Reducing data complexity using autoencoders with class-informed loss functions VI.

Source

Charte, D., Charte, F., & Herrera, F. (2021). Reducing Data Complexity using Autoencoders with Class-informed Loss Functions. *IEEE Transactions on Pattern Analysis and Machine Intelligence.*

Abstract

Available data in machine learning applications is becoming increasingly complex, due to higher dimensionality and difficult classes. There exists a wide variety of approaches to measuring complexity of labeled data, according to class overlap, separability or boundary shapes, as well as group morphology. Many techniques can transform the data in order to find better features, but few focus on specifically reducing data complexity. Most data transformation methods mainly treat the dimensionality aspect, leaving aside the available information within class labels which can be useful when classes are somehow complex.

This paper proposes an autoencoder-based approach to complexity reduction, using class labels in order to inform the loss function about the adequacy of the generated variables. This leads to three different new feature learners, Scorer, Skaler and Slicer. They are based on Fisher's discriminant ratio, the Kullback-Leibler divergence and least-squares support vector machines, respectively. They can be applied as a preprocessing stage for a binary classification problem. A thorough experimentation across a collection of 27 datasets and a range of complexity and classification metrics shows that class-informed autoencoders perform better than 4 other popular unsupervised feature extraction techniques, especially when the final objective is using the data for a classification task.

VI.1. Introduction

A classical obstacle in the field of data science is obtaining data of sufficient quality in order to extract the desired knowledge. The process of learning a model can be notably hindered by data presenting very common traits such as noise [1], outliers [2], high dimensionality [3] or complex class boundaries [4]. This results in long periods of time spent cleaning and preprocessing data [5] before the actual data mining step can even begin. Although data cleaning techniques can be of good use in order to identify and filter out noise, outliers and missing data, other aspects can be trickier to solve.

Many real world situations can be modeled as supervised classification problems, those where each instance belongs to one of several classes, and the objective is to learn from the observed data from each class in order to automatically assign the corresponding class labels to new, unobserved instances. Some examples of classification problems are text categorization [6], spam filtering [7], object recognition in images [8] and automatic interpretation of medical data to facilitate diagnostics [9]. Many of these problems correspond to the simplest case, binary classification, where there are only two categories.

One of the type of issues that is very commonly overlooked in classification problems is the complexity of data [4, 10]. Consider a clean dataset with no presence of errors or abnormalities. There can still be aspects related to the geometrical shapes and overlap among classes which can hinder the performance of a learning technique. For example, there could be no separability between classes, or even regions of the feature space with a mix of instances from different classes. In cases where separability is achievable, boundaries can present complex shapes that can be difficult for a parameterized model to fit. Figure VI.1 illustrates some of these cases, more concretely, one where the features do not allow to separate the classes, and another where class boundaries are difficult to model due to there being several small groups from one class, sometimes known as small disjuncts [11], within a group of the other.

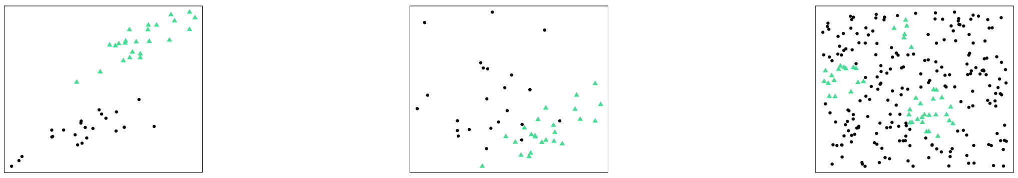
Figure VI.1.: Different situations relating to class complexity. The graph on the left shows separable classes, the middle one is an example where classes are not separable and the one on the right shows separable classes with complex boundaries (small disjuncts).

An additional hindrance that frequently occurs in data mining scenarios is associated to the representation of instances, to the features themselves [12]. These can be typically seen as observed outcomes of underlying factors that cause them and, as a result, are not always the ideal representation of the data. This can depend on the objective task and the learning method to be used. For example, a pixel-based representation for images can be ideal for a convolutional neural network to perform classification, but may be difficult for a lazy learning method to process.

When data have some kind of complexity, it can affect the performance of machine learning methods and these are usually not able to overcome the issue by themselves. Instead, a preprocessing step can transform data aiming to find a better representation which makes it easier to categorize points. Operating with features for this purpose is a task known as feature extraction or feature learning. There exists a wide variety of approaches to feature extraction [13], including linear transformations, manifold learning and neural network-based models. However, very few of them take class complexity into account and, as a result, extracting quality features with a specific strategy to reduce this complexity is still an important challenge.

In particular, autoencoders (AEs) [14] are neural networks specifically designed to extract features from the data. These are typically trained to reconstruct the input at its output, feeding the data through several layers which impose some kind of restriction or bottleneck in order to find more appropriate representations along the way. AEs can also be easily restricted or adapted in order to promote certain kinds of transformations and encodings, for example, finding sparse variables which only take high values for a small number of instances [15].

This work makes use of the well-known technique for regularizing the behavior of an AE, applied in this case to achieve class complexity reduction. Aiming to transform features onto a more useful space with special attention to class complexity, three concrete models that use different criteria are proposed. The bases for these are: Fisher's discriminant ratio, the Kullback-Leibler divergence (KLD) and least-squares support vector machines (LSSVMs). The new models have been tested against well-established feature extraction methods within a binary classification pipeline.

In summary, the main contributions of this paper are the following:

- New AE-based models able to learn from input features as well as binary class labels, specifically the following three variants:
  - Scorer, a model which enables separability among classes by means of the Fisher's discriminant ratio.
  - Skaler, a model that receives feedback from the KLD and can thus provide features where positive and negative instances belong to very different distributions.
  - Slicer, an extended AE using a LSSVM in order to simultaneously evaluate a simple linear classifier and assess the adequacy of the new features for classification.
- A thorough experimentation across 27 cases and 11 evaluation measures, focusing on different complexity rates and

classification performance, and against 4 other well-known feature extraction methods.

- A comparison between the most interpretable complexity metrics and several evaluation metrics for classifiers, revealing which of the complexity metrics are better predictors of classification performance.

As an important conclusion after the experimental analysis of the newly proposed models, we must point out that they can be trained to generate better features for the purposes of classification than other popular feature extraction methods.

The rest of this paper is organized as follows. Section VI.2 describes the current state with respect to available complexity measures and techniques to overcome complexity in data. Next, Section VI.3 introduces our proposals and provides all the details about their inner workings. Section VI.4 explains the details of the experimentation process, while Section VI.5 discusses the results. Lastly, conclusions and final comments are provided in Section VI.6.

VI.2. State of the art in complexity reduction

A dataset can present many different problems that may drive it to be considered difficult or complex to classify. Initially, a possible measure of complexity could be the error rate of the classifier itself. However, the objective of this work is identifying difficult datasets to treat them before learning a classifier. For this reason, we rely on other metrics which aim to characterize the complexity of supervised problems.

Sources of difficulty

Ho and Basu [16] identify three possible sources of difficulty: (1) class ambiguity, (2) boundary complexity and (3) sample sparsity and feature space dimensionality. The first applies to the circumstances where classes cannot be distinguished using the given features, either because they provide insufficient knowledge about the problem or because the classes are not well defined. The second source refers to the situation where classes are interleaved or not easily separable. In these cases, the complexity can be measured attending to class overlap and class separability as well as geometry, topology and density of manifolds. The last category covers issues with the structure of the sample, whether it is complete enough and the amount of variables the classifier needs to work with.

Complexity measures

In order to quantitatively assess how complex a dataset is, a wide variety of complexity metrics have been proposed over the years [4, 10]. The following sections briefly describe the most relevant approaches to measure complexity, paying special attention to the metrics that will be applied throughout the experimentation. Each metric is abbreviated according to the original nomenclature in [4] and [10].

Class overlap

Geometrical complexity in a dataset can be characterized in several ways. One approach is to measure the overlap in feature values among different classes. Each feature can be assessed as to how much it contributes to distinguishing the classes. In this case, measures usually focus on binary problems. The following measures follow this approach:

Maximum Fisher's discriminant ratio (F1) Fisher's discriminant ratio is a measure of class overlap, based on the simplest statistics for a distribution, mean and standard deviation. Higher values of this metric mean lower levels of overlap. The maximum over all features is taken as a measure of the class separation in a dataset.

Maximum feature efficiency (F3) Feature efficiency is calculated as the proportion of examples that can be unambiguously classified by a simple threshold, that is, they lie outside an overlapping region. This rate gives an idea of the usefulness of a given feature when attempting to classify every instance in the dataset. The maximum of this ratio across all features is known as F3.

Class separability and nonlinearity

Instead of measuring the importance of the overlapping regions in features, an alternative approach is to look for complexity of the boundary separating classes, that is, its ability to actually isolate both classes and its nonlinearity. Several measures have been developed regarding the shape and separation degree of classes.

Linear classifier error (L2) Linear separability of classes is the core of a branch of classification methods, support vector machines (SVM) [17]. In its simplest form, a SVM is a binary classifier that attempts to find the hyperplane which best separates both classes. Its training error can be used as a metric to characterize the separability (or lack thereof) of a dataset.

Linear classifier nonlinearity (L3) Describing the shape of the regions occupied by each class can also contribute to learning about the complexity of the data. In particular, this measure tackles nonlinearity, i.e. the smoothness of the decision boundary of a classifier, which can be detected by interpolating pairs of points of the same class to extract a test set and computing the classification error for this new set.

Neighborhoods and morphology

The previous traditional measures for data complexity come from a statistical or geometrical point of view. Other metrics look at how instances are located around each other, so they study local behavior instead of global properties.

1-NN classifier error (N3) Similarly to the L1-L3 measures, which make use of a simple classifier in order to measure complexity, this metric performs a leave-one-out validation of a nearest neighbor classifier, that is, it checks the class of every instance according to the nearest one, and measures complexity as the error rate obtained.

Recently, some new metrics have been proposed that attempt to describe data complexity from the perspective of data morphology [18]. These are based on the Pure Class Cover Catch Digraph (P-CCCD) classification method [19].

P-CCCD creates a collection of balls that cover the feature space so that each ball only contains points from the same class. The process consists in choosing a ball so that it is centered in a point of the target class and is the largest possible ball that does not any point of the other class. This is repeated until all points of that class are in at least one ball, producing a cover which is not necessarily optimal but is a good approximation.

Morphology-based complexity metrics are inspired by this algorithm in the sense that they look for a ball cover of all points where balls only contain points from one class, and then perform some computations according to the number of balls created. The main metrics are as follows:

Total number of balls (ONB_{tot}) This measure counts the total number of balls required to produce the cover. If b^+ balls are needed to cover all positive instances and b^- are necessary for the negative points, it is calculated as

$$ONB_{\text{tot}} = \frac{b^+ + b^-}{n} \,. \tag{VI.1}$$

Average number of balls (ONB_{avg}) It averages the amount of balls used to cover the points of each class. In a binary classification environment, the definition would just be the sum of the balls-to-points ratios, divided by 2:

$$ONB_{\text{tot}} = \frac{\frac{b^+}{n^+} + \frac{b^-}{n^-}}{2} \,. \tag{VI.2}$$

These metrics can turn into a very general way of describing the geometrical complexity of the classes, since the shape of the balls depends on the distance chosen (e.g. Euclidean, Manhattan or the maximum distance). The mechanism for covering the feature space attempts to use as few balls as possible to cover all points. If the dataset can be covered by a few large balls, then its complexity will be low, but if many small balls are needed, it means that many little clusters of different classes are near each other, and the complexity is thus high. Both ONB_{tot} and ONB_{avg} are, as a result, higher the more complex the data is. The difference between them is that ONB_{avg} gives the same weight to all classes, while ONB_{tot} does not distinguish classes but gives the same weight to all instances.

Feature space dimensionality

One of the main issues that occur in many datasets and has been tackled from many perspectives is dimensionality. Dimensionality refers to the number of variables where each instance takes values. High dimensionality has long been considered a problem for classification algorithms, known as curse of dimensionality [20]. It is not directly related to the way classes interact with each other, but a high number of features can hinder the performance of a classifier with a dataset that is otherwise not considered complex, due to the fact that most distance metrics lose meaning when measuring across many variables.

Dimension can be measured in absolute terms, but the complexity that derives from it is also related to the number of instances in the dataset. Two problems with the same number of features are not equally complex if the first one has 10 times more instances than the other. As a result, an instances-to-features rate (T2) can be considered a complexity metric that can give a better account of this relation.

Other models for complexity

When trying to reduce the complexity present in a dataset, one can take complexity measures into account for evaluation purposes, and use other ways of modeling complexity when training and performing data transformations. For instance, considering that each class presents different distributions across each variable, some similarity or dissimilarity metrics for distributions could be used.

The Kullback-Leibler divergence is a well-known measure of how a distribution differs from another one, it is asymmetric as it usually compares a distribution coming from data with a distribution representing a model or theory. If these are defined on a discrete probability space $\mathcal{X}$, then the divergence is formulated as

$$D_{\mathrm{KL}}(p\|q) = \sum_{x\in\mathcal{X}} p(x) \log \frac{p(x)}{q(x)} \,. \tag{VI.3}$$

This quantity could provide an intuition on how two distributions are overlapping or separated. It is higher the more different the distributions are. One way to retrieve a symmetric value out of it is to add the Kullback-Leibler divergence of the distributions in reverse order: $D_{\mathrm{KL}}(p\|q) + D_{\mathrm{KL}}(q\|p)$. For both measures, the chosen distribution when applying them to class separability could be a Bernoulli distribution for each feature in the encoding, so that their values are considered either high or low. If we model all features at the same time, a categorical distribution could be employed. Assuming a binary classification problem, we could measure the dissimilarity of the distribution corresponding to positive instances against the distribution of negative instances, which would provide a sense on how easy it is to differentiate them.

Reducing complexity in datasets

There are several approaches to complexity reduction in datasets. This section provides a general overview of the different aspects that can be treated and techniques for doing so.

Dimensionality of data has been one of the most diversely tackled issues. A multitude of methods exist in the literature, ranging from simple feature selection to nonlinear feature learning. A thorough review of all these can be found in [5], but we enumerate and describe the main ones below.

However, there are other emerging methods that may be able to modify other aspects of the data and reduce complexity along the way. Some of those are distance metric learning methods.

Feature selection

Assuming that not every variable has the same relevance for the purposes of classification, an initial approach to dimensionality reduction can be to simply discard some of them, retaining only the ones that help the classifier the most. This process is known as feature selection [5]. Of course, there exist a plethora of criteria that can apply for this purpose.

Filters This variety of techniques is mostly founded on statistical and information theory measures, such as the joint mutual information, the conditional mutual information, the Kullback-Leibler divergence or minimum-reduncancy-maximum-relevance. The objective is to quantify the utility of each variable and keep only the most useful ones. In fact, some approaches take class separability into account as well [21, 22].

Wrappers Another way of looking at feature selection is shaping it as an optimization problem, finding an adequate fitness function, typically the performance of a classifier, and making use of one of the many existing metaheuristics available, for instance, genetic algorithms, simulated annealing or particle swarm optimization, to name a few.

Embedded methods Some classifiers have built-in feature selection, so that they only look at the information provided by the most relevant variables. These are usually decision trees like C4.5 [23].

Linear feature extraction

Another way of reducing the number of variables is to attempt to summarize most of the information of the original variables in a smaller set of new variables, which emerge as linear transformations of the original ones.

Principal component analysis (PCA) [24, 25] PCA is a well-studied technique that solves the problem of obtaining features which retain the maximum possible variance while being uncorrelated to each other. It also allows to recover the original data from the projected points while losing the minimum amount of information as measured by the mean squared error.

Linear discriminant analysis (LDA) [26] This is a supervised method able to extract linear combinations of features which achieve good class separation. Under assumptions of normality, independence and homoscedascity, it can project the data onto a space consisting of new coordinates that best discriminate the classes. Its main drawback is that the number of resulting variables is completely determined by the number of classes. More specifically, for a problem with c classes, LDA will output a space of $c - 1$ linear combinations of the original variables. There is a recent generalization of LDA which claims to solve its stability issues and achieve better class separation through a maximum margin criterion [27].

Factor analysis [28] This technique assumes, unlike PCA, that a series of hidden factors are generating the observed data by means of linear combinations. The number of underlying factors is lower than the number of observed variables, and they are assumed to have zero mean and unit covariance (i.e. the identity matrix).

Nonlinear feature extraction

The most advanced methods for dimensionality reduction base their new variables on nonlinear transformations of the original ones.

A lot of these techniques can be grouped in a concept known as manifold learning, since they attempt to find structure for a manifold where most of the data lie, and thus transform each data point onto its coordinates on that manifold.

Multidimensional scaling (MDS) [29] This is a classical methodology that has served as basis for several other algorithms as well. Its objective is to compute new coordinates for data points while preserving distances among them as faithfully as possible. Instead of having points as inputs, it only receives the pairwise distances themselves, and minimizes a loss function which helps the model obtain coordinates for each point, creating a space where the given distances are maintained.

Isomap [30] This method extends metric MDS in order to find coordinates that describe the actual degrees of freedom of the data while preserving distances among neighbors and geodesic distances between the rest of points. Isomap constructs a neighborhood graph where each edge is weighted according to the Euclidean distance among vertices, then uses this to compute geodesic distances instead of using straight lines. These new distances are potentially higher than the Euclidean but help capture more information about the manifold.

Locally linear embedding (LLE) [31] The objective of LLE is similar to that of the previous techniques, but with a different approach to preserving the local structure. It finds a linear combination which describes each point from its neighbors. Once its coefficients have been computed, LLE optimizes the coordinates for a lower-dimensional space so that they fit the same expressions.

t-stochastic neighbor embedding (t-SNE) [32] This is a technique specially oriented for visualization, so it finds specially attractive low-dimensional projections of the data. It consists on assigning, for each pair of points, the probability that one point would choose the other as its nearest neighbor if neighbors are computed according to Gaussian distributions centered on each point. t-SNE then defines a low-dimensional mapping that tries to preserve these probability scores.

Autoencoder networks (AE) [14] AEs are neural network models which reconstruct the input at their output, using some kind of bottleneck in between so as to learn useful information from the data. We explain AEs in further detail in Section VI.3.

Distance metric learning

Distance metric learning [33] is an area of machine learning dedicated to learning distances from datasets. These distances are built to better represent the similarities and differences among examples than standard distances, such as the Euclidean distance.

In a supervised learning context, the problem of learning a distance can be formulated as follows:

$$\operatorname*{argmin}_{d \in \mathcal{D}} l(d, S, D), \text{ where} \tag{VI.4}$$

$$S = \{(x_i, x_j) \in \mathcal{X} \times \mathcal{X} \ : \ y_i = y_j\} \tag{VI.5}$$

$$D = \{(x_i, x_j) \in \mathcal{X} \times \mathcal{X} \ : \ y_i \neq y_j\} \tag{VI.6}$$

and l is a loss function that determines the fitness of a distance to describe the similarities and differences provided as sets S and D.

Some of the dimensionality reduction methods mentioned above can be also seen as distance metric learning techniques, but there are more algorithms which can learn distances. Some of them are addressed at improving the performance of k-nearest neighbors, and others are based on information theory. Among the most relevant are: NCA [34], LMNN [35], NCMML [36] and NCMC [36].

Current limitations

Many of the complexity reduction methods explained above tackle complexity only partially or from a limited perspective. In most of the cases, tha main focus is reducing dimensionality. This can help model data when good quality coordinates are found, but may discard useful information present in the class labels.

Some of the available methods consider class separability when selecting features [21, 22] but they do not generate new features, and can capture only a partial view of the whole feature space as a result.

In summary, there is an unexplored possibility of advanced feature extraction techniques which perform nonlinear transformations of variables in order to find spaces where classes are further away and easier to identify.

VI.3. Autoencoders for complexity reduction

The objective of this work is to develop strategies that address data complexity in a more complete way, that is, working with transformations of all available features and incorporating class complexity measures to acquire information from the class labels. The result is a collection of models that are based on AEs because they are very versatile deep learning architectures, able to transform the data in diverse ways according to their loss function. Our hypothesis is that when the loss function takes data complexity into account, then the AE will have more information to work with and will generate better features than other feature extraction methods.

In order to provide an indication on data complexity to a loss function, it is necessary to look for computationally simple ways of calculating a penalty that points the training method in the right direction. This problem can be approached from several possible perspectives, including the integration of a complexity metric, a measure of distribution dissimilarity or a linear separation method.

This section details the theoretical underpinings of our proposals: Scorer, Skaler and Slicer. First, some basic notions of AEs help establish a starting point for these new models. Next, the added penalties for Scorer and Skaler are explained. Lastly, the necessary modifications and computations needed for Slicer are shown as well.

Autoencoder fundamentals

A neural AE [14, 37] is generally a symmetrical neural network trained to reconstruct the inputs at its output. The composition of layers up to the middle one computes a new representation of the input data where some traits may be induced: lower dimension, sparsity, or robustness against noise, for example. The feature transformation is learned by means of a training process that optimizes the reconstruction error as well as, potentially, other penalties allowing the introduction of those specific aspects.

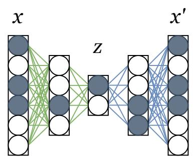

Figure VI.2.: The essential structure of an AE implemented as a fully connected feed-forward neural network, composed of an encoder f and a decoder g. The training loss of this model is measured as the distance d between the input x and its reconstruction $x' = (g \circ f)(x)$.

A simple AE models the reconstruction problem as a deterministic function given by the composition of an encoder f and a decoder g. When an instance is feeded to the model, the encoder transforms it to a vector located within the encoding space, and the decoder maps this vector to the original feature space.

Consider the diagram in Fig. VI.2. During training, mini-batches of samples are propagated through the network. The AE is evaluated according to the average distance between original and reconstructed samples. Its weights are iteratively modified in order to minimize this distance. There are two typical dissimilarity metrics for this purpose:

- Mean squared error: defined as the average of squared errors. If x and x' are a training sample and its reconstruction, it is expressed as:

$$\mathcal{L}(x, x') = \frac{1}{n}\sum_{i=1}^{n}(x_i - x'_i)^2 \qquad \text{(VI.7)}$$

- Cross entropy: this measure is effective when modeling data where values lie in the $[0, 1]$ interval, since it is usually implemented as the cross entropy of two Bernoulli distributions. The formulation is as follows:

$$\mathscr{L}(x, x') = -\sum_{i=1}^{n} x_i \log x'_i + (1 - x_i) \log(1 - x'_i) \tag{VI.8}$$

In general, any kind of measure that indicates the difference among two data points of the same type can be used. For certain types of structured data, such as images or sequences, specific reconstruction errors may also apply. For instance, a perceptual loss [38] can be very fitting for image reconstruction, since it focuses more in the appearance of the image instead of trying to accurately recover each individual pixel, which can lead to softer and blurrier images.

Once one of these dissimilarity metrics is chosen, the loss function of the AE can be defined:

$$J(\theta; S) = \sum_{(x,y)\in S} \mathscr{L}(x, (g \circ f)(x)) \,, \tag{VI.9}$$

where θ holds the parameters of the network, and thus determines f and g, and S is a set of training instances.

Diverse kinds of regularizations can be applied to the loss function with the objective of adjusting the behavior of the AE, such as sparsity, contraction or variational inference. Each of these result in a slightly different AE variant with its own applications. Although these and several other regularizations help build better feature spaces, to the best of our knowledge there is no AE variant focusing on enabling class separability or reducing data complexity yet.

AEs are generally trained with common neural network optimizers, such as stochastic gradient descent [39] or Adam [40]. They decide how to update the parameters in an iterative process which computes the gradient of the loss function via backpropagation [41].

Regularizing autoencoders with label information

As described above, a basic autoencoder is trained using a loss function which evaluates the distance between the input feature vector and its reconstruction through the network. It is clear, by its definition, that instance labels are not used at all to compute the loss function, nor does the AE receive this information as input. This has its advantages and shortcomings. A benefit is that one may train AEs using unlabeled data and obtain valuable knowledge as a result. This allows for their use in several widespread applications

[42], such as anomaly detection, data denoising, synthetic instance generation and semantic hashing.

One possible drawback, when applying AEs to classification problems, is that they will extract features that may or may not help distinguish the classes, since they are not provided with the labels. However, this is not applicable to all AE-based models, since some of them can take the class label into account when computing the encoded representation, either directly as an input layer to the network, or indirectly, by informing the loss function.

A common way to modify the behavior of the training process and improve the solutions is to add a penalty term Ω to the loss function promoting certain aspects of the encoding or reconstruction mappings. This penalty may be dependent on the weights of the network or the resulting codes. It is added with a weight coefficient λ in order to adjust its importance with respect to the standard reconstruction error:

$$J(\theta; S) = \sum_{(x,y)\in S} \mathcal{L}(x, (g \circ f)(x)) + \lambda\Omega(\theta; S) \tag{VI.10}$$

For instance, one well-known regularization consists in penalizing high levels of simultaneous activations within the codes. This, usually called a sparsity regularization [15], helps maintain a low number of active neurons in the encoding for each sample.

In our case, the objective is that the resulting feature transformation helps better separate different classes, so the loss function should receive some kind of label information in order to be able to learn from it. The procedure can thus be similar to a penalty modification, but using the class label within the penalty term Ω.

There could be numerous ways of analyzing the relation of codes and classes. For example, trying to optimize a complexity measure or maximizing the difference among class distributions, as well as wrapping a simple classifier so as to assess the quality of the features. These are the main ideas behind our three proposals:

- Scorer, an AE model with a Fisher's discriminant ratio-based penalty. Its objectives may be collaborating or in opposition, but it needs to find a balance between good instance reconstructions and low class overlap.
- Skaler, an AE using the KLD to separate class distributions. The encodings are modeled as a categorical distribution and the model attempts to maximize the divergence among the distribution of positive instances and that of negative instances. This should draw them apart from each other.

- Slicer, an AE which internally trains a linear least-squares support vector machine. The internal classifier need not be perfect, but it helps the model analyze how easy it is to classify the instances using the generated features. The objective, in this case, is to maximize the linear separation of both classes.

Along the rest of this section, each one of these AE-based models is thoroughly described.

Scorer

The first of our approaches to complexity reduction is to directly employ one of the complexity metrics as penalty, assuming that, if an AE is able to optimize this metric for a given dataset, the resulting representation will be less complex than the original. For this purpose, Fisher's discriminant ratio has been selected, as it is simple enough to be computed on the fly during training. The result is an AE which performs supervised class overlap reduction, or Scorer for short.

In order to introduce a complexity penalty based on Fisher's discriminant ratio, we consider the average of the discriminant ratios of each feature. This is different to the complexity measure commonly known as maximum Fisher's discriminant ratio or F1, which instead calculates the maximum of those ratios. In this case, we chose the average because it should provide better gradients in order to optimize the objective. This was corroborated by a preliminary experimentation.

The following equations formally define the complexity penalty computed within Scorer, N^+ denoting the amount of positive examples and N^- the number of negative ones. First, we define the necessary terms for the mean of each variable for positive instances and the same for negative instances.

$$\mu_j^+ = \frac{1}{N^+} \sum_{(x,+1)\in S} f(x)_j, \quad \mu_j^- = \frac{1}{N^-} \sum_{(x,-1)\in S} f(x)_j, \tag{VI.11}$$

Next comes the standard deviation for each variable and for each class, calculated as the mean squared value minus the square of the mean:

$$\sigma_j^+ = \left[\frac{1}{N^+} \sum_{(x,+1)\in S} f(x)_j^2\right] - (\mu_j^+)^2 \tag{VI.12}$$

$$\sigma_j^+ = \left[\frac{1}{N^-} \sum_{(x,-1)\in S} f(x)_j^2\right] - (\mu_j^-)^2 \tag{VI.13}$$

This allows to put together an expression for the average Fisher's discriminant ratio, which is introduced in the loss function in a way that ensures its value to be between 0 and 1:

$$F = \frac{1}{n_f} \sum_{j=1}^{n_f} \frac{(\mu_j^+ - \mu_j^-)^2}{\sigma_j^+ + \sigma_j^-}, \quad \Omega(\theta; S) = \frac{1}{1+F} \tag{VI.14}$$

The penalty term, drawing inputs from the encoding layer and the class label, is simply computed at the end of each training and added to the loss function, multiplied by a weight hyperparameter λ so as to balance it with the reconstruction objective. Figure VI.3 extends the basic AE diagram with these new components in order to illustrate how Scorer differs from the basic model.

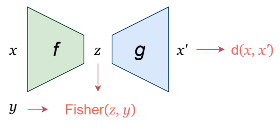

Figure VI.3.: Schematic illustration of the Scorer model. The average Fisher discriminant ratio of the encoded class distributions contributes to the training loss.

Skaler

The next step in inducing a class-separating behavior in an AE is to use information theory-based measures. In this case, the AE is not forced to directly optimize a complexity metric. Instead, it receives information about the current relation among class distributions, and is assessed according to the similarity of those.

Although cross entropy is the conventional measure for classification loss, we refrain from using it as a penalty because the objective is not to directly classify, thus concentrating all instances on one of two points, but to provide a representation that better clusters examples.

Skaler is a supervised feature extraction model with a KLD-based penalty for class separation. As explained above, the KLD gives an asymmetric view on how two distributions are different. In this case, the objective is to maximize the difference among the distribution of encodings belonging to the positive class and those belonging to the negative class. A schematic view is provided in Figure VI.4.

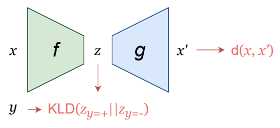

Figure VI.4.: Schematic illustration of the Skaler model. The KLD between the positive and the negative-class encodings contributes negatively to the training loss.

Both positive and negative encodings can be modeled as following categorical distributions, if we assume that each feature in the encoding can have a high (1) and a low (0) state and that the highest feature is the one that matters. This is a simplification but it helps build a KLD-based formulation that is easy to implement and able to train successfully. Then, the sample space of each distribution consists of events associated to each feature, indicating whether that feature is the highest. If $P_+(j)$ denotes the probability that the j-th feature is highest for positive instances and $P_-(j)$ does the same for the negative class, the KLD-based penalty function would be as follows:

$$\Omega(\theta; S) = -\sum_{j=1}^{d} P_+(j) \log \frac{P_+(j)}{P_-(j)} \tag{VI.15}$$

Now, in order to compute valid probabilities for each case of the categorical distribution out of the encoding generated by the AE, we take the mean of each variable in a vector and perform the softmax activation function, obtaining a vector of values summing 1, thus representing a distribution. The j-th component of that vector corresponds to the probability that the j-th feature is high for any given data sample:

$$P_+(j) = \text{softmax}\left(\frac{1}{N^+} \sum_{(x,+1)\in S} f(x)\right)_j \tag{VI.16}$$

$$P_-(j) = \text{softmax}\left(\frac{1}{N^-} \sum_{(x,-1)\in S} f(x)\right)_j \tag{VI.17}$$

Some preliminary tests revealed that it is easy for this penalty to force encodings onto a single class-dependent value for any inputs. It was observed, however, that maximizing the entropy of the encoding variables helped prevent this issue, so it is added as a negative term to the penalty in the implementation.

Slicer

The third proposal of this work goes a bit further than the two previous ones, since it incorporates not only a different penalty function, but also additional learnable parameters.

This alternative regularization is inspired on least-squares support vector machines (LSSVM) [43]. These models attempt to learn the hyperplane which best separates both classes, but the difference between them and traditional SVMs lies on the objective function. For both models, the linear (non-kernelized) version of the classifier optimizes parameters w and b of the hyperplane $w^T x + b$. However, the functions that both models minimize are different. In the case of the LSSVM, the parameters are fitted to optimize the following expression:

$$\frac{1}{2}\|w\|^2 + \frac{\beta}{2}\sum_{(x,y)\in S}\left(1 - y\left(w^T x + b\right)\right)^2 \tag{VI.18}$$

The idea behind our model is to find a representation which facilitates the task of fitting a linear classifier. The resulting model is an AE for supervised linear classifier error reduction, hereinafter called Slicer.

In order for the model to compute the linear classifier objective function, we add trainable weights w and b to the computation graph of the neural network. These are used to get the output of a linear SVM, allowing thus to train the SVM and use it as a penalty to modify the behavior of the encoder at the same time:

$$\Omega(\theta; S) = \frac{1}{2}\|w\|^2 + \frac{\beta}{2}\sum_{(x,y)\in S}\left(1 - y\left(w^T f(x) + b\right)\right)^2 , \tag{VI.19}$$

where β is just a hyperparameter weighting the importance of the LSSVM loss, and w and b are updated by the model after each epoch, just like the rest of neural network weights. In this case, the value of y is 1 for the positive class and -1 for the negative one.

Figure VI.5 illustrates how the AE is modified using the LSSVM objective function, taking the encoding $z = f(x)$ as input for the LSSVM and using a prediction $p = w^T z + b$ to calculate its loss.

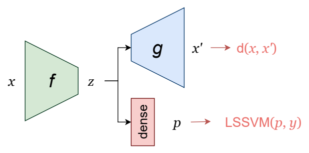

Figure VI.5.: Schematic illustration of the Slicer model.

The result is a model that simultaneously trains a very simple classifier on the encoded data and uses its objective in order to find better representations. Our assumption is that there will exist some level of collaboration between both models and this will help the new features become more practical for classifiers to use.

VI.4. Experimental setup

Our proposals have been tested to verify their performance in reducing the complexity of data according to some measures, as well as in generating feature spaces where binary classification is easier. This section first goes through the materials for the experiments: data, methods and metrics, and then provides details on the implementations of the newly proposed models.

The experiments that were performed in order to analyze whether using class information in an AE provides an advantage include a broad range of datasets as well as several well-established methods for comparison purposes. The following sections explain the data, compared methods and evaluation metrics used along the experimentation.

Data

The methods have worked with a collection of 27 datasets from several sources with varying dimensionalities. Thirteen of them originally have binary classes, six derive from the individual labels in a multilabel dataset, five are "grouped" binarizations where several classes are taken as the positive class and the rest as the negative one, and two originate from one-vs-all scenarios where only one arbitrary class is chosen as the positive one.

The grouped binarizations have been chosen so as to present a binary scenario that would make sense with each of the problems posed by the datasets: distinguishing vowels from consonants (when the original label was the letter), odd from even handwritten digits, walking from staying movement signals and two high-level categories of soil from images. One-vs-all schemes have not been performed in these cases due to the high amount of classes and resulting experiments.

The datasets are briefly described and referenced in the supplementary material.

Complexity reduction methods

In addition to our proposals Scorer, Skaler and Slicer, we have selected four dimension reduction methods which can contribute to reducing the complexity of these datasets: PCA, LLE, Isomap and basic AE. PCA is used as the baseline for dimension reduction, LLE and Isomap are selected due to their manifold learning purpose, and the basic AE serves to analyze whether our proposals improve its behavior. None of these has the capability of learning from the classes, but they do address the dimensionality problem. The objective of our experiments is, thus, to test whether class information can be useful for an automatic feature learner to retrieve better quality attributes. Please refer to Table VI.1 and Section VI.2 for a brief description of the idea behind each technique and a longer explanation, respectively.

Method	Description
PCA[25]	Linear variance maximization
LLE[31]	Neighborhood-based manifold learning
Isomap[30]	MDS-based manifold learning
Basic AE[44]	Neural network for data reconstruction
Skaler	Proposed AE with Kullback-Leibler-based penalty
Scorer	Proposed AE with discriminant ratio-based penalty
Slicer	Proposed AE with LSSVM-based penalty

Table VI.1.: Brief description of each method available in the experiments

Evaluation metrics

In order to provide different perspectives on the performance of all methods, a diverse set of evaluation metrics has been selected. The objective is to be able to analyze the possible advantages and shortcomings of each available method.

We have trained our proposals and the compared methods to reduce the dimension of the datasets to the square root of the original dimension. The feature transformation learned by each one has been used to compute a list of complexity metrics for the resulting projections. Afterwards, we have trained several simple classifiers in order to assess the ease of classification with the generated features.

In summary, the metrics used for evaluation of each complexity reduction method can be categorized into classifier-agnostic and classifier-dependent.

Classifier-agnostic metrics are some of the complexity measures discussed in Section VI.2. They have been chosen essentially for their popularity and easy interpretation. Morphology-based ONB metrics have also been computed so as to verify their affinity with the actual classification performance as well.

On the other hand, a partial objective of this experimentation is to check whether the generation of new features can actually ease classification tasks if aided by class information. The logical step is thus to analyze the performance of several datasets with the resulting variables.

- F-score. Derived from precision (the ratio of instances correctly predicted as positive) and recall (the ratio of positive instances correctly detected), it is essentially the harmonic mean of both:

$$\text{F-score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{VI.20}$$

- Area under the ROC curve (AUC). This metric is computed as the area, out of 1, that lays under the receiver operating characteristic curve, which denotes, as the prediction threshold goes from 0 to 1, the ratio of true positives related to the ratio of false positives.

- Cohen's Kappa. It measures the level of agreement between the predictor and the ground truth, that is, the extent to which the coincidences differ from random chance (p_c). The mathematical definition is:

$$\kappa = 1 - \frac{1 - \text{Acc}}{1 - p_c} \, . \qquad \text{(VI.21)}$$

These evaluation metrics have been chosen over other popular ones such as accuracy or precision since they attempt to provide a better overall picture of the performance without being affected by imbalance. Some other metrics that are also considered common complexity measures are not actually classifier-independent: linear classifier error (L2), nonlinearity of linear classifier (L3) and 1NN classifier error (N3). These were previously defined in Section VI.2.

Table VI.2 gathers all of these selected metrics with a short interpretation of each one. In order to obtain robust values, they have been computed over a 5-fold cross validation scheme.

	Metric	Meaning
Agnostic	F1 (Fisher)	Class overlap according to mean and variance
	F3 (efficiency)	Feature ability to separate classes
	ONB (total)	Total number of balls in cover
	ONB (average)	Average number of balls in cover
Dependent	L2	Linear classifier error
	L3	Linear classifier nonlinearity
	N3	1NN classifier error
	F-score	Tradeoff between precision and recall
	AUC	Area under the ROC curve
	Kappa	Agreement between prediction and truth

Table VI.2.: Brief description of each evaluation metric used for the experiments, classified according to their dependency on a classifier

Parameters

The last details about the execution of the experiments are provided in Table VI.3, which shows all the values for the parameters involved in the each of the different methods.

Parameter	Value
Encoding dimension	$\max\left\{\min\left\{\sqrt{d}, \frac{n}{10}\right\}, 2\right\}$
Epochs	200
Number of hidden layers	3 (1 for $<$ 100 variables)
Activation function (AEs except Skaler)	ReLU
Activation function (Skaler)	Sigmoid
Penalty weight - Scorer	0.01
Penalty weight - Skaler	0.1
Penalty weight - Slicer	1
Reconstruction error	Cross entropy

Table VI.3.: Enumeration of parameters used throughout the experiments.

VI.5. Experimental results

This section presents the outcomes of the experiments performed, focusing on comparing the different methods, as well as drawing conclusions from the obtained metrics and graphics.

Results

Experiments for the 27 datasets have been conducted in a 5-fold cross validation scheme. A total of 16 metrics were computed for each case, and the full results are available at the associated website*. Next, we show and analyze aggregated results and the corresponding statistical tests.

Table VI.4 holds the average ranking that each dimensionality reduction method achieved for each metric throughout the dataset collection. The winning method for each row is marked in underlined bold text. The number of overall first positions in rankings is summed up and shown in the last row of the table. The first observation that can be extracted is that model Slicer turns out to be consistently superior in most metrics, resulting in a vastly higher amount of won cases than the rest.

Looking at the table into more detail, we can observe that the three supervised AE-based methods overall reach better metrics than the traditional feature extraction techniques, especially when analyzing the resulting predictive performance of the different classifiers.

In order to calculate which differences are significant, the Friedman's Aligned Ranks test was performed with its corresponding post-hoc test where the winning algorithm was chosen as the control for each metric. Table VI.4 organizes the results of these tests, indicating which methods were significantly worse than the

* https://ari-dasci.github.io/S-reducing-complexity/

		PCA	LLE	Isomap	AE	Skaler	Scorer	Slicer
	F1	⊗ 5.885	⊗ 6.115	⊗ 5.731	⊗ 4.038	★ **1.577**	2.308	2.346
	F3	⊗ 4.250	⊗ 5.231	⊗ 4.846	⊗ 5.654	2.788	3.000	★ **2.231**
	N3	⊗ 4.692	⊗ 5.173	⊗ 5.308	× 4.269	3.654	2.577	★ **2.327**
	L2	★ **2.712**	× 4.519	3.981	⊗ 5.654	× 4.442	3.846	2.846
	L3	2.769	× 4.500	4.115	⊗ 5.500	⊗ 4.923	3.654	★ **2.538**
	ONB_{tot}	⊗ 4.481	⊗ 6.115	⊗ 4.519	⊗ 4.481	3.442	3.019	★ **1.942**
	ONB_{avg}	⊗ 4.442	⊗ 6.077	⊗ 4.519	⊗ 4.577	× 3.538	2.904	★ **1.942**
kNN	F-score	3.096	⊗ 6.923	⊗ 4.904	4.115	3.519	3.231	★ **2.212**
	AUC	3.288	⊗ 7.000	⊗ 4.750	× 4.038	3.500	3.250	★ **2.173**
	Kappa	3.096	⊗ 7.000	⊗ 4.827	× 4.038	3.558	3.346	★ **2.135**
SVM	F-score	3.788	⊗ 6.846	⊗ 4.673	3.538	3.538	2.923	★ **2.692**
	AUC	× 4.000	⊗ 6.846	⊗ 4.865	3.462	3.346	2.962	★ **2.519**
	Kappa	3.904	⊗ 6.846	⊗ 4.827	3.346	3.577	2.962	★ **2.538**
MLP	F-score	⊗ 4.077	⊗ 6.769	⊗ 3.962	⊗ 4.731	2.596	⊗ 3.865	★ **2.000**
	AUC	⊗ 4.000	⊗ 7.000	⊗ 4.058	⊗ 4.635	2.404	⊗ 3.942	★ **1.962**
	Kappa	⊗ 4.000	⊗ 7.000	⊗ 4.077	⊗ 4.596	2.558	⊗ 3.827	★ **1.942**
	wins	62	10	17	19	78	64	**188**

Table VI.4.: Average ranking for each method in each evaluated metric. A horizontal line separates complexity metrics from classifier evaluation metrics. The best method is marked with a star ★. Those which are worse with $p < 0.05$ are marked with ×, and those which are worse with $p < 0.01$ are marked with ⊗. The total number of first positions achieved by each method is shown in the last row.

winner with two levels of confidence ($p < 0.05$ and $p < 0.01$). The tests allow to assess whether the values found by the rankings can be considered enough to state that two methods are performing differently. It is important to notice, however, that each statistical test had information only about a specific metric, and not the whole picture. As a result, the fact that a test does not find significant differences among two methods does not mean that they perform the same in general.

As a last summary, we have gathered all results and performed the Friedman's Aligned Ranks test in order to obtain potential global differences among methods. Although values from different metrics are mixed in these data, the test only considers rankings so it allows to extract some intuitions about the overall performance and whether different methods can be discerned from their evaluation metrics. This test is visualized in Figure VI.6, where critical distances are annotated with a confidence level of 99% ($p < 0.01$).

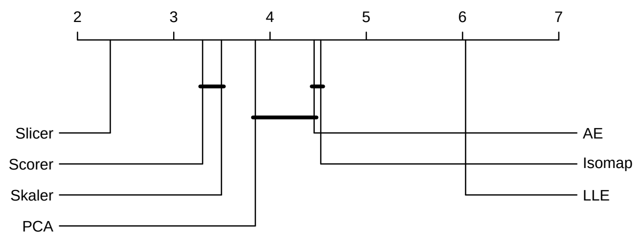

Figure VI.6.: Critical distance plot for all results. Horizontal lines join methods where significant differences were not detected.

The supplementary material for this paper also includes density estimation plots which provide a visual account of the overall results of each method for each metric.

Discussion

The previous results allow to notice some interesting details. One of them is the fact that the model that performed best according to the F1 metric, Skaler, is not the one that attempts to optimize that metric. This suggests that, within a neural network model, the KLD-based loss penalty is more useful for this purpose than the F1 metric itself. It is also noteworthy that Skaler, having reached the best separability metrics according to the F1 metric, ended up losing performance to Slicer when evaluated by means of actual classifiers. This could mean that the F1 metric, despite being widely known and used, is not the best estimator of classifier performance.

Looking further at the relation between the complexity metrics and the classification metrics for each classifier, it is straightforward to identify the complexity metrics that align best with classifier performance. Those would be F3, N3, L3 and the morphology-based metrics. However, the complexity metrics where rankings are most similar to the classification rankings seem to be ONB_{tot} and ONB_{avg}.

As to the comparison among complexity reduction methods, the computed rankings, critical distance plots and joyplots reveal that Slicer presents a clear advantage over the rest of methods in the majority of metrics. The individual statistical tests show many significant differences between Slicer and other methods, specifically LLE and Isomap. Some differences are also found from Slicer to PCA and AE, although not for every case. However, it is important to notice that Slicer is consistently superior to every other method across almost all metrics, something that these tests do not take into account. A more global perspective is given by the critical distance plot in Figure VI.6, which does find significant superiority of Slicer over the rest of methods.

In addition to Slicer taking the top place in classification tasks, we can also see that the other supervised AEs tend to be superior than the unsupervised methods in many cases, but the differences are smaller. In fact, PCA is also competing with them when the kNN classifier is employed for classification, which is interesting considering the simplicity of the method. The standard AE, however, does not achieve very good results in comparison to the improved versions with class-informed penalties. This leads to deduce that these types of regularizations are helping differentiate our proposals from what is otherwise the exact same neural network architecture. This idea is corroborated by the overall number of wins in Table VI.4, where both Skaler and Scorer achieve a higher number of wins than the rest except Slicer. Furthermore, the critical distance plot in Figure VI.6 shows a significant difference from

Scorer and Skaler to the classical feature extraction techniques, without being able to discard whether they perform equally.

In summary, the experimentation has shown that defining a class-based penalty in an otherwise unsupervised learning method such as an AE does help generate more useful features when the objective is training classifiers. Although the level of benefit will depend on the classifier used, any one of them will have some improvement in its performance. Of all the proposed approaches, the one that seems to retain the most information about classes within the encoded features is Slicer, which simultaneously learns the encoding and a linear classifier for the encoded variables.

VI.6. Final comments

This work has introduced the concept of class complexity onto AEs that learn from class labels. If dimension reduction helps classifiers by providing more compact versions of the data, complexity reduction goes further by also improving the shape and distribution of the different classes. This concept is realized in 3 specific models with distinct behavior, Scorer, Skaler and Slicer.

Similarly to other preprocessing methods, these intend to ease a later classification task. In this case, only binary targets have been supported, since most of the complexity metrics were initially defined for binary problems. It would be possible to extend the proposed framework to multiclass or even multi-label datasets, although the penalty functions as well as the implementations would need notable modifications. A similar approach could also be followed for regression problems, where the target variable could help model the embedding space.

An additional option that may be explored is the application of a combination of penalties, since they are not necessarily exclusive, including improvements on basic AEs such as variational AEs. Some preliminary tests suggested that combining Slicer and Scorer may provide a slight improvement in classification performance, but it would need meticulous optimization of the selected penalties and their weights in order to provide promising results. Further steps would include introducing other complexity measures into the penalty function, so as to tackle other aspects of data complexity, as well as learning from just a small number of labels in a semi-supervised scenario [45].

The experimentation developed to support the proposals has served not only to highlight the potential of Slicer as a better alternative for feature extraction, but also to notice which complexity measures are better predictors of classifier performance.

In particular, morphology-based metrics like ONB_{tot} and ONB_{avg} seemed to be more aligned with classifiers.

Acknowledgments

We thank José A. Fernández for suggesting the KLD as a possible separation measure. D. Charte is supported by the Spanish Ministry of Science under the FPU program (Ref. FPU17/04069). F. Charte is supported by the Spanish Ministry of Science project PID2019-107793GB-I00 / AEI / 10.13039/501100011033. F. Herrera is supported by the Spanish Ministry of Science project PID2020-119478GB-I00 and the Andalusian Excellence project P18-FR-4961. This work is supported by the project DeepSCOP Ayudas Fundación BBVA a Equipos de Investigación Científica en Big Data 2018.

References

[1] Hui Xiong et al. "Enhancing data analysis with noise removal". In: *IEEE Transactions on Knowledge and Data Engineering* 18.3 (2006), pp. 304–319 (cit. on p. 1).

[2] Charu C Aggarwal. "Outlier analysis". In: *Data mining*. Springer. 2015, pp. 237–263 (cit. on p. 1).

[3] Shaeela Ayesha, Muhammad Kashif Hanif, and Ramzan Talib. "Overview and comparative study of dimensionality reduction techniques for high dimensional data". In: *Information Fusion* 59 (2020), pp. 44–58. DOI: https://doi.org/10.1016/j.inffus.2020.01.005 (cit. on p. 1).

[4] Tin Kam Ho and Mitra Basu. "Complexity measures of supervised classification problems". In: *IEEE transactions on pattern analysis and machine intelligence* 24.3 (2002), pp. 289–300 (cit. on pp. 1, 2, 5).

[5] Salvador García, Julián Luengo, and Francisco Herrera. *Data preprocessing in data mining*. Vol. 72. Springer, 2015 (cit. on pp. 1, 9).

[6] Yiming Yang and Xin Liu. "A re-examination of text categorization methods". In: *SIGIR '99*. 1999 (cit. on p. 2).

[7] E. Dada et al. "Machine learning for email spam filtering: review, approaches and open research problems". In: *Heliyon* 5 (2019) (cit. on p. 2).

[8] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision* 115 (2015), pp. 211–252 (cit. on p. 2).

[9] Rahul C Deo. "Machine learning in medicine". In: *Circulation* 132.20 (2015), pp. 1920–1930 (cit. on p. 2).

[10] Ana C Lorena et al. "How complex is your classification problem? a survey on measuring classification complexity". In: *ACM Computing Surveys (CSUR)* 52.5 (2019), pp. 1–34 (cit. on pp. 2, 5).

[11] Gary M Weiss. “Learning with rare cases and small disjuncts”. In: *Machine Learning Proceedings 1995*. Elsevier, 1995, pp. 558–565 (cit. on p. 2).

[12] Laurens Van Der Maaten, Eric Postma, and Jaap Van den Herik. *Dimensionality reduction: a comparative review*. Tech. rep. 2009 (cit. on p. 2).

[13] Yoshua Bengio, Aaron Courville, and Pascal Vincent. “Representation learning: A review and new perspectives”. In: *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013), pp. 1798–1828 (cit. on p. 3).

[14] David Charte et al. “A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines”. In: *Information Fusion* 44 (2018), pp. 78–96. DOI: 10.1016/j.inffus.2017.12.007 (cit. on pp. 3, 11, 13).

[15] Andrew Ng. “Sparse autoencoder”. In: *CS294A Lecture notes* 72.2011 (2011), pp. 1–19 (cit. on pp. 3, 15).

[16] Mitra Basu and Tin Kam Ho. *Data complexity in pattern recognition*. Springer Science & Business Media, 2006 (cit. on p. 4).

[17] Ingo Steinwart and Andreas Christmann. *Support vector machines*. Springer Science & Business Media, 2008 (cit. on p. 6).

[18] José Daniel Pascual-Triana et al. “Revisiting Data Complexity Metrics Based on Morphology for Overlap and Imbalance: Snapshot, New Overlap Number of Balls Metrics and Singular Problems Prospect”. In: *Knowledge and Information Systems* (2021 (forthcoming)) (cit. on p. 6).

[19] Artür Manukyan and Elvan Ceyhan. “Classification of Imbalanced Data with a Geometric Digraph Family”. In: *Journal of Machine Learning Research* (2016) (cit. on p. 6).

[20] Charu C Aggarwal, Alexander Hinneburg, and Daniel A Keim. “On the surprising behavior of distance metrics in high dimensional space”. In: *International conference on database theory*. Springer. 2001, pp. 420–434 (cit. on p. 7).

[21] Yishi Zhang et al. “Divergence-based feature selection for separate classes”. In: *Neurocomputing* 101 (2013), pp. 32–42. DOI: https://doi.org/10.1016/j.neucom.2012.06.036 (cit. on pp. 9, 12).

[22] Lei Wang. “Feature selection with kernel class separability”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30.9 (2008), pp. 1534–1546 (cit. on pp. 9, 12).

[23] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers, 1993 (cit. on p. 9).

[24] Karl Pearson. “LIII. On lines and planes of closest fit to systems of points in space”. In: *Philosophical Magazine Series 6* 2.11 (Nov. 1901), pp. 559–572. DOI: 10.1080/14786440109462720 (cit. on p. 10).

[25] Ian T Jolliffe. “Introduction”. In: *Principal component analysis*. Springer, 1986, pp. 1–7. DOI: 10.1007/978-1-4757-1904-8 (cit. on pp. 10, 21).

[26] Ronald A Fisher. “The statistical utilization of multiple measurements”. In: *Annals of Human Genetics* 8.4 (1938), pp. 376–386. DOI: 10.1111/j.1469-1809.1938.tb02189.x (cit. on p. 10).

[27] Haifeng Li, Tao Jiang, and Keshu Zhang. “Efficient and robust feature extraction by maximum margin criterion”. In: *IEEE transactions on neural networks* 17.1 (2006), pp. 157–165 (cit. on p. 10).

[28] Ian T Jolliffe. “Principal Component Analysis and Factor Analysis”. In: *Principal component analysis*. Springer, 1986, pp. 115–128. DOI: 10.1007/978-1-4757-1904-8 (cit. on p. 10).

[29] Ingwer Borg and Patrick JF Groenen. *Modern multidimensional scaling: Theory and applications*. Springer Science & Business Media, 2005 (cit. on p. 10).

[30] Joshua B Tenenbaum, Vin De Silva, and John C Langford. "A global geometric framework for nonlinear dimensionality reduction". In: *science* 290.5500 (2000), pp. 2319–2323. DOI: 10.1126/science.290.5500.2319 (cit. on pp. 11, 21).

[31] Sam T Roweis and Lawrence K Saul. "Nonlinear dimensionality reduction by locally linear embedding". In: *science* 290.5500 (2000), pp. 2323–2326. DOI: 10.1126/science.290.5500.2323 (cit. on pp. 11, 21).

[32] Laurens Van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE." In: *Journal of machine learning research* 9.11 (2008) (cit. on p. 11).

[33] Juan Luis Suárez, Salvador García, and Francisco Herrera. "A tutorial on distance metric learning: Mathematical foundations, algorithms and software". In: *arXiv preprint arXiv:1812.05944* (2018) (cit. on p. 11).

[34] Jacob Goldberger et al. "Neighbourhood components analysis". In: *Advances in neural information processing systems* 17 (2004), pp. 513–520 (cit. on p. 12).

[35] Kilian Q Weinberger and Lawrence K Saul. "Distance metric learning for large margin nearest neighbor classification." In: *Journal of machine learning research* 10.2 (2009) (cit. on p. 12).

[36] T. Mensink et al. "Distance-Based Image Classification: Generalizing to New Classes at Near-Zero Cost". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.11 (2013), pp. 2624–2637. DOI: 10.1109/TPAMI.2013.83 (cit. on p. 12).

[37] Yoshua Bengio. "Deep learning of representations for unsupervised and transfer learning". In: *Proceedings of ICML workshop on unsupervised and transfer learning*. 2012, pp. 17–36 (cit. on p. 13).

[38] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. "Perceptual losses for real-time style transfer and super-resolution". In: *European conference on computer vision*. Springer. 2016, pp. 694–711 (cit. on p. 14).

[39] Herbert Robbins and Sutton Monro. "A stochastic approximation method". In: *The annals of mathematical statistics* (1951), pp. 400–407. DOI: 10.1007/978-1-4612-5110-1_9 (cit. on p. 14).

[40] Diederik Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *International Conference on Learning Representations*. 2015 (cit. on p. 14).

[41] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (1986), pp. 533–538. DOI: 10.1038/323533a0 (cit. on p. 14).

[42] David Charte et al. "An analysis on the use of autoencoders for representation learning: Fundamentals, learning task case studies, explainability and challenges". In: *Neurocomputing* 404 (2020), pp. 93–107. DOI: https://doi.org/10.1016/j.neucom.2020.04.057 (cit. on p. 15).

[43] Johan AK Suykens and Joos Vandewalle. "Least squares support vector machine classifiers". In: *Neural processing letters* 9.3 (1999), pp. 293–300 (cit. on p. 18).

[44] Dana H. Ballard. "Modular Learning in Neural Networks". In: *Proceedings of the Sixth National Conference on Artificial Intelligence - Volume 1*. AAAI'87. Seattle, Washington: AAAI Press, 1987, pp. 279–284 (cit. on p. 21).

[45] Diederik P Kingma et al. “Semi-supervised learning with deep generative models”. In: *Advances in neural information processing systems*. 2014, pp. 3581–3589 (cit. on p. 26).