Stock	0.062	**0.008**	0.012	0.005
Strike	0.209	0.006	0.006	0.006
Treasury	0.096	0.000	0.000	0.000
Triazines	0.081	0.025	**0.021**	0.021
Vineyard	0.072	0.016	0.016	0.008
Yacht hydrodynamics	0.168	0.006	**0.005**	0.001
Ranking	3.92	2.56	2.21	1.33

Best results achieved per noise level are highlighted in bold between the results of the Proposal and of the Soares' meta-features. This reinforces the good performance of our meta-features, which achieved results comparable to those of the strong baseline of Soares et al. (2004), Soares and Brazdil (2006).

6 Experiments with real datasets: meta-regression

The data complexity measures proposed in our work try to characterize the difficulty of regression problems on a variety of situations. We expect that the proposed measures are actually related to the empirical performance obtained by regression algorithms once evaluated on a diverse set of regression problems.

In order to verify this statement, we performed a case study in which the proposed measures are adopted as meta-features to predict the expected performance of some regression algorithms. The case study is defined as a *meta-regression* task, in which the independent attributes are the complexity measures and the target attribute is the error obtained by a regression algorithm. This is a natural case study, since the performance of algorithms and the estimated difficulty of problems should be directly related. The usefulness of a set of data complexity measures depends on the ability of assessing the difficulty of different problems for different algorithms.

In this case study, the complexity measures were adopted to predict the performance of fourteen different regression algorithms implemented in the Weka environment (i.e., fourteen different instances of the meta-regression task were considered in this case study). The adopted algorithms correspond to different families of regression techniques, with different biases and assumptions, ranging from simple linear regression to more sophisticated kernel methods (see Table 6). By adopting a diverse set of regression algorithms, we aim to verify how robust the proposed measures are to characterize the intrinsic difficulty of regression problems.

All algorithms were executed using their default parameters as defined in Weka, apart from IBk (for which we considered two values for the number of nearest neighbors) and the kernel-based methods (for which we considered different options for the kernel function).

6.1 Meta-base

In this case study, a meta-base is built for each of the fourteen considered base-learners (in our case, a regression algorithm). Given a base-learner, a meta-base was built from the 39 regression problems collected in our previous experiments. Each meta-example was generated from a regression problem and we stored: (1) the meta-attributes describing the regression problem; (2) the NMSE error produced by the base-learner for that problem in a 10-fold cross-validation (CV) experiment.

In the CV experiments, we observed a varied learning performance depending on the regression problem considered. In fact, the best algorithms for each problem varied a lot and statistical differences between the base-learners were commonly observed. Despite this variability, the correlation of the NMSE values obtained by the base-learners across the datasets was commonly high (the average correlation was 0.8). Each regression problem has an intrinsic difficulty that homogeneously affected the performance level of all regression algorithms. We could say that part of the learning performance obtained by a base-learner on a problem is explained by its specific skills (which is related to the relative complexity concept briefly discussed in Sect. 3) and the remaining part is explained by the difficulty of the problem itself.

Table 6 Algorithms adopted in the case study of meta-regression

Acronym	Algorithm
A1	Linear regression
A2	Least med. square
A3	IBk (k=3)
A4	IBk (k=5)
A5	M5P
A6	REPTree
A7	MLP network
A8	RBF network
A9	GP (kernel Poly, p=1)
A10	GP (kernel Poly, p=2)
A11	GP (kernel RBF)
A12	SVM (kernel Poly, p=1)
A13	SVM (kernel Poly, p=2)
A14	SVM (kernel RBF)

In order to assess problem difficulty in this case study, we evaluated two different sets of meta-attributes: the proposed set of measures and the meta-features proposed in Soares et al. (2004) and Soares and Brazdil (2006), adopted here as a baseline of comparison. This is the same set of meta-features named as Soares in the previous Section. Therefore, considering the set of 14 base-learners and 2 meta-feature sets, the total of 28 different meta-bases were built in our experiments.

6.2 Meta-learner

As the meta-learning task in this case study is to predict the error rate of a given base-learner, the meta-learner is also a regression algorithm. In this case study, we adopted the SVR algorithm (with RBF Kernel), since is has shown to be a very competitive and robust algorithm for regression.

The performance of the SVR meta-regressor was estimated by a 10-fold CV experiment (with 10 repetitions). The performance was measured using the NMSE error (as adopted for estimating the performance of the base-learners). In this experiment, we employed a t-test to verify whether the NMSE value was statistically lower than 1, which would indicate that the adopted meta-features actually bring useful information to predict the algorithm performance.

6.3 Relevance of complexity measures

Before presenting the predictive results obtained by the meta-learner, we investigated the relevance of the complexity measures and their relationship with the performance obtained by the base-learners. In this section, we aim to provide insights about the usefulness of the proposed complexity measures.

Initially, we analyze the correlation between the complexity measures and the algorithms performance. In order to summarize the algorithm performance in a lower dimension, we applied PCA on the 39 (datasets) $\times$ 14 (base-learners) matrix of NMSE values. Next we ploted each proposed measure against the first principal component derived from the NMSE

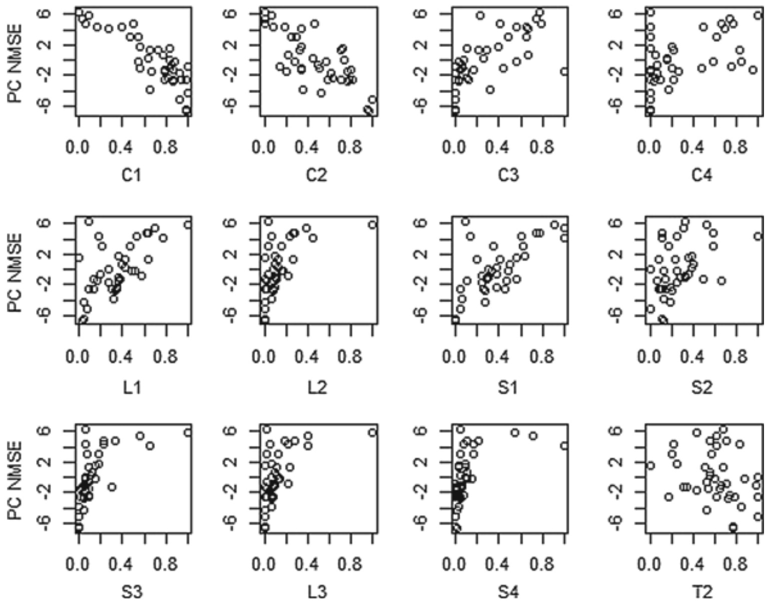


Fig. 12 Scatter plots of complexity measures (x-axis) and the first principal component (y-axis) derived from the NMSE values of the base-learners

Table 7 Correlation among the proposed complexity measures and the NMSE values obtained by the base-learners on the regression problems adopted

PC	C_1 − 0.86	C_2 − 0.75	C_3 0.66	C_4 0.48	L_1 0.59	L_2 0.57	S_1 0.67	S_2 0.44	S_3 0.61	L_3 0.56	S_4 0.54	T_2 − 0.29*
A1	− 0.66	− 0.69	0.52	0.35	0.37	0.42	0.47	0.52	0.46	0.43	0.40	− 0.37
A2	− 0.71	− 0.70	0.55	0.17*	0.30*	0.38	0.47	0.39	0.47	0.39	0.45	− 0.22*
A3	− 0.69	− 0.70	0.53	0.23*	0.35	0.40	0.53	0.41	0.48	0.41	0.46	− 0.28*
A4	− 0.68	− 0.64	0.46	0.18*	0.42	0.58	0.51	0.53	0.62	0.60	0.44	− 0.31*
A5	− 0.56	− 0.72	0.43	0.39	0.37	0.38	0.39	0.39	0.38	0.39	0.34	− 0.23*
A6	− 0.70	− 0.71	0.55	0.28*	0.40	0.47	0.55	0.47	0.54	0.47	0.49	− 0.31
A7	− 0.56	− 0.54	0.38	0.29*	0.32	0.40	0.39	0.30*	0.42	0.41	0.38	− 0.19*
A8	− 0.59	− 0.53	0.36	0.28*	0.33	0.45	0.40	0.32	0.48	0.48	0.42	− 0.19*
A9	− 0.64	− 0.75	0.52	0.26*	0.41	0.42	0.44	0.52	0.43	0.44	0.39	− 0.25*
A10	− 0.66	− 0.69	0.52	0.22*	0.38	0.40	0.42	0.48	0.42	0.41	0.38	− 0.21*
A11	− 0.55	− 0.64	0.39	0.51	0.42	0.41	0.49	0.34	0.41	0.41	0.39	− 0.34
A12	− 0.69	− 0.62	0.49	0.21*	0.34	0.44	0.53	0.42	0.54	0.45	0.50	− 0.30*
A13	− 0.63	− 0.57	0.50	0.15*	0.30*	0.35	0.56	0.39	0.48	0.35	0.46	− 0.38
A14	− 0.56	− 0.68	0.41	0.46	0.38	0.37	0.47	0.46	0.40	0.37	0.32	− 0.40

The symbol * indicates that correlation is not greater than 0 with 95% level of confidence

matrix (see Fig. 12). Table 7 (first line) presents the values of correlation associated to the plots in Fig. 12.

As expected (by definition), we observed a positive correlation between the regression error and the measures C_3, C_4, L_1, L_2, S_1, S_2, S_3, L_3 and S_4. In turn, negative correlation

Table 8 Centroid of regression problems obtained by *k-Means*

	Cluster 1 high complex	Cluster 2 low complex
C_1	0.383	0.830
C_2	0.207	0.620
C_3	0.593	0.108
C_4	0.512	0.170
L_1	0.521	0.252
L_2	0.256	0.068
S_1	0.589	0.261
S_2	0.404	0.194
S_3	0.275	0.065
L_3	0.236	0.057
S_4	0.236	0.038
T_2	0.536	0.641

was observed for C_1, C_2 and T_2. The correlation values suggest the relative importance of each attribute. Most correlations were statistically significant (except for T_2).

Figure 12 indicates that some measures (like C_1, C_2, C_3 and S_1) present a more regular linear relationship with the learning performance. Other measures (like L_2, S_3, L_3 and S_4) seem to present an exponential behavior in the sense that small modifications in their values indicate a high increase in the regression errors. Such measures may be less useful for identifying some of the complex problems (i.e., problems may be considered more simple than they really are).

Table 7 also details the correlation between the proposed measures and the performance of the base-learners. Feature correlation measures C_1, C_2 and C_3 were notably related to regression performance (i.e., high feature correlation to the target attribute results in low regression error). Other complexity measures present intermediate correlations (mostly statistically significant).

The less correlated measures were C_4 and T_2, for which significance was only verified in 4 and 5 base-learners respectively. Nevertheless, some of these significant correlations were observed for the Gaussian Processes (A9–A11) and SVRs using the RBF kernel (A14). This is an interesting result to be further explored in the future, since these base-learners potentially produce more complex regression surfaces. Although these measures may be not useful as the other measures in isolation to predict problem complexity in this setup, they can be useful in combination to other measures.

Another way to investigate the relevance of the complexity measures is to cluster the regression problems according to the measures and analyze the performance obtained by the base-learners in each cluster. By applying the *k-means* algorithm (with $k = 2$), we produce two clusters, which we associated respectively to low and high complexity. Table 8 presents the centroid of each cluster. For each base-learner, the average NMSE was higher in the high complexity cluster (Appendix Fig. 17). The cluster property can also be seen when the PCA is applied on the complexity measures and plotted against the first principal component originally derived for the NMSE matrix (see Fig. 13). There, problems with similar NMSE values also tend to be closer in the reduced space of complexity.

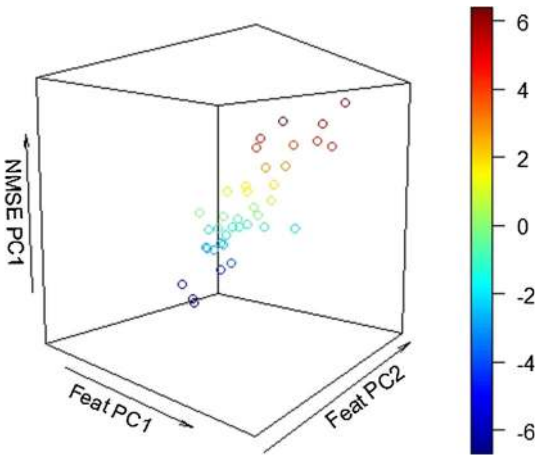


Fig. 13 Projection of the regression problems in the space of complexity measures versus regression performance. The *x* and *y* axes represent the principal components extracted from the complexity measures. The *z* axis in turn represents the first component previously extracted from the NMSE values. Colors are set according to NMSE component, in such a way that red–yellow–blue scale indicates high-to-low regression error

Table 9 Average NMSE values (and standard deviations) obtained by the SVR meta-regression for each meta-learning task considering different base-learners and meta-feature sets

Base learner	Meta-feature set Baseline (Soares)		Proposal	
A1	0.83 (0.18)		0.68 (0.13)	•
A2	0.85 (0.15)		0.79 (0.07)	•
A3	0.78 (0.19)		0.64 (0.15)	•
A4	0.81 (0.08)		0.72 (0.10)	•
A5	0.86 (0.11)		0.74 (0.10)	•
A6	0.89 (0.08)		0.79 (0.10)	•
A7	0.80 (0.16)		0.69 (0.09)	•
A8	0.94 (0.08)		0.77 (0.11)	•
A9	0.91 (0.08)		0.83 (0.09)	•
A10	0.88 (0.10)		0.74 (0.11)	•
A11	0.82 (0.15)		0.65 (0.13)	•
A12	0.82 (0.10)	•	0.69 (0.09)	•
A13	0.87 (0.08)	•	0.76 (0.10)	•
A14	0.86 (0.10)		0.78 (0.08)	•

The symbol • indicates when the NMSE value was statistically lower than 1 with 95% level of confidence

6.4 Meta-regression results

Finally, we present the predictive results of the SVR meta-learner in the meta-regression tasks. Table 9 presents the average NMSE values obtained by the meta-regressor using the two adopted sets of meta-features. As it can be seen, by adopting the proposed complexity measures as meta-features, we observed lower NMSE values for all base-learners considered. Additionally, the NMSE values were statistically lower than 1 (thus indicating the usefulness of the meta-features) in all base-learners when the proposed meta-features were adopted. In turn, although the NMSE values obtained with the baseline set of meta-features were all lower than 1, the results were only statistically significant for two base-learners (the SVRs with polynomial Kernel). We could argue that as the proposed measures are general enough to predict the performance of diverse regression algorithms, the baseline set is more specific.

7 Conclusion

This paper presented and formalized a set of meta-features that can be employed to characterize the complexity of regression problems. Thereby, they estimate how complex should be the function that must be fitted to the data in order to solve the regression problem. The utility of these measures in the description of regression problems was assessed in three MTL setups: (i) predicting the regression function type of some datasets; (ii) determining the parameter values of SVR regressors; and (iii) predicting the expected NMSE of various regressors. While the first analysis employs synthetic datasets containing known regression functions, the last two use real regression datasets from public repositories.

Experiments with synthetic datasets revealed that many of the individual measures were able to separate simple problems from more complex variants. However, each measure provides a perspective regarding the problem complexity. Since several aspects can influence the complexity of a regression problem, a combination of various measures should be used to describe a problem. This is supported by the success in the use of the complexity measures as meta-features in a MTL system designed to predict the regression function generating the data. In other two setups the complexity measures are also jointly employed as meta-features to characterize a set of 39 regression problems from public repositories. For these datasets, good predictive results were obtained in two distinct meta-learning setups: the first one designed to tune the parameter values of the Support Vector Regression technique and the second one used to predict the expected NMSE of various regression techniques.

PCA analysis further supports the importance and association between proposed measures. The PCA analysis on simulated data revealed that most measures are important to characterize data complexity. There, the first component contained similar contributions of all measures, except from T_2 and S_2, which were in turn prevalent in the second PC component. PCA and correlation analysis also indicates that some groups of variables are redundant, i.e. C_1–C_2, S_1–S_3. Similar importances were found in the association of complexity measures with the first PCA of the NMSE of all evaluated regression methods for real data. Here, features of correlation (such as C_1, C_2 and C_3) and of the data smoothness (as S_1) were most effective in the discrimination of data complexity.

It is worth noting that the calculated values of the measures give an apparent estimate of the regression problem complexity. This happens because their values are estimated from the dataset available for learning, which contains a sample of the problem instances.

7.1 Future work

Future work shall investigate the ability of other measures from relate literature that can be used to characterize the difficulty of regression problems. Examples include the proportion of features with outliers and the coefficient of variation of the target used in Soares et al. (2004) and Gomes et al. (2012). If there are many features with outliers, the task of fitting a function to the data will be impaired. More complex functions may also be required if the outputs vary too much. Other candidate measure is the coefficient of determination from the Statistics literature, which can be used to measure data conformation to a linear model. Therefore, there is already a range of measures in the literature which, although have not been analyzed with the aim of characterizing the complexity of regression problems, can be directly used or adapted for such.

Some of the proposed measures can also be reformulated to become more robust. An example is the C_3 measure. We considered a threshold of 0.9 as a high correlation value, but other values could be employed instead to decide whether the correlation is significant.

The same reasoning applies to the C_4 measure and the threshold employed for the residuals values.

The values of some measures are also very dependent on the number of features present on a dataset. This issue should be investigated further in future work. For correlation measures, for example, the higher the number of features, the lower their individual correlation values tend to be. This happens because a combination of the features will correlate to the output. Since correlation does not imply causality and can be observed at random (Armstrong 2012), some caution must also be taken in their interpretation. Nonetheless, in most of the ML datasets we do expect that most of the features are related to the domain and that the causality component is present.

The formulated measures can only be applied to datasets containing quantitative/numerical features. For nominal features, the measures must be adapted or changed, although it is also possible to map the symbolic values into numerical values and apply the standard measures to the resulting values.

Finally, the proposed measures can be evaluated in different case studies for selecting regression algorithms or hyperparameter values using conventional meta-learners or combined with search strategies, as described in Wistuba et al. (2016) and Leite et al. (2012). Such case studies will reveal in which situations the proposed measures are more or less useful and will provide insights for new proposals.

Acknowledgements To the research agencies FAPESP (2012/22608-8), CNPq (482222/2013-1, 308858/2014-0 and 305611/2015-1), CAPES, DAAD and IZKF Aachen for the financial support.

8 Appendix

This section presents the Supplementary Material from the paper. It contains the algorithms of most of the measures and some additional graphs which complement the paper results.

8.1 Algorithms for complexity measures

The description of the algorithms of most of the complexity measures are presented next. An analysis of their asymptotic complexities is also presented.

8.1.1 Algorithm—individual feature efficiency (C_3)

A naïve implementation of C_3 would have a cubic complexity on the number of samples. We propose therefore an efficient implementation of C_3 as described in Algorithm 1. For this, we take advantage of the fact the Spearman correlation works on differences of rankings. Instead of re-estimating the rankings after the removal of each observation, we simply update the difference of the rankings. The proposed algorithm has a worst time complexity of $\mathrm{O}(d \cdot n^2)$.

That is, for each feature $\mathbf{x}^j$ of the original dataset X, two ranks are obtained: one by the order of the values in the vector $\mathbf{x}^j$ ($Rank_x$ in line 2) and other by the order of the values in $\mathbf{y}$ ($Rank_y$ in line 3). Next the difference of these rankings is calculated (line 4). Taking this difference vector as input, the ρ' value is calculated in line 5, according to Eq. 17.

$$\rho'(\mathbf{d}) = 1 - \frac{6\sum d_i^2}{(n^3 - n)}, \tag{17}$$

where $\mathbf{d}$ is a vector containing the differences of the rankings and n is its number of elements. This is the standard definition of the Spearman's rank correlation coefficient if the ranks have no ties. Here, tied elements are ranked according to their original order of appearance. Therefore, the calculated value can be considered an heuristic or approximation (that is why it is denoted as ρ' in Algorithm 1), which will hold tight only when there are no repeated values in both $\mathbf{x}^j$ and $\mathbf{y}$ vectors.

Algorithm 1 Algorithm for C_3 computation

```
Require: A data matrix $X$ and a label vector $\mathbf{y}$;
1: for all (feature vectors $\mathbf{x}^j \in X$) do
2:    $Rank_x$ = rank of $\mathbf{x}^j$;
3:    $Rank_y$ = rank of $\mathbf{y}$;
4:    $Rank_{dif} = Rank_x - Rank_y$;
5:    if ($\rho'(Rank_{dif}) < 0$) then
6:       $Rank_y$ = reverse ranking of $\mathbf{y}$;
7:       $Rank_{dif} = Rank_x - Rank_y$;
8:    end if
9:    while ($|\rho'(Rank_{dif})| \leq 0.9$) do
10:      $id_r = \max(\text{abs}(Rank_{dif}))$;
11:      for all $(id_j|((Rank_x(id_j) > Rank_x(id_r))$ and $(Rank_y(id_j) < Rank_y(id_r))))$ do
12:         $Rank_{dif}(id_j) = Rank_{dif}(id_j) - 1$;
13:      end for
14:      for all $(id_j|((Rank_y(id_j) > Rank_y(id_r))$ and $(Rank_x(id_j) < Rank_x(id_r))))$ do
15:         $Rank_{dif}(id_j) = Rank_{dif}(id_j) + 1$;
16:      end for
17:      Remove element $id_r$ from $Rank_{dif}$, $Rank_x$ and $Rank_y$;
18:   end while
19:   $n^j$ = size($Rank_{dif}$);
20: end for
21: return $\frac{\min(n^j)}{n}$;
```

If the correlation between the j-th feature vector and the output is negative, $Rank_y$ is reversed (line 6). This is done because we are interested in the absolute values of the correlation only. Examples are successively removed in the while loop from lines 9 to 18. Since the Spearman correlation is based on the difference between the ranks of $\mathbf{x}^j$ and $\mathbf{y}$, the best candidate example for removal at each step will be that one which shows a higher ranking difference (id_r in line 10). By eliminating examples with most dissimilar rankings, the correlation is increased quickly. The operations in lines 11–13 and 14–16 will update the rankings differences. When id_r is removed, the ranking of all elements ranked after id_r in $Rank_x$ and $Rank_y$ must be decremented. But the $Rank_{dif}$ vector will be affected only for indexes id_j for which one of the two original rankings changed. If both of them are decremented or do not change, their difference remains the same. For elements id_j where $Rank_x(id_j)$ changes (that is, for which $Rank_x(id_j) > Rank_x(id_r)$) while $Rank_y(id_j)$ remains the same (ie, $Rank_y(id_j) < Rank_y(id_r)$), $Rank_{dif}(id_j)$ decreases in one unit. On the other hand, when $Rank_y(id_j)$ decreases and $Rank_x(id_j)$ maintains its value, $Rank_{dif}(id_j)$ increases by one unit.

After a correlation higher or equal to 0.9 is achieved, n^j computes the size of the differences vector $Rank_{dif}$. In the end, the algorithm returns the minimum of the n^j values found (line 19). Concerning asymptotic computational complexity, the loops in lines 1 and 9 will be executed up to d and n times, respectively. Moreover, most of the steps from the while loop (lines 10–17) require, in the worst case, going over all the remaining observations at least once. Therefore, the algorithm will have a worst time complexity of $O(d \cdot n^2)$.

8.1.2 Algorithm—individual feature efficiency (C_4)

The idea of C_4 is to systematically examine whether a simple linear fit of a given predictive variable to the output is already able to explain the relationship of the examples to the target variable. At each round one variable (the most correlated to the output so far) is considered. Those examples already explained by this variable according to the $|\varepsilon_i| \leq 0.1$ criterion are then eliminated and the next variable is considered. This is done until all examples are explained by some or all of the predictive variables or until no predictive feature remains.

Algorithm 2 illustrates how C_4 proceeds. The variable A, which holds the set of features to be analyzed, is initialized with $1, \ldots, d$ in line 1. In line 3 the index of the feature which shows the maximum absolute correlation to the output is calculated. The index of the chosen feature must be excluded from the set A (line 4). In line 5 a linear fit of the examples in X, described by the m-th feature vector only, is found. All examples for which the absolute value of the residual value is lower than 0.1 are removed from the dataset (lines 6, 7). This process is iterated until all features are analyzed or the dataset X becomes empty. The algorithm returns the rate of remaining examples (line 9).

Algorithm 2 Algorithm for C_4 computation

```
Require: A data matrix $X$ and a label vector $\mathbf{y}$;
1: Make $A = \{1, \ldots, d\}$;
2: while (($A \neq \{\}$) and ($X \neq \{\}$)) do
3:     $m = \{j \mid \max_{j \in A} |\rho(\mathbf{x}^j, \mathbf{y})|\}$;
4:     $A = A - \{m\}$;
5:     $f$ = LinearFit($\mathbf{x}^m, \mathbf{y}$);
6:     $X = X - \{\mathbf{x}_i \,||\epsilon_i| < 0.1\}$;
7:     $\mathbf{y} = \mathbf{y} - \{y_i \,||\epsilon_i| < 0.1\}$;
8: end while
9: return $\frac{\sharp X}{n}$;
```

The most expensive computational task in Algorithm 2 is estimating the correlation between the remaining variables and $\mathbf{y}$ in line 3 and finding their maximum value. In the worst case, the dataset will remain unchanged for all rounds and the correlation computation will always require $O(n \cdot \log n)$ operations. Since all features will be analyzed in this worst case scenario, we have a $O(d \cdot n \cdot \log n)$ cost for computing all correlations required. When finding the maximum among them, the set of features is decreased from d to 1, at steps of 1, giving the total computational cost of $O(d^2)$, given that the outer loop (line 2) will be executed for d times. This gives a total asymptotic complexity of $O(d^2 + d \cdot n \cdot \log n)$ for C_4 computation. It is easy to reduce this worst case time complexity to $O(d^2)$ by recomputing the correlations in line 3 only when there are changes in the dataset size. But usually there will be reductions in the dataset size between consecutive steps, so that the correlation will have to be recalculated (although using datasets of continuously reduced size). Also note that, since many examples can be removed at each round, an incremental update of the correlation value would not be so trivial as that performed in the case of C_3.

8.1.3 Algorithm—input distribution (S_2)

The computation of S_2 can be described by Algorithm 3. First the dataset is ordered according to the ascending order of the $\mathbf{y}$ values (line 1). The while loop from lines 3–6 computes the average *Euclidean distance* of neighbor examples in the ordered dataset.

Algorithm 3 Algorithm for S_2 computation

```
Require: A data matrix $X$ and a label vector $\mathbf{y}$;
1: Order $X$ according to the ascending order of the output values $\mathbf{y}$;
2: Make $i = 2$ and $d = 0$;
3: while $(i < n)$ do
4:     $d = d + \|\mathbf{x}_i - \mathbf{x}_{i-1}\|_2$;
5:     $i = i + 1$;
6: end while
7: return $d/n$;
```

Concerning the computational complexity, line 2 of Algorithm 3 requires $\mathrm{O}(n \cdot \log n)$ operations, while line 4 will be executed $n \cdot d$ times. In total, this measure has an asymptotic complexity of $\mathrm{O}(n \cdot (d + \log n))$.

8.1.4 Algorithm—error rate of a nearest neighbor regressor (S_3)

The computation of S_3 is described in Algorithm 4. First a distance matrix between all data items in X is obtained (line 1). The for loop in lines 3 to 6 seeks for the nearest neighbor of each example $\mathbf{x}_i$ (the example $\mathbf{x}_j$ with minimum distance to $\mathbf{x}_i$) and predicts its output according to the NN rule.

The asymptotic computational complexity of this measure is dominated by the estimation of the distance matrix of all pairs of examples in line 1, which costs $\mathrm{O}(d \cdot n^2)$ operations.

Algorithm 4 Algorithm for S_3 computation

```
Require: A data matrix $X$ and a label vector $\mathbf{y}$;
1: $DM = DistanceMatrix(X)$;
2: $error = 0$;
3: for all $\mathbf{x}_i \in X$ do
4:     $i_{NN} = \min_{j \neq i} DM(i, j)$;
5:     $error = error + (y_i - y_{i_{NN}})^2$;
6: end for
7: return $error/n$;
```

8.1.5 Algorithm—non-linearity of a linear regressor (L_3)

Algorithm 5 describes L_3 operation. A linear fit between the original input data X and the label vector $\mathbf{y}$ is found in line 1. The examples in X are ordered according to their output values in step 2. New random inputs with random labels are generated in steps 5 and 6 by interpolating the entries and outputs of examples with similar outputs. This process is repeated until all l examples are generated. A new dataset X' is then formed. The linear regressor found in line 1 has its MSE measured for the new data points generated.

The most computationally demanding step in this algorithm is performed in line 1, which requires the estimation of a linear regression with $\mathrm{O}(d^2 \cdot n)$. There is also an ordering step in line 2, requiring $\mathrm{O}(n \cdot \log n)$ operations. Creating new points in the loop has a cost of $\mathrm{O}(d \cdot l)$, that will correspond to $\mathrm{O}(d \cdot n)$ in our implementation of the measure. Therefore, the total cost of the algorithm is $\mathrm{O}(n(d^2 + \log n))$.

Algorithm 5 Algorithm for L_3 computation

```
Require: A data matrix $X$, a label vector $\mathbf{y}$;
1: $f(X) = \text{LinearFit}(X, \mathbf{y})$;
2: Order $X$ and $\mathbf{y}$ according to the ascending order of $\mathbf{y}$;
3: Make $i = 1$, $X' = \{\}$ and $\mathbf{y}' = \{\}$;
4: while ($i < n - 1$) do
5:     Create a new point $\mathbf{x}'_i$ by randomly interpolating $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$ and add it to $X'$;
6:     Create a label $y'_i$ by randomly interpolating $y_i$ and $y_{i+1}$ and add it to $\mathbf{y}'$;
7:     $i = i + 1$;
8: end while
9: return $MSE(f(X'), \mathbf{y}')$;
```

8.1.6 Algorithm—non-linearity of nearest neighbor regressor (S_4)

The algorithm for this measure is the same as presented in Algorithm 5, with an alteration in line 1, which is suppressed, and line 9, where f is replaced by a *nearest neighbor regressor*. The computation of the measure has an asymptotic cost of $O(n \cdot d \cdot \log n)$, using a KD-tree implementation of the NN algorithm.

8.2 Additional results

This section presents some additional results that were generated during the preparation of this paper (Table 10 and Figs. 14, 15, 16, 17).

Table 10 PCA analysis of synthetic datasets

PCA	PC_1	PC_2	PC_3	PC_4
C_1	– 0.36	0.19	0.09	– 0.42
C_2	– 0.34	0.23	– 0.03	– 0.44
C_3	0.35	– 0.22	– 0.12	0.31
C_4	0.34	0.19	– 0.04	– 0.26
L_1	0.35	0.23	0.20	– 0.12
L_2	0.29	0.32	0.30	– 0.09
S_1	0.27	– 0.31	– 0.31	– 0.38
S_2	0.07	– 0.40	0.50	– 0.13
S_3	0.27	– 0.27	– 0.32	– 0.50
L_3	0.29	0.32	0.29	– 0.08
S_4	0.27	0.30	– 0.14	0.08
T_2	– 0.02	0.39	– 0.54	0.14
Var.	0.46	0.33	0.11	0.05
Cum.	0.46	0.79	0.90	0.95

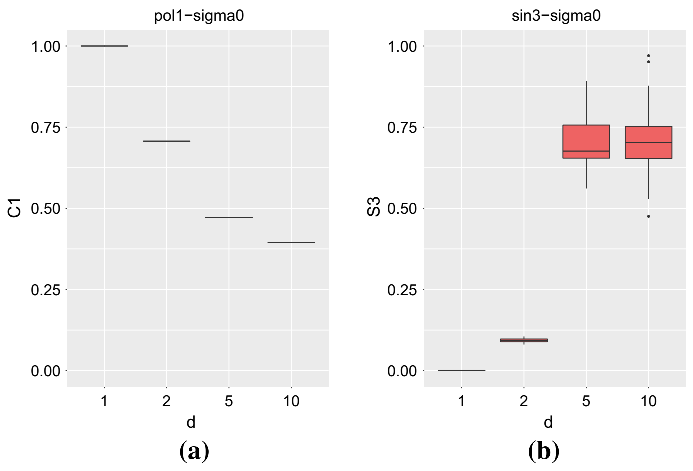


Fig. 14 Effect of data dimension on the results of some measures ($\sigma = 0$). **a** C_1 for pol1. **b** S_3 for sin3.

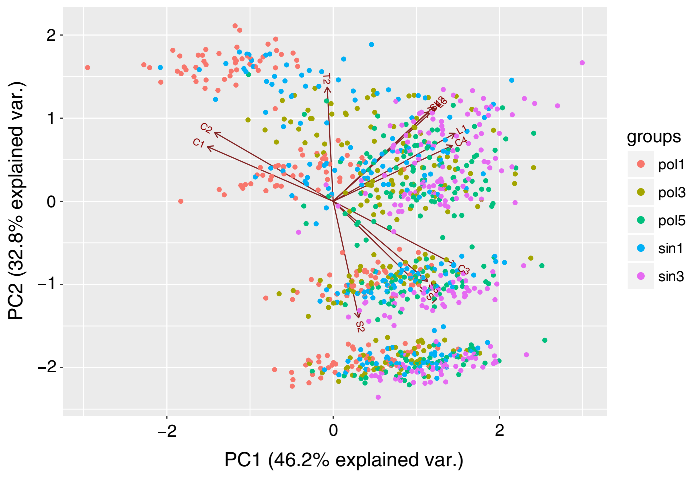


Fig. 15 Scatter plot with PCA values and loadings for all datasets with $\sigma = 0.5$

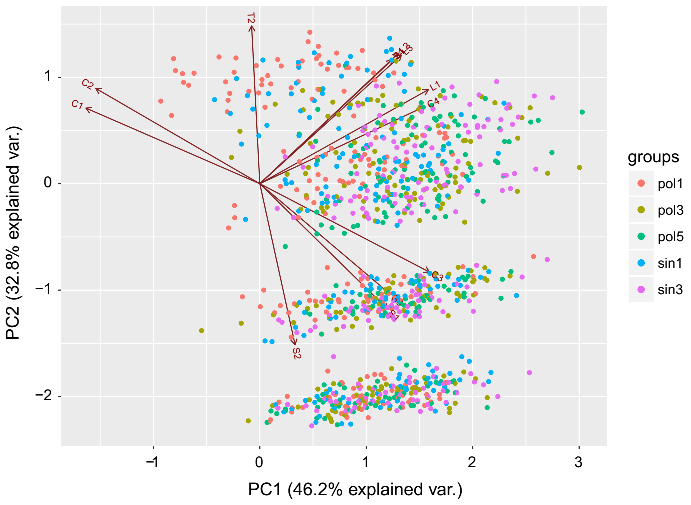


Fig. 16 Scatter plot with PCA values and loadings for all datasets with $\sigma = 1.0$

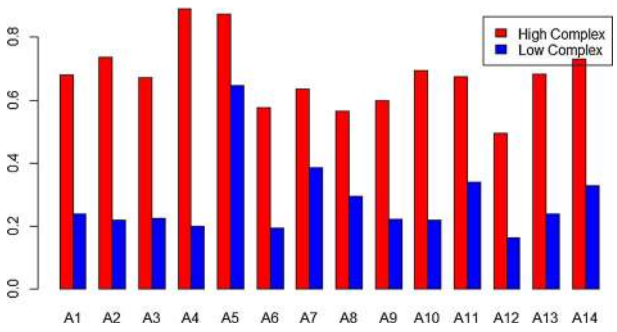


Fig. 17 Bar plot of average NMSE obtained for each base-learner in the problems respectively associated to the clusters of high and low complexity. Problems considered complex in the cluster analysis have higher NMSE values

References

Amasyali, M., & Erson, O. (2009). *A study of meta learning for regression*. Tech. rep. ECE Technical Reports 386, Purdue University.

Armstrong, J. S. (2012). Illusions in regression analysis. *International Journal of Forecasting*, *28*(3), 689–694.

Bache, K., & Lichman, M. (2013). *UCI machine learning repository*. http://archive.ics.uci.edu/ml, University of California, Irvine, School of Information and Computer Sciences.

Basak, D., Pal, S., & Patranabis, D. C. (2007). Support vector regression. *Neural Information Processing-Letters and Reviews*, *11*(10), 203–224.

Brazdil, P., Giraud-Carrier, C., Soares, C., & Vilalta, R. (2008). *Meta-learning: Applications to data mining*. New York: Springer Science and Business Media.

Cavalcanti, G., Ren, T., & Vale, A. (2012). Data complexity measures and nearest neighbor classifiers: A practical analysis for meta-learning. In: *IEEE 24th international conference on tools with artificial intelligence (ICTAI), 2012* (Vol. 1, pp. 1065–1069). IEEE.

Cristianini, N., Shawe-Taylor, J., Elisseeff, A., & Kandola, J. (2002). On kernel-target alignment. *Advances in Neural Information Processing Systems*, *14*, 367–373.

de Miranda, P., Prudêncio, R. B. C., Carvalho, A., & Soares, C. (2014). A hybrid meta-learning architecture for multi-objective optimization of SVM parameters. *Neurocomputing*, *143*, 27–43.

Garcia, L. P., de Carvalho, A. C., & Lorena, A. C. (2015). Effect of label noise in the complexity of classification problems. *Neurocomputing*, *160*, 108–119.

Garcia, L. P., de Carvalho, A. C., & Lorena, A. C. (2016). Noise detection in the meta-learning level. *Neurocomputing*, *176*, 14–25.

Gomes, T. A. F., Prudêncio, R. B. C., Soares, C., Rossi, A. L. D., & Carvalho, A. (2012). Combining meta-learning and search techniques to select parameters for support vector machines. *Neurocomputing*, *75*(1), 3–13.

Ho, T. K., & Basu, M. (2002). Complexity measures of supervised classification problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *24*(3), 289–300.

Keerthi, S. S., & Lin, C. J. (2003). Asymptotic behaviors of support vector machines with Gaussian kernel. *Neural Computation*, *15*(7), 1667–1689.

Kuba, P., Brazdil, P., Soares, C., & Woznica, A. (2002). Exploiting sampling and meta-learning for parameter setting for support vector machines. In: *VIII Iberoamerican conference on artificial intelligence proceedings of workshop learning and data mining associated with iberamia 2002*, (University of Sevilla, Sevilla (Spain), (pp. 209–216).

Leite, R., Brazdil, P., & Vanschoren, J. (2012). Selecting classification algorithms with active testing. In: *Proceedings of the 8th international conference on machine learning and data mining in pattern recognition* (pp. 117–131).

Leyva, E., Gonzalez, A., & Perez, R. (2015). A set of complexity measures designed for applying meta-learning to instance selection. *IEEE Transactions on Knowledge and Data Engineering*, *27*(2), 354–367.

Loterman, G., & Mues, C. (2012). Selecting accurate and comprehensible regression algorithms through meta learning. In: *IEEE 12th international conference on data mining workshops* (pp. 953–960).

Maciel, A. I., Costa, I. G., & Lorena, A. C. (2016). Measuring the complexity of regression problems. In: *IEEE proceedings of the 2016 international conference on neural networks* (**in press**).

Morán-Fernández, L., Bolón-Canedo, V., & Alonso-Betanzos, A. (2017). Can classification performance be predicted by complexity measures? A study using microarray data. *Knowledge and Information Systems*, *51*(3), 1067–1090.

Orriols-Puig, A., Maci, N., & Ho, T. K. (2010). *Documentation for the data complexity library in c++*. Tech. rep., La Salle—Universitat Ramon Llull.

Pappa, G. L., Ochoa, G., Hyde, M. R., Freitas, A. A., Woodward, J., & Swan, J. (2014). Contrasting meta-learning and hyper-heuristic research: The role of evolutionary algorithms. *Genetic Programming and Evolvable Machines*, *15*(1), 3–35.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, *12*, 2825–2830.

Smith, M. R., White, A., Giraud-Carrier, C., & Martinez, T. (2014). An easy to use repository for comparing and improving machine learning algorithm usage. Preprint. arXiv:14057292.

Soares, C. (2008). Development of metalearning systems for algorithm recommendation. In: P. Brazdil, C. Giraud-Carrier, C. Soares & R. Vilalta (Eds.), *Meta-learning: applications to data mining* (pp. 33–62). Springer.

Soares, C., & Brazdil, P. B. (2006). Selecting parameters of SVM using meta-learning and kernel matrix-based meta-features. In: *Proceedings of the 2006 ACM symposium on applied computing*, ACM, SAC '06, (pp. 564–568).

Soares, C., Brazdil, P. B., & Kuba, P. (2004). A meta-learning method to select the kernel width in support vector regression. *Machine Learning*, *54*(3), 195–209.

Thornton, C., Hutter, F., Hoos, H., & Leyton-Brown, K. (2013). Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In: *Proceedings of the 19th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 847–855).

Wistuba, M., Schilling, N., & Schmidt-Thieme, L. (2016). Two-stage transfer surrogate model for automatic hyperparameter optimization. In: *European conference on machine learning and knowledge discovery in databases* (pp. 199–214).