Valls, J.M., Aler, R., Galván, I.M. et al. Supervised data transformation and dimensionality reduction with a 3-layer multi-layer perceptron for classification problems. J Ambient Intell Human Comput (2021).

DOI: https://doi.org/10.1007/s12652-020-02841-y

Noname manuscript No.
(will be inserted by the editor)

Supervised Data Transformation and Dimensionality Reduction with a 3-layer Multi-Layer Perceptron for Classification Problems

José M. Valls · Ricardo Aler · Inés M. Galván · David Camacho

Received: date / Accepted: date

Abstract The aim of data transformation is to transform the original feature space of data into another space with better properties. This is typically combined with dimensionality reduction, so that the dimensionality of the transformed space is smaller. A widely used method for data transformation and dimensionality reduction is Principal Component Analysis (PCA). PCA finds a subspace that explains most of the data variance. While the new PCA feature space has interesting properties, such as removing linear correlation, PCA is an unsupervised method. Therefore, there is no guarantee that the PCA feature space will be the most appropriate for supervised tasks, such as classification or regression. On the other hand, 3-layer Multi Layer Perceptrons (MLP), which are supervised methods, can also be understood as a data transformation carried out by the hidden layer, followed by a classification/regression operation performed by the output layer. Given that the hidden layer is obtained after a supervised training process, it can be considered that it is performing a supervised data transformation. And if the number of hidden neurons is smaller than the input, also dimensionality reduction. Despite this kind of transformation being widely available (any neural network package that allows access to the hidden layer weights can be used), no ex-

J. M. Valls
Universidad Carlos III de Madrid
E-mail: jvalls@inf.uc3m.es

R. Aler
Universidad Carlos III de Madrid
E-mail: aler@inf.uc3m.es

I. M. Galván
Universidad Carlos III de Madrid
E-mail: igalvan@inf.uc3m.es

D. Camacho
Universidad Politécnica de Madrid
E-mail: david.camacho@upm.es

tensive experimentation on the quality of 3-layer MLP data transformation has been carried out. The aim of this article is to carry out this research for classification problems. Results show that, overall, this transformation offers better results than the PCA unsupervised transformation method.

1 Introduction

The aim of dimensionality reduction is to transform data in a high-dimensionality space into a lower one in order to facilitate machine learning (ML) tasks such as classification, regression, or clustering. With respect to classification and regression, it is known that some issues like irrelevant attributes, redundant attributes, or the curse of dimensionality may worsen the performance of ML methods. This is specially so for techniques such as K-Nearest Neighbour (KNN), as it does not construct a model but depends on distances computed on the attributes of the training dataset. Feature/attribute selection techniques can be used to discard irrelevant or redundant attributes. However, these techniques can either keep or remove the attribute, which in some cases may lead to loss of information. Also, feature selection does not transform the data space (at least, not beyond removing attributes), and in some cases, ML tasks may obtain better performance in transformed spaces. Feature extraction techniques can be used for that purpose. Principal Components Analysis (PCA) is one of the most widely used technique for both transforming the data space and reducing dimensionality. PCA computes the attributes that explain the greatest variance of the data. It basically finds a subspace that includes most of the data variance, and those dimensions with low variance are considered irrelevant. This is achieved by transforming the data space by constructing new (linearly) uncorrelated attributes. PCA has been used successfully in classification and regression tasks as a transformation and dimensionality reduction technique. However, PCA is an unsupervised method because it does not take into account the response variable in order to compute the transformation. Henceforth, although it may depend on the problem, there is no guarantee that the transformation will be the most appropriate one for some supervised ML tasks. There have been approaches where evolutionary algorithms are used to evolve a supervised data transformation in order to improve classification tasks [Valls and Aler(2009), Echeverria et al.(2012), Aswolinskiy(2018)], with good results, but as is common in evolutionary approaches, computationally demanding.

Multi Layer Perceptrons (MLP) are feedforward neural networks composed of a sequence of layers that alternate linear and non-linear operations. A minimal MLP has three layers: input, hidden, and output. The hidden layer transforms the data instance present in the input layer by applying a linear transformation (matrix multiplication) followed by a non-linear operation (typically sigmoid, tanh, and nowadays relu). The output layer performs a similar transformation after the hidden layer has been applied. This operation sequence from inputs to outputs is called 'forward'. MLP's are trained by means of

backpropagation, which updates the matrices of the hidden and output layers by propagating the gradient of the error computed in the output layer (comparing the MLP's output and the desired output), backwards towards the inputs. This is the so called 'backward' process. Although MLP's can be used to carry out classification or regression, it is interesting to note that they can be interpreted as a two-step process (in the case of one hidden layer MLP). First, the hidden layer transforms the inputs into a new space (i.e., it changes the representation of the data); second, the output layer performs the supervised task in the new space. In fact, in the case of classification, which is the purpose of this article, the hidden layer can be understood as a transformation that converts the original data into a new space where the output layer can carry out linear classification. If the number of hidden neurons is smaller than the number of inputs, both data transformation and dimensionality reduction are taking place.

The aim of this article is to explore the performance of data transformations carried out by the hidden neurons of 3-layer MLP's. This transformation method will be treated as any machine learning method, with hyper-parameters that must be adjusted using proper methodologies. One example of such hyper-parameter is the optimal number of dimensions of the new feature space. In order to assess the quality of the data transformation, KNN will be used to compute accuracy on the transformed data. KNN is appropriate for this task, as it is a method where the model is the training data, and therefore only performs well if data representations are correct. In any case, the transformation method could be used with any other machine learning classification method. Finally, the 3-layer MLP transformation will be compared to PCA.

The structure of this article is as follows. Section 2 contextualizes the proposal within the related work. Section 3 describes how data can be transformed using a 3-layer MLP. Section 4 illustrates the process by applying it to a synthetic domain. Section 5 describes the 50 domains used, explains the experimental methodology and presents the results. Finally, Section 6 draws the main conclusions of this work. An R software package that implements the ideas discussed in this article can be found in [Aler et al.(2020)][1].

2 Related Work

The idea that layers in MLP's can be understood as a change of data representation is not new [Rumelhart et al.(1986)]. For instance, [Hinton and Salakhutdinov(2006)] explains how neural networks can be used for dimensionality reduction and compares it with PCA. However, that work was carried out with autoencoders, a kind of MLP whose purpose is to reconstruct the inputs. For the most usual autoencoders, that means the number of outputs of autoencoders is the same as the number of inputs, and the goal is that outputs and inputs are as similar as possible. If only a few neurons are present in the

[1] Please note that the doughnut dataset available in the package is smaller than the one used for this article, because of space constraints at the CRAN repository.

hidden layer, autoencoders force the network to represent the data using fewer dimensions. But autoencoders are still an unsupervised technique (like PCA), because dimensionality reduction is not carried out taking into account classification performance. [Van Der Maaten(2009)] explores this idea by learning a deep autoencoder mapping so that the local structure of the data is preserved in the new representation of data. It was tested on the USPS and MNIST handwritten digit recognition datasets and the 20 newsgroup dataset, by reducing dimensionality to 2, 10, and 30. Another work that uses deep autoencoders is [Salakhutdinov and Hinton(2007)]. In this case, a pre-trained autoencoder is used to find a low-dimension representation of the data, which is subsequently fine-tuned by using a cost function that maximizes the accuracy of KNN. Although the initial dimensionality reduction method itself is unsupervised, the whole process is supervised, and a KNN-appropriate representation is learned while optimizing a proxy for accuracy. The methodology was tested on MNIST, using dimensions from 2 to 30. A similar approach, now in the context of large-margin KNN classification, was employed by [Min et al.(2009)] and tested on USPS, MNIST and 20-newsgroup datasets. Other cost functions have been employed following the same methodology [Min et al.(2010)].

Deep autoencoders continue to be a common theme for feature extraction and dimensionality reduction in recent research work. [Lv et al.(2018)] uses a contractive autoencoder for change detection in synthetic aperture radar images, which compares favourably with PCA and Markov random fields. Another article proposes a convolutional deep learning autoencoder to support unsupervised image feature extraction for lung nodule using unlabeled data, requiring only a small amount of labeled data [Chen et al.(2017)]. Autoencoders are not the only neural-network related unsupervised technique used for feature extraction. Another commonly used method is the self-organized-maps (SOM). For instance, in [Febrianto et al.(2020)] SOM are used to reduce dimensionality for clustering (not a supervised problem in this case). [Sakkari et al.(2020)] extends the idea of SOM to the deep learning context by proposing a convolutional deep SOM for feature extraction. In [Aissa et al.(2017)], a SOM is used for different views of an image (color, texture, and shape) so that a multi-view feature vector is obtained. The technique is applied to image classification problems.

More closely related to the research presented here, [Parviainen and Vehtari(2009),Parviainen et al.(2011)] studies a MLP-based supervised approach for classification in 5 domains. In this case, the MLP is not an autoencoder, but a standard 3-layer MLP where the output of the network is the class. Therefore, the feature space (i.e., the data transformation computed at the hidden layer) is obtained by the MLP after the network has optimized its cost function (related to classification accuracy in classification problems). This new feature space could be used by KNN or by any other machine learning method. This possibility for the 3-layer MLP is not explored in [Parviainen and Vehtari(2009), Parviainen et al.(2011)], because its main purpose is visualization (reduction to 2 dimensions). What is actually studied is a more complex MLP architecture, a so called bottleneck network, with 6 layers, con-

taining x, y, z, d, h, and c neurons, respectively. x, y, and z are the bottleneck layers, which for the MNIST problem contain 1000, 500, and 250 neurons, respectively (but this may vary for other domains). d is the dimension of the new feature space, and two values were tested: 30 and 2 (the latter for data visualization purposes). h is the number of neurons in an additional layer and c is the output layer, which in classification problems has as many neurons as classes (c). MNIST, USPS and the 20 newsgroups datasets were used for validation. The quality of the new representation was tested by measuring the accuray of KNN on the transformed datasets.

Currently, the idea that modern convolutional deep networks (CNNs) [Martín et al.(2020), Martín et al.(2018)] can be understood as automatic feature extractors is widely acknowledged, and studied on some concrete domains, mainly related to image classification and processing. Some examples are [Bluche et al.(2013)] (where it is used for handwritten word recognition), [Chen et al.(2016)] (hyperspectral image classification), or [Yang et al.(2019)] (tumor image classification). In [Huertas-Tato et al.(2020)] a CNN is used as a source of features to feed a meta-learner (a Support Vector Machine) in order to classify cloud images into types (additionally, features are also obtained from a Random Forest and fused with those of the CNN). [Khan et al.(2019)] follows a similar approach, where Resnet deep networks are used for feature extraction, which are subsequently given to a SVM classifier. Recent research uses a CNN and a feature selector as a source of features, which are used as input to a forest ensemble of MLP's [Mejri and Mejri(2020)]. The method is shown to work well on image classification domains with small datasets.

From the literature review above, it can be seen that within the neural network field, the most commmonly used approaches for data transformation are autoencoders, which are unsupervised techniques. Also, CNN's are commonly used as supervised feature extractors, but mostly for image related problems, which is the natural context for convolutional networks. Therefore, the question remains of how well the simpler 3-layer MLP would perform as feature extractor on common tabular datasets (not related to images). This question has some practical significance. Many practitioners who want to carry out feature transformation/reduction resort to PCA, as it is a simple and widely available technique, which is known to produce good results. However, it has already been pointed out that PCA is unsupervised, and therefore, not guaranteed to offer good performance in supervised tasks, like classification. On the other hand, a 3-layer MLP architecture is one of the most simple approaches to carry out supervised reduction/transformation of data. It is widely available and simple to use, because all that is required is that the hidden layer weights can be extracted after training the network. Despite these advantages, a 3-layer MLP for this purpose has not been hitherto extensively evaluated. The main contribution of this article is to fill in this gap by evaluating the approach on a large set of tabular classification problems (beyond the few typically used in similar research). Additionally, an R package that implements the ideas described in this article is also provided [Aler et al.(2020)].

3 Transforming data with a 3-layer MLP

From now on, **nntrf** (neural net transformation) will stand for the 3-layer MLP data transformation to be assessed in this research. Its aim is to use the weights of the hidden layer of a 3-layer MLP as a way to transform the data. A 3-layer MLP contains three layers: inputs ($\mathbf{x}$), hidden layer ($\mathbf{W_1}$) and output layer ($\mathbf{W_2}$). In its most standard form, which is the approach taken here, $\mathbf{x}$ is a one-dimensional vector, with as many elements as input attributes, plus a 1: $\mathbf{x} = (1, x_1, x_2, x_n)$. $\mathbf{W_1}$ is a $m \times (n+1)$ matrix, where n is the number of attributes and m is the number of neurons in the hidden layer. $\mathbf{W_2}$ is a $c \times m$ matrix, where c is the number of classes in the classification problem. $\mathbf{W_1}$ and $\mathbf{W_2}$ are known as the 'weights' of the hidden and output layers, respectively.

Mathematically, the MLP is a mathematical model that performs the following transformation $\hat{\mathbf{y}} = S(\mathbf{W_2} \star S(\mathbf{W_1} \star \mathbf{x}))$, where $S()$ is a non-linear function, known as the activation function. Commonly, $S()$ is the sigmoid function. Given a training set with N instances $\{(\mathbf{x_1}, y_1), (\mathbf{x_1}, y_1), \ldots, (\mathbf{x_N}, y_N)\}$, fitting a MLP model means finding the weights $\mathbf{W_1}$ and $\mathbf{W_2}$, so that the output of the network $\hat{\mathbf{y}}_\mathbf{i}$ is as close as possible to the desired output y_i. This is achieved by applying optimization techniques, mainly backpropagation, so that the error of the network is minimized [Rumelhart et al.(1986)]. In the research carried out in this article, we have used the **nnet** package, which for classification problems uses crossentropy as the loss function [Venables and Ripley(2002)].

The aim of **nntrf** is to train a MLP with some training data and then use $\mathbf{W_1}$ to transform datasets via $\mathbf{x'_i} = S(\mathbf{W_1} \star \mathbf{x_i})$. In some cases, it can be interesting to use only the linear transformation $\mathbf{x'_i} = \mathbf{W_1} \star \mathbf{x_i}$. Obviously, the same transformation can be applied to new (test) data. This transformation is supervised, because a MLP was trained to learn the classification problem, as opposed to unsupervised transformations like PCA. It is important to notice that the dimensionality of the original data n can be different from the dimensionality of the transformed data m. Typically $m < n$ if dimensionality reduction is desired. If $m < n$, the information originally contained in the n attributes will be concentrated into the m new attributes, forcing the transformation to remove irrelevant or redundant features.

In some cases, MLP training may get stuck in local minima. To avoid that, our method allows to repeat the training process several times and keep the best MLP obtained in training. This is displayed in Algorithm 1, where trainingLoss is the loss function used by the gradient descent algorithm to optimize the MLP weights. In the case of this work is cross entropy, which is typically used for classification problems in neural networks. All this functionality has been included into the **nntrf** R software package [Aler et al.(2020)].

Data: X and y is the training data
Data: *repetitions* is the number of repetitions
Result: W_1, W_2

trainingLoss, W_1, $W_2 \leftarrow$ trainMLP(X, y, *repetitions*))
repetitions $\leftarrow$ *repetitions* $- 1$
while *repetitions* > 0 **do**
  trainingLoss$'$, W_1', $W_2' \leftarrow$ trainMLP(X, y, *repetitions*))
  if *trainingLoss*$'$ $<$ *trainingLoss* **then**
    trainingLoss $\leftarrow$ *trainingLoss*$'$
    $W_1 \leftarrow W_1'$
    $W_2 \leftarrow W_2'$
  end
  repetitions $\leftarrow$ *repetitions* $- 1$
end
return W_1, W_2

Algorithm 1: Pseudo-code for obtaining the W_1 and W_2 weight matrices.

4 Results on the Doughnut Synthetic Domain

In order to illustrate the behavior of the method, we have built a synthetic dataset called **doughnutRandRotated** in the following way. The initial domain is a two-dimensional dataset, the **Doughnut**, with two classes that can be seen in Figure 1. It must be noted that in order to achieve high classification accuracy on this dataset, a non-linear boundary is required (one that approximates two concentrical circumferences). This dataset has been then transformed by adding 8 random features with uniform noise between 0 and 1. Finally, we perform a random rotation to this 10-feature dataset. The aim of the two latter transformations is to add irrelevant information (8 noise attributes), which is then mixed with the 2 original relevant attributes. The final **doughnutRandRotated** domain contains 10 attributes, with relevant information spread on them, and mixed with noise.

The goal of **nntrf** here is to recover the original **Doughnut** from the final dataset **doughnutRandRotated**. In other words, we intend that after training the MLP, the transformation $\mathbf{x_i'} = S(\mathbf{W_1} \star \mathbf{x_i})$ or $\mathbf{x_i'} = \mathbf{W_1} \star \mathbf{x_i}$ is able to transform the $\mathbf{x_i}$ of the **doughnutRandRotated** into the $\mathbf{x_i'}$ of the original **Doughnut** (or something close to it).

For illustration purposes, a MLP with 5 hidden neurons has been used. This means that **nntrf** will transform a 10-attribute dataset into a 5-attribute dataset. Afterwards, **KNN** with 1 neighbor is applied to the resulting transformed data. **KNN** is a lazy ML method that does not build an explicit model. Instead, it is assumed that the best model of the data is the data itself and therefore uses the training data to classify new instances. It is known that **KNN** does not behave well when dimensionality is high or when there are

irrelevant or redundant attributes. For this reason, **KNN** is a good choice to evaluate the quality of the features generated by **nntrf** (although in principle, the transformed dataset could be used by any other ML method).

The success rate of **KNN** goes from 0.674025 (before the **nntrf** transformation) to 0.897975 (if the linear transformation $\mathbf{x'_i} = \mathbf{W_1} \star \mathbf{x_i}$ is used) or 0.712025 (if the non-linear transformation $\mathbf{x'_i} = S(\mathbf{W_1} \star \mathbf{x_i})$ is used). For this problem, the linear transformation works better, even though the MLP was trained using a sigmoid in the hidden layer.

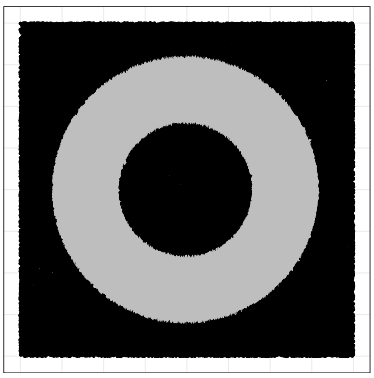

Fig. 1 Doughnut dataset. Two classes: black and grey.

Interestingly, attributes 2 and 5 of the transformed dataset have recovered the doughnut to some extent (see Figure 2).

If the training of the MLP is done for 5 repetitions (as explained in the previous section in Algorithm 1), it is possible to get 0.9914 KNN success rate with only 3 hidden neurons (compared to 5 hidden neurons previously).

Next, more extensive results of **nntrf** on the **doughnutRandRotated** dataset will be presented in order to understand better a few aspects of the method. The MLP training process is stochastic, because the initial weights are random. In order to assess the influence of the random initialization, each configuration has been run 50 times, with different random seeds. Every run is called 'experimental run' in what follows. This must be differentiated from the number of repetitions: one experimental run can do several repetitions (in order to avoid local minima), and the best repetition (evaluated on the training data) is selected for that experimental run (see Algorithm 1).

Figure 3 displays the percentage of experimental runs (out of 50) where the success rate is larger than 80%. The purpose of this figure is to assess how frequently the method achieves good solutions (in this case, it is considered good a solution that obtains 80% accuracy with KNN after the **nntrf** transformation). Rows "No Sigmoid" and "Sigmoid" display results for the dataset having been transformed by **nntrf** with a linear tranformation ($\mathbf{x'_i} = \mathbf{W_1} \star \mathbf{x_i}$) and with the sigmoid tranformation ($\mathbf{x'_i} = S(\mathbf{W_1} \star \mathbf{x_i})$), respectively and then

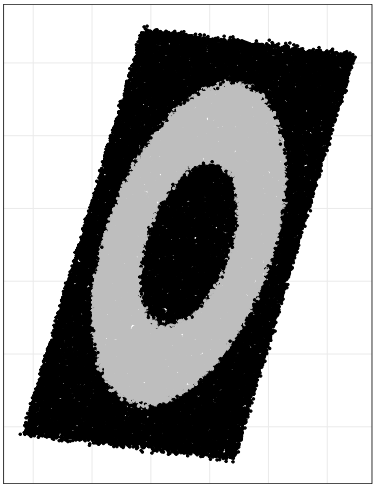

Fig. 2 Data transformed by **nntrf**. Only attributes 2 and 5 are plotted.

classified with KNN (with number of neighbors k=1). Row "MLP" shows the classification results of a standard MLP. The x-axis stands for the number of hidden neurons (from 1 to 10) and columns 1 and 5 stand for the number of times the MLP training process has been repeated (1 for no repetition, 5 for 5 repetitions, in order to avoid random initial weights that lead to local minima). Colors are used to represent different number of MLP training iterations (10, 100, 1000).

It can be seen that for **nntrf**+KNN, 'No Sigmoid' works slightly better than 'Sigmoid', at least for this problem. For instance, for 4 neurons and 100 iterations (and 1 repetition), 'No-Sigmoid' obtains about 62% of the 50 experimental runs having KNN accuracy larger than 80%, while 'Sigmoid' gets around 25%. Therefore, in some cases (like this one), the linear transformation can be the right one, even though the original MLP used a Sigmoid for the hidden layer.

It is interesting to note that **nntrf**+KNN ('No Sigmoid') works much better than MLP. For 1 repetition, **nntrf**+KNN needs only 100 iterations and 5 neurons to get nearly 75% of experimental runs obtaining a success rate larger than 80%, while MLP requires 1000 iterations and 8 neurons to get a similar value. This makes sense, because a MLP requires many hidden neurons in order to approximate well the doughnut (the boundary is made of two concentrical circumferences), while only a few neurons are required to find a good transformation that can be used by KNN (and fewer iterations). After all, the transformation from **doughnut** to **doughnutRandRotated** (adding 8 random columns, followed by a rotation) can be reversed by multiplying by a matrix that undoes the rotation and selects the relevant columns, and it is expected that $\mathbf{W_1}$ will do that approximately. But it is important to note that for some problems (like this one), obtaining the data transformation

may be simpler (i.e., using fewer iterations or hidden neurons) than fitting the complete MLP.

Finally, it can be seen that when using one repetition, **nntrf** requires 5 neurons to solve the problem, while theoretically, only 2 should be needed (there are only two relevant attributes). The reason seems to be that with just one repetition, using few hidden neurons leads the training of the MLP to get stuck in local minima. When doing 5 repetitions, it can be seen in Figure 3 that almost a 100% of solutions can be reached with just 3 neurons and 100 iterations. This is still not a solution with 2 neurons, but is closer to the theoretical optimal number. It is also interesting to notice that even with 5 repetitions, a MLP requires 6 neurons and 1000 iterations to achieve values close to 100% of good solutions.

Similar conclusions can be reached from Figure 4 where the average success rate of the 50 runs is displayed.

5 Experiments

5.1 Datasets

The **doughnutRandRotated** was intended to illustrate **nntrf**. In this Section, the **nntrf** method will be applied to 50 widely used benchmark datasets (plus the **doughnutRandRotated** dataset). Table 5.1 shows the characteristics of all the datasets: name, number of instances, number of classes, number of attributes, number of categorical attributes, and final number of attributes. The latter is the actual number of attributes used in that domain, after the categorical ones have been encoded with one-hot-encoding (dummy variables). Thus, all attributes are numerical, because the dummy variables contain binary values. All attributes in all domains have been standarized.

5.2 Experimental Validation

In order to evaluate the performance of **nntrf**, a set of experiments has been done with the original datasets described in the previous section. Also, in order to assess the ability of **nntrf** to identify the relevant information in a dataset, extended versions of the domains have been prepared as follows. For each dataset (with n attributes, as seen in the last column of Table 5.1), an extended version is generated by adding 20 random attributes (with uniform values between 0 and 1) to the n original ones and performing a random rotation to these $20 + n$ attributes. This is a similar transformation carried out to get the **doughnutRandRotated** domain.

For each domain (original and extended) four methods are applied: KNN, standard neural networks (MLP), PCA, and **nntrf**. In order to evaluate each method, a 3-fold Cross Validation (CV) has been run for each dataset.

nntrf applies a transformation with a given number of neurons and then applies KNN to the transformed data. Therefore the hyper-parameters of the

Dataset	Instances	Classes	Attributes	Categorical attributes	Final number of attributes
bgp	24984	4	9	0	9
boundary	3505	2	175	0	175
breast-w	569	2	30	0	30
bupa	345	2	6	0	6
cam	18916	2	132	0	132
car	1728	4	6	6	21
compustat	10358	2	20	0	20
covtype	38500	2	10	0	10
credit-g	1000	2	20	13	61
dna	3186	3	180	0	180
doughnutRandRotated	100000	2	10	0	10
estate	5322	2	12	0	12
fourclass	862	2	2	0	2
german.numer	1000	2	24	0	24
glass	214	6	9	0	9
heart-c	303	2	13	2	20
ion	351	2	34	0	34
ism	11180	2	6	0	6
krkp	3196	2	36	36	73
led-24	5000	10	24	0	24
letter	20000	2	16	0	16
letter-26	36000	26	16	0	16
nursery	12958	4	8	8	27
optdigits	5620	2	64	0	64
page	5473	2	10	0	10
page-blocks-5	5473	5	10	0	10
pendigits	10992	2	16	0	16
pendigits-10	10992	10	16	0	16
phoneme	5400	2	5	0	5
pima	768	2	8	0	8
promoters	106	2	57	57	228
ringnorm	300	2	20	0	20
sat	6435	6	36	0	36
satimage	6430	2	36	0	36
segment	2310	2	19	0	19
segment-7	2310	7	19	0	19
shuttle	58000	7	9	0	9
sonar	208	2	60	0	60
spambase	4601	2	57	0	57
splice	1000	2	60	0	60
splice-libsvm	1000	2	60	0	60
SVMguide1	3089	2	4	0	4
threenorm	300	2	20	0	20
tic-tac-toe	958	2	9	9	27
twonorm	300	2	20	0	20
vehicle	846	4	18	0	18
vote	435	2	16	16	48
vote1	435	2	15	15	45
vowel	528	11	10	0	10
waveform	5000	3	21	0	21
zip	9298	10	256	0	256

Table 1 Datasets Description.

complete **nntrf**+KNN pipeline are the number of hidden neurons (**size**), the number of **iterations**, the number of KNN neighbors **k**, and whether sigmoid is used or not (**sigmoid**). The hyper-parameters of the other methods are **size** and **iterations** for MLP, **k** and **size** for the PCA+KNN pipeline, and **k** for KNN. All of them with the same possible values as in **nntrf**. To decide the best hyper-paramenter combination, grid-search has been used, evaluating hyper-parameters by means of an internal 3-fold CV (using only the training

partitions). Modern machine learning libraries, such as scikit-learn for Python [Pedregosa et al.(2011)] or Caret [Kuhn(2013)] and MLR [Bischl et al.(2016)] for R, offer utilities for doing standard tasks such as hyper-parameter tuning or pipelines with data pre-processing (although they can be programmed by hand as well). In the case of this article, the MLR package [Bischl et al.(2016)] has been used to build the MLP, KNN, **nntrf**+KNN, and PCA+KNN pipelines, and then doing grid-search on their hyper-parameters.

The set of hyper-parameter values that have been explored by grid-search is summarized below:

- **size** (number of hidden neurons or PCA components): 1,2,3, ..., 10
- **iterations** (number of iterations of the MLP): 10, 50, 100, 200, 300, 400, 500, ..., 1000
- **k** (number of KNN neighbors): 1, 2, 3, 4, ..., 10
- **sigmoid** (whether sigmoid is used after the transformation): yes, no

The averages of all these values for each method (KNN, MLP, PCA, and **nntrf**) over all the original datasets are shown in Table 2. Table 3 shows the same information for the extended datasets. It can be observed that the number of k neighbors increases for the domains with noise, from around 5 to 6. This should be expected because KNN requires more neighbors for noisy domains. No remarkable differences are found between the methods with respect to the number of iterations (around 300) or the number of components (around 7). It can be seen that sigmoid is used only about 30% of the time.

Figures 5 to 6 display scatter plots to compare the error obtained by **nntrf** versus the rest of methods. Figure 5 compares **nntrf**+KNN and KNN. The aim of this comparision is to check whether **nntrf** transformation improves the performance of KNN. The left figure corresponds to the original datasets and the right figure to the extended datasets. Each point indicates the error rate obtained by both KNN (horizontal axis) and **nntrf** (vertical axis). The points below the diagonal line correspond to the datasets where **nntrf** behaves better than KNN (i.e., the error obtained by **nntrf** is lower than the one obtained by KNN). We can see that, although a few points are above the diagonal, they are very close to it, so **nntrf** is only slightly worse in those cases. However, there are more points below the diagonal and many of these are far below it. We can also observe that for the domains with noise (Figure 5, right) more domains are placed below the diagonal, and fewer above, which means that the performance of **nntrf** is even better for the noisy domains. This is important, because that is exactly what should be expected from a method able to extract information from the original feature space.

The purpose of Figure 6 is to visualize the two transformation methods **nntrf** and PCA. In both cases, classification is done with KNN, after the data transformation. This plot is important, because it illustrates the difference between the supervised and unsupervised methods. It can be observed that several points lay in the diagonal, or close to it, which suggests a similar performance of PCA and **nntrf** for those domains. This confirms that PCA is a good transformation method, despite being unsupervised. However, it can be

seen that more domains are below the diagonal, and some of them show a large improvement of **nntrf** over PCA (those points far below the diagonal). This superior performance of **nntrf** is even more clear for the extended datasets, which shows that using supervised transformation methods is more important when the set of attributes contain large amounts of irrelevant information.

Finally, Figure 7 compares **nntrf**+KNN with the MLP. As explained in Section 3, a 3-layer MLP can be understood as a transformation (hidden layer) followed by a classification performed by the output layer. Therefore, what Figure 7 compares is whether KNN is a better classifier than the MLP output layer, given the transformation carried out by the hidden layer. This comparison is less relevant for the purposes of this article, as our main goal is to compare supervised vs. unsupervised transformations. Besides, what classification model is superior is going to depend on the domain. However, it is still interesting to visualize the performance of a classifier that does not construct a model (KNN) versus a classifier that constructs an actual model from the training data (MLP). It can be seen that indeed, for most domains, KNN performs better (points below the diagonal), therefore the combination of a model-less classifier and a supervised transformation turns out to obtain lower error results.

Method	k	iterations	size	sigmoid
KNN	4.9739	NA	NA	NA
MLP	NA	286.2745	6.7647	NA
nntrf+KNN	5.2680	318.1699	7.1961	0.301
PCA+KNN	5.4052	NA	7.6144	NA

Table 2 Hyper-parameters mean values for each method. Original datasets.

Method	k	iterations	size	sigmoid
KNN	6.3529	NA	NA	NA
MLP	NA	312.6797	6.1046	NA
nntrf+KNN	5.9020	258.5621	7.1503	0.320
PCA+KNN	5.7843	NA	6.5948	NA

Table 3 Hyper-parameters mean values for each method. Extended datasets.

6 Conclusions

3-layer MLP's can be understood as a sequence of two processes, a transformation of the original data into a new representation (carried out by the hidden layer) followed by a classification on the new feature space (carried out by the output layer). This means that a trained MLP offers a way to transform datasets, using the hidden layer after the MLP has been fit to the training data. If the number of neurons of the hidden layer is small compared to the

number of inputs, dimensionality reduction also takes place. This transformation is supervised, because it was obtained by the MLP training process which aims to improve the classification ability of the network. This constrasts with other widely used transformation methods, such as PCA, which are not supervised, and therefore not guaranteed to provide transformed data optimized for classification.

Despite its simplicity and wide availability, transformations obtained by 3-layer MLP's have not been extensively experimented with, as far as we know. The aim of this article is to carry out such experimentation so that the practitioner has a clear idea of the advantages of the MLP supervised transformation, named **nntrf** here. The method has been experimented on 50 benchmark classification domains. In order to assess the quality of **nntrf**, the KNN classification method has been used, because it is a method that does not construct a model from the data, but assumes that the training data is its best model, and therefore, depends largely on the data representation being adequate. **nntrf** has been compared with the widely used unsupervised transformation PCA. An important hyper-parameter of the transformations is the number of hidden neurons (for **nntrf**) or the number of components (which is the respective parameter for PCA). The optimal value of this hyper-parameter, and others, such as the number of training MLP iterations or the number of KNN neighbors, has been chosen using standard ML procedures (cross-validation grid-search in this case). Given that the training of MLP's depends on the random initial weights, every MLP fitting has been repeated 50 times. All methods have been evaluated by means of 3-fold crossvalidation.

Results show that the supervised transformation **nntrf** performs better than PCA. There are some domains where PCA obtains better accuracies than **nntrf**, but the difference is small in those cases. More importantly, there are more domains where **nntrf** performs better than PCA, and in some of these cases, **nntrf**'s error is much smaller. This is true specially in problems with attribute noise.

7 Acknowledgements

This work has been supported by Agencia Estatal de Investigación (PID2019-107455RB-C22 /AEI/ 10.13039/501100011033), and Spanish Ministry of Science and Education under TIN2017-85727-C4-3-P (DeepBio) grant.

References

[Aissa et al.(2017)] Aissa, Fatma Ben, et al. Unsupervised Features Extraction Using a Multi-view Self Organizing Map for Image Classification. 2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA). IEEE, 2017.

[Aler et al.(2020)] Aler R, Valls J M, Galván I M, Camacho D (2020) nntrf: Supervised Data Transformation by Means of Neural Network Hidden Layers. R package version 0.1.3. https://CRAN.R-project.org/package=nntrf

[Aswolinskiy(2018)] Aswolinskiy, Witali. Learning in the Model Space of Neural Networks. Bielefeld: Universitat Bielefeld. (2018).

[Bischl et al.(2016)] Bischl B, Lang M, Kotthoff L, Schiffner J, Richter J, Studerus E, Casalicchio G, Jones ZM (2016) mlr: Machine learning in r. The Journal of Machine Learning Research 17(1):5938–5942

[Bluche et al.(2013)] Bluche T, Ney H, Kermorvant C (2013) Feature extraction with convolutional neural networks for handwritten word recognition. In: 2013 12th International Conference on Document Analysis and Recognition, IEEE, pp 285–289

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

[Chen et al.(2017)] Chen, M., Shi, X., Zhang, Y., Wu, D., and Guizani, M. (2017). Deep features learning for medical image analysis with convolutional autoencoder neural network. IEEE Transactions on Big Data.

[Echeverria et al.(2012)] Echeverría, Alejandro, José María Valls, and Ricardo Aler. Evolving linear transformations with a rotation-angles/scaling representation. Expert Systems with Applications 39.3 (2012): 3276-3282.

[Febrianto et al.(2020)] Febrianto, R. T., Saputra, D. M., and Jambak, M. I. (2020). Dimension Reduction with Extraction Methods (Principal Component Analysis-Self Organizing Map-Isometric Mapping) in Indonesian Language Text Documents Clustering. In Hybrid Intelligent Systems: 19th International Conference on Hybrid Intelligent Systems (HIS 2019) Held in Bhopal, India, December 10-12, 2019 (Vol. 1179, p. 1). Springer Nature.

[Hinton and Salakhutdinov(2006)] Hinton GE, Salakhutdinov RR (2006) Reducing the dimensionality of data with neural networks. science 313(5786):504–507

[Huertas-Tato et al.(2020)] Huertas-Tato, Javier, Alejandro Martín, and David Camacho. Cloud Type Identification Using Data Fusion and Ensemble Learning. International Conference on Intelligent Data Engineering and Automated Learning. Springer, Cham, 2020.

[Khan et al.(2019)] Khan, Muhammad Attique, et al. Multi-model deep neural network based features extraction and optimal selection approach for skin lesion classification. 2019 international conference on computer and information sciences (ICCIS). IEEE, 2019.

[Kuhn(2013)] Kuhn M (2013) Predictive modeling with r and the caret package. Google Scholar

[Lv et al.(2018)] Lv, N., Chen, C., Qiu, T., and Sangaiah, A. K. (2018). Deep learning and superpixel feature extraction based on contractive autoencoder for change detection in SAR images. IEEE transactions on industrial informatics, 14(12), 5530-5538.

[Martín et al.(2018)] Martín A, Lara-Cabrera R, Fuentes-Hurtado F, Naranjo V, Camacho D (2018) Evodeep: a new evolutionary approach for automatic deep neural networks parametrisation. Journal of Parallel and Distributed Computing 117:180–191

[Martín et al.(2020)] Martín A, Vargas VM, Gutiérrez PA, Camacho D, Hervás-Martínez C (2020) Optimising convolutional neural networks using a hybrid statistically-driven coral reef optimisation algorithm. Applied Soft Computing 90:106144

[Mejri and Mejri(2020)] Mejri, M., and Mejri, A. (2020). RandomForestMLP: An Ensemble-Based Multi-Layer Perceptron Against Curse of Dimensionality. arXiv preprint arXiv:2011.01188.

[Min et al.(2010)] Min M, van der Maaten L, Yuan Z, Bonner AJ, Zhang Z (2010) Deep supervised t-distributed embedding. In: ICML

[Min et al.(2009)] Min R, Stanley DA, Yuan Z, Bonner A, Zhang Z (2009) A deep non-linear feature mapping for large-margin knn classification. In: 2009 Ninth IEEE International Conference on Data Mining, IEEE, pp 357–366

[Parviainen and Vehtari(2009)] Parviainen E, Vehtari A (2009) Features and metric from a classifier improve visualizations with dimension reduction. In: International Conference on Artificial Neural Networks, Springer, pp 225–234

[Parviainen et al.(2011)] Parviainen E, et al. (2011) Studies on dimension reduction and feature spaces

[Pedregosa et al.(2011)] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, et al. (2011) Scikit-learn: Machine learning in python. the Journal of machine Learning research 12:2825–2830

[Rumelhart et al.(1986)] Rumelhart DE, Hinton GE, Williams RJ (1986) Learning representations by back-propagating errors. nature 323(6088):533–536

[Sakkari et al.(2020)] Sakkari, Mohamed, and Mourad Zaied. A Convolutional Deep Self-Organizing Map Feature extraction for machine learning. Multimedia Tools and Applications (2020): 1-20.

[Salakhutdinov and Hinton(2007)] Salakhutdinov R, Hinton G (2007) Learning a nonlinear embedding by preserving class neighbourhood structure. In: Artificial Intelligence and Statistics, pp 412–419

[Valls and Aler(2009)] Valls, José M., and Ricardo Aler. Optimizing linear and quadratic data transformations for classification tasks. 2009 Ninth International Conference on Intelligent Systems Design and Applications. IEEE, 2009.

[Van Der Maaten(2009)] Van Der Maaten L (2009) Learning a parametric embedding by preserving local structure. In: Artificial Intelligence and Statistics, pp 384–391

[Venables and Ripley(2002)] Venables WN, Ripley BD (2002) Modern Applied Statistics with S, 4th edn. Springer, New York, URL http://www.stats.ox.ac.uk/pub/MASS4, iSBN 0-387-95457-0

[Yang et al.(2019)] Yang A, Yang X, Wu W, Liu H, Zhuansun Y (2019) Research on feature extraction of tumor image based on convolutional neural network. IEEE Access 7:24204–24213

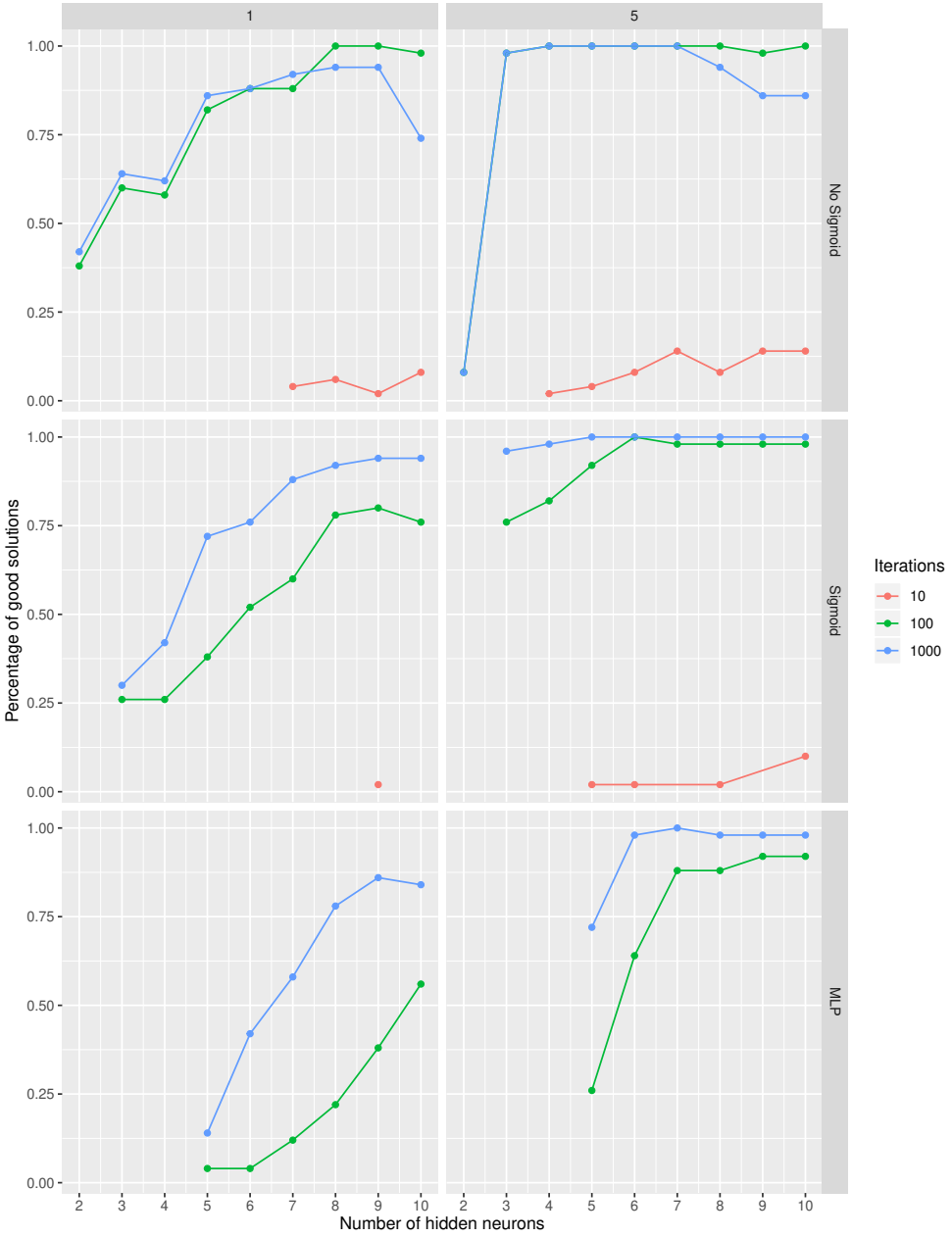

Fig. 3 Percentage of experimental runs (out of 50) for which the success rate is higher than 80% (y-axis) vs. number of hidden neurons. Columns 1 (left) and 5 (right) stand for training the MLP with 1 and 5 repetitions, respectively. Colors indicate the number of MLP training iterations. 'No-Sigmoid' and 'Sigmoid' represent the linear and non-linear transformations prior to using KNN for classification, while 'MLP' is the result of using the MLP directly for classification.

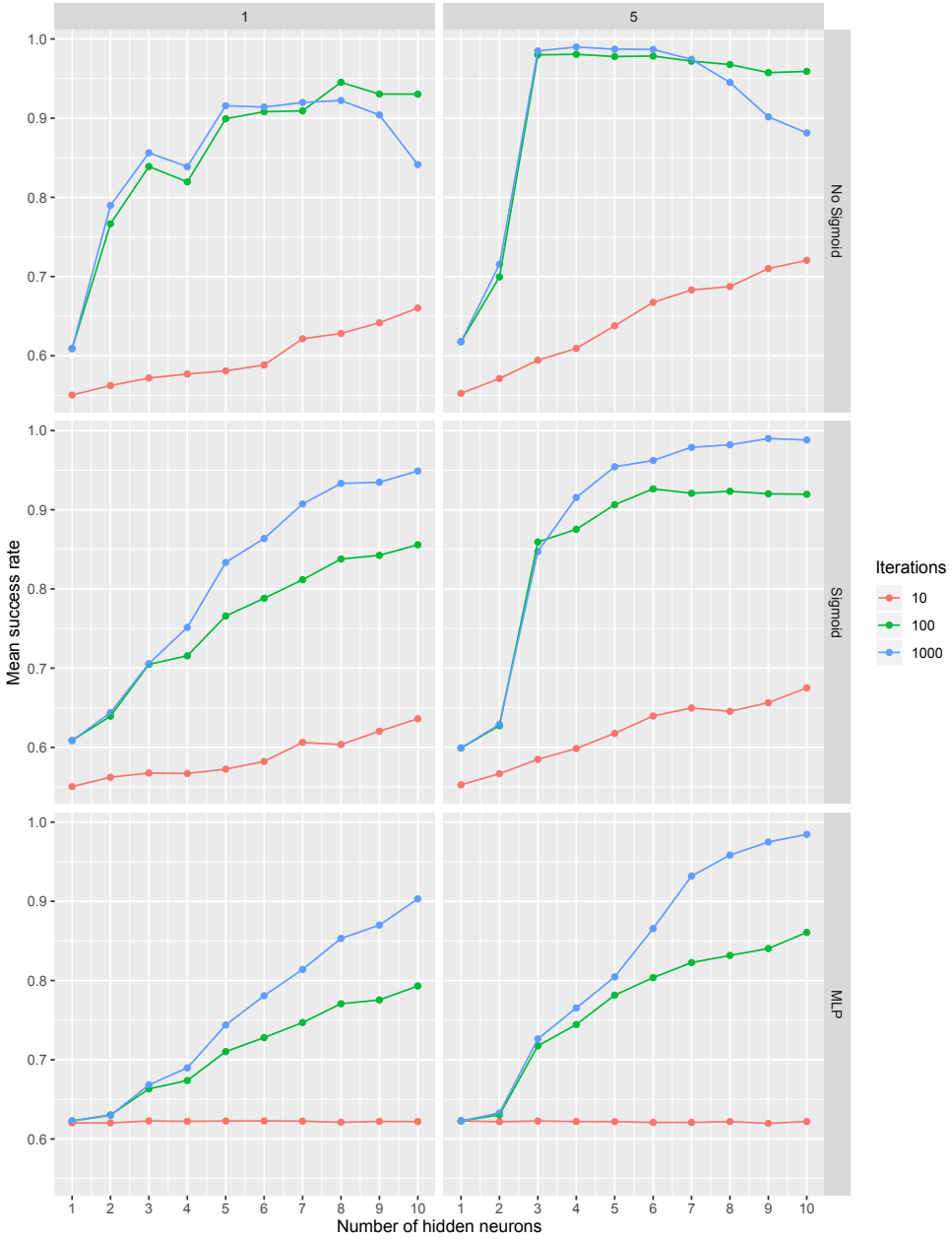

Fig. 4 Average success rate (over the 50 experimental runs) (y-axis) vs. number of hidden neurons (x-axis). This figure complements Figure 3.

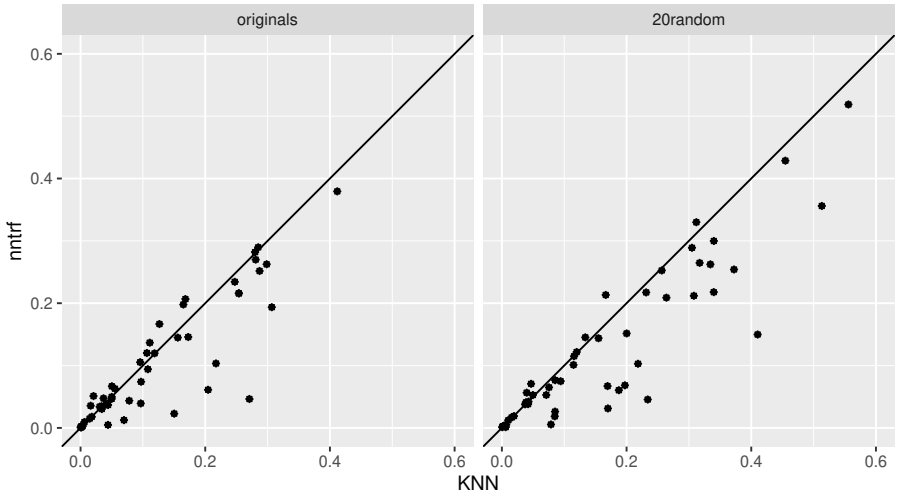

Fig. 5 Error rate of **nntrf** vs. KNN. Original domains (left) and extended domains with 20 random attributes (right).

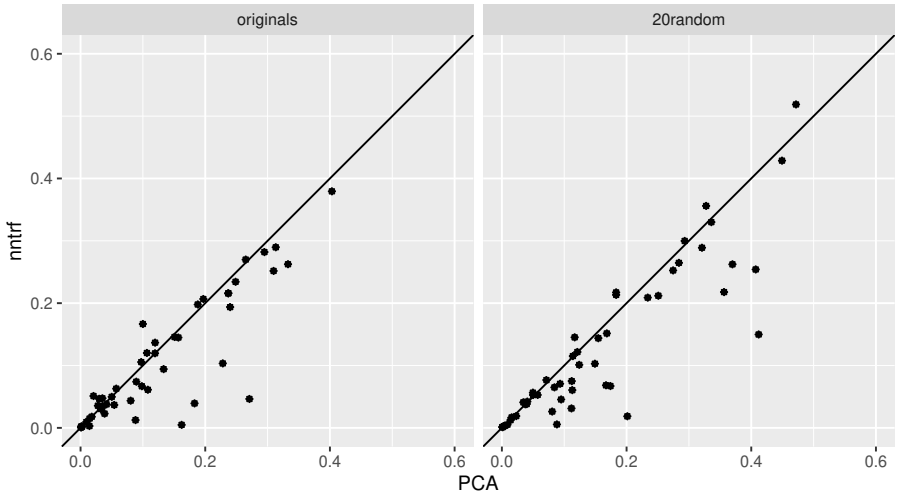

Fig. 6 Error rate of **nntrf** vs. PCA. Original domains (left) and extended domains with 20 random attributes (right).

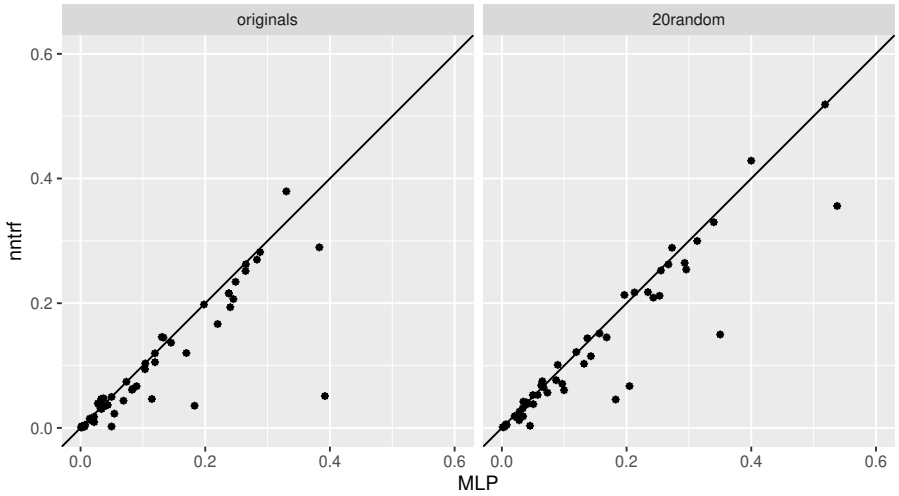

Fig. 7 Error rate of **nntrf** vs. MLP. Original domains (left) and extended domains (right) with 20 random attributes.