Applied Intelligence (2022) 52:4927–4939
https://doi.org/10.1007/s10489-021-02626-6

Tuning Database-Friendly Random Projection Matrices for Improved Distance Preservation on Specific Data

Daniel López-Sánchez[1] · Cyril de Bodt[2] · John A. Lee[3] · Angélica González Arrieta[1] · Juan M. Corchado[1]

Accepted: 17 June 2021 / Published online: 30 July 2021

Abstract
Random Projection is one of the most popular and successful dimensionality reduction algorithms for large volumes of data. However, given its stochastic nature, different initializations of the projection matrix can lead to very different levels of performance. This paper presents a guided random search algorithm to mitigate this problem. The proposed method uses a small number of training data samples to iteratively adjust a projection matrix, improving its performance on similarly distributed data. Experimental results show that projection matrices generated with the proposed method result in a better preservation of distances between data samples. Conveniently, this is achieved while preserving the database-friendliness of the projection matrix, as it remains sparse and comprised exclusively of integers after being tuned with our algorithm. Moreover, running the proposed algorithm on a consumer-grade CPU requires only a few seconds.

Keywords Random projection · Nearest neighbor search · Neighborhood preservation · Dimensionality reduction · Randomized algorithms

1 Introduction

The recent advances and democratization of technology have come along with an unprecedented volume of high-dimensional data. In this context, the search for effective but at the same time efficient methods of exploiting this ever-increasing volume of information has attracted a lot of attention among researchers in the field of machine learning. Among the techniques developed to deal with these previously unmanageable volumes of data, Random Projection (RP) [1] is arguably one of the most popular tools.

In essence, Random Projection is an extremely simple linear dimensionality reduction method. Just like any other linear dimensionality reduction algorithm, Random Projection reduces the dimension of samples by applying a linear transformation to input data, so that each output feature is computed as a linear combination of the original features. However, the main difference between Random Projection and other approaches is that it generates the projection matrix from a random distribution. Therefore, as opposed to other methods where training data is required to select an appropriate projection matrix, Random Projection is a data-independent method. This means that no knowledge about the distribution of data is required to generate the projection matrix. Surprisingly, if an appropriate distribution is used to generate the entries of the projection matrix, the structure of data in the high-dimensional input feature space can be mostly preserved after the projection. Moreover, the projection matrix can be sparse and made of integers, allowing for computational savings and efficient implementation in database environments [2, 3].

Thanks to this property, Random Projection has become a widespread tool for dimensionality reduction, especially in large-scale applications where the volume of data or the dimensionality of samples is too big for alternative methods. For instance, Random Projection has been successfully used to accelerate tasks such as multivariate correlation analysis [4], high-dimensional data clustering

Cyril de Bodt is a post-doctoral fellow of the Belgian American Educational Foundation. John A. Lee is a Senior Research Associate with the Belgian F.R.S.-FNRS.

✉ Daniel López-Sánchez
lope@usal.es

1 BISITE Research Group, University of Salamanca, Salamanca, Spain

2 MIT Media Lab, Massachusetts Institute of Technology, Cambridge, USA

3 IREC Institute & ICTEAM Institute, UCLouvain, Brussels, Belgium

[5, 6], image search [7] or texture classification [8], among many others.

Despite its success, the random nature of this algorithm is also the cause of its major disadvantage. Since the entries of the random projection matrix are chosen at random, different initializations of the projection matrix often yield different results. In some cases, the differences between worst and best case projection matrices are significant. In the literature, some preliminary work has been done on the use of genetic algorithms to generate a pool of projection matrices with the goal of evolving the one that is best suited to the dataset under consideration [9]. However, this approach is highly inefficient both in terms of memory and computation, as it involves storing numerous projection matrices simultaneously in memory.

In this paper, we propose a new guided random search algorithm specially designed to tune sparse RP matrices to improve their performances on specific data. The proposed method works under the assumption that, in most application scenarios, at least a small number of data samples are available when initializing the projection matrix. Therefore, in such cases, it might be worth sacrificing the data-independent nature of Random Projection to generate a projection matrix whose performance is optimal on similarly distributed data. Conveniently, this algorithm preserves other beneficial features of Random Projection matrices, such as their sparsity, which can lead to important memory and computational savings. The experimental results show that (1) the proposed algorithm noticeably increases the performance of RP matrices on specific data according to neighborhood preservation quality metrics, (2) as little as fifty samples are enough to obtain statistically significant improvements and (3) training times are in the order of seconds.

2 Related work

The modern Random Projection algorithm has its roots in the Johnson-Lindestrauss (JL) lemma [10], which states that a small set of points in a high-dimensional space can be embedded into a space of much lower dimension in such a way that squared Euclidean distances between the points are nearly preserved. Formally, for any $0 < \epsilon < 1$ and $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$, there is a map $f : \mathbb{R}^d \to \mathbb{R}^k$ for $k = \mathcal{O}(\epsilon^{-2}\log(n))$ such that, for all $i, j \in \{1, \ldots, n\}, i \neq j$,

$$\begin{aligned}(1-\epsilon)||\mathbf{x}_i - \mathbf{x}_j||^2 &\leq ||f(\mathbf{x}_i) - f(\mathbf{x}_j)||^2 \\ &\leq (1+\epsilon)||\mathbf{x}_i - \mathbf{x}_j||^2.\end{aligned} \tag{1}$$

In addition, this map can be found in randomized polynomial time [11].

In practice, the map f consists in multiplying the d-dimensional data samples by a $d \times k$ projection matrix with entries chosen at random from a suitable distribution, and applying a $1/\sqrt{k}$ scaling factor[1] to compensate for the reduction in the dimensionality [2]. Once the $d \times k$ projection matrix R has been populated, an arbitrary set of n points represented as an $n \times d$ matrix X can be projected from $\mathbb{R}^d$ to $\mathbb{R}^k$ as follows:

$$X' = \frac{1}{\sqrt{k}} XR\ , \text{ where } X' \in \mathbb{R}^{n \times k}. \tag{2}$$

Over the years, different distributions for the entries of the projection matrix were proven to fulfill the JL-lemma. In the early days, the standard normal distribution was used [13]. Later on, studies demonstrated that the projection matrix can be drawn from many other distributions. For instance, in [14], the authors showed that a result, that is analogous to the JL-lemma, can be proven for any zero-mean distribution with subgaussian tail. Another well-known study by D. Achlioptas showed that the entries of the projection matrix could instead be drawn from a sparse and much simpler distribution [15]. In particular, Achlioptas' work proved that if the entries of the projection matrix are drawn from the distribution defined by (3) with sparsity term $s = 1$ or $s = 3$; then the JL-lemma will be satisfied.

$$r_{ij} = \sqrt{s} \times \begin{cases} 1 & \text{with probability} \quad 1/2s, \\ 0 & \text{with probability} \quad 1 - 1/s, \\ -1 & \text{with probability} \quad 1/2s. \end{cases} \tag{3}$$

Intuitively, when $s = 1$, each entry of the projection matrix is randomly set to $+1$ or -1 with equal probability. Similarly, when $s = 3$, roughly two-thirds of the entries of the projection matrix are set to zero, and the remaining entries are set to $\sqrt{3}$ or $-\sqrt{3}$ with equal probability. Conveniently, using the distribution proposed by Achlioptas reduces the computational cost of the projection. If the multiplication by $\sqrt{s}$ present in (3) is delayed, the computation of the projection reduces to aggregate evaluation (i.e. summation and subtraction but no multiplication), which can be efficiently performed in database environments using standard SQL primitives. In

[1] In the early versions of Random Projection, the linear map from $\mathbb{R}^d$ to $\mathbb{R}^k$ was chosen as the orthogonal projection on a random k-dimensional subspace of $\mathbb{R}^d$. With this formulation, the appropriate scaling factor was proven to be $\sqrt{d/k}$ [10, 12]. More recent variations of the algorithm populate the projection matrix with entries independently drawn from a suitable distribution, such as the standard normal [13] or a sparse distribution [2]. In those cases, an additional scaling factor of $1/\sqrt{d}$ is typically added to ensure that the projection vectors (the columns of the projection matrix) are close to unit length. When combined, the $\sqrt{d/k}$ and $1/\sqrt{d}$ factors result in the simplified $1/\sqrt{k}$ scaling factor present in most modern formulations of the Random Projection algorithm.

addition, the sparsity term s enables further storage and computational savings. For instance, when using $s = 3$, only $\frac{1}{3}$ of the entries of the projection matrix are nonzero. Moreover, it has been suggested that it is possible to use greater sparsity levels in (3) with little loss in the preservation of distances. In particular, some studies recommend using $s = \sqrt{d}$ [3], which potentially reduces computing times by a $1/\sqrt{d}$ factor.

For most distributions of the projection matrix, the proof of the JL-lemma follows a similar line of reasoning. First, it is shown that the squared length of an arbitrary vector is preserved in expectation after the projection. The proof goes further, showing that the squared length has a low variance after the projection, and that therefore there is a high probability that the squared length of the vector will not get distorted by more than $(1 \pm \epsilon)$ after the projection. Then, the trivial union bound guarantees, for $k = \mathcal{O}(\epsilon^{-2}\log(n))$, that the probability the projection matrix produces a relative distortion greater than $(1 \pm \epsilon)$ for any pair among the n samples, is lower than $1 - 1/n$. Therefore, by generating and evaluating $\mathcal{O}(n)$ projection matrices, the probability of success can be raised to the desired constant, leading to the claimed randomized polynomial time [11].

Despite these results, in practice, the usual approach is to generate just one projection matrix, trusting that the average-case matrix will perform well enough and avoiding the need for any training data. This, as mentioned before, causes the performance of Random Projection to be inconveniently variable among different instantiations of the projection matrix. In turn, the proposed algorithm is motivated by the informal observation that, in most scenarios, at least a small number of data samples are available at the moment of initializing the projection matrix. Therefore, if the performance of a given projection matrix can be measured in terms of how well it preserves the pairwise distances of the available data samples, it becomes possible to tune it to work better than the average random matrix on similarly distributed test samples.

2.1 Random projection variants

Over the years, the scope of applications of the RP algorithm has greatly expanded, and a wide range of specialized versions of the algorithm have been developed to fit the needs of different domains. While some variants simply modify the distribution from which the entries of the projection matrix are drawn, others modify the algorithm at a more fundamental level. Nevertheless, most variants fall into one of the following categories:

- **Quantized Random Projections**: This line of research studies the possibility of performing quantization subsequent to the projection in order to achieve additional data compression [16]. Proposals include single-bit [17] as well as multiple-bit quantization [18].
- **Sparse Random Projections**: Inspired by the database-friendly random projection matrices of [2], several authors have explored the use of sparse [3, 19, 20] and binary [21] random projection matrices.
- **Non-linear Random Projections**: RP-based techniques have been used to capture non-linear features in a compact representation. Approaches range from RP-based preprocessing for existing non-linear dimensionality reduction methods [22] to *ad-hoc* variants for non-linear kernel functions [5, 23].
- **Structured Johnson-Lindestrauss**: Following the work of [24], structured JL methods try to approximate the result of a traditional RP by decomposing the projection matrix into a set of low-memory matrices [25, 26].

In terms of applications, variants of the RP algorithm have been successfully applied to address some of the most important challenges of big data systems, including privacy protection [27, 28], handling of high-dimensional data [6, 29], and system scalability [7, 30, 31], among many others.

3 Proposed algorithm

Essentially, the proposed algorithm aims to find a sparse Random Projection matrix that is optimized to preserve pairwise distances among data samples following a specific distribution. To achieve this, the assumption is made that, in most application scenarios, at least a small number of data samples are available at the moment of selecting the projection matrix. These few data samples are used to iteratively tune a randomly initialized projection matrix, with the aim of minimizing a specially designed loss function which measures the relative distortion induced by the projection in pairwise distances.

Formally, let $X = [\mathbf{x}_1, \cdots, \mathbf{x}_n]^\top$ be the $n \times d$ matrix-representation of the n data points that are available at the moment of initializing the projection matrix. Also, let $R = [\mathbf{r}_1, \mathbf{r}_2, \cdots, \mathbf{r}_k]$ be a projection matrix formed by the concatenation of k projection directions of the form $\mathbf{r}_i \in \{-1, 0, 1\}^{d\times 1}$. Then, an individual data sample can be projected using the map $f : \mathbb{R}^d \to \mathbb{R}^k$ defined as

$$f(\mathbf{x}) = \frac{\sqrt{s}}{\sqrt{k}}\mathbf{x}R. \qquad (4)$$

The loss function used by the proposed algorithm emerges naturally from the JL-lemma. As discussed before, this lemma states that, for a sufficiently large output dimension, the relative distortion induced by the projection in the pairwise distances among a set of points is lower than $(1 \pm \epsilon)$ with high probability. This bound is expressed by the inequality in (1), which compares distances between data samples before and after the projection, explicitly introducing ϵ as the relative distortion bound.

In order to derive an algorithm which can optimize a projection matrix to perform better on samples from a given dataset, we begin by defining a loss function that, similarly to the JL-lemma, considers the degree to which distances between data samples are preserved after the projection. Particularly, the selected loss function computes the empirical average distortion induced on a concrete set of n samples by a specific projection matrix. To this end, the loss iterates over the $\binom{n}{2}$ possible pairs of points, computing the empirical relative distortion as the absolute difference of the distances before and after the projection, divided by the distance before the projection. Formally, the loss function is defined as

$$\mathcal{L}(R) = \frac{1}{\binom{n}{2}} \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{\left| ||\mathbf{x}_i - \mathbf{x}_j||^2 - ||f(\mathbf{x}_i) - f(\mathbf{x}_j)||^2 \right|}{||\mathbf{x}_i - \mathbf{x}_j||^2}. \tag{5}$$

Conveniently, this loss can be easily interpreted. For instance, a loss of $\mathcal{L}(R) = 0.12$ for a projection matrix R on a given set of n samples would indicate that, on average, pairwise distances between samples suffered a 12% distortion (extension or contraction) relative to their values before the projection.

However, note that the naive evaluation of this loss function would be prohibitive from a computational point of view. First, we can observe that it involves iterating over the $\binom{n}{2}$ pairwise distances. As detailed below, this can be palliated, given that only a small number of training samples are needed to achieve a notable improvement in the performance of a random projection matrix. However, computing the loss function also involves re-projecting all data samples and re-calculating the pairwise distances with the evaluated projection matrix. We will therefore introduce some ic tricks in this section to mitigate this problem.

Now that we have defined an optimization goal, the next step is to decide on the optimization we will use to minimize this loss function. While traditional optimization s such as gradient descent could in principle be applied, this approach would lose one of the major advantages of Random Projection, as we would be sacrificing the database-friendliness [15] granted by the sparsity of the projection matrix and the fact that its entries are constrained to lie in $\{-1, 0, 1\}$. Therefore, the goal is to find a random projection matrix that minimizes the loss function for a specific set of data samples, while preserving its sparsity and integer nature. To achieve this, we introduce a guided random search similar in nature to the family of simulated annealing methods [32][2]. Essentially, the proposed iteratively evaluates random alterations of the current projection matrix, and keeps them only if they reduce the loss function. Specifically, the consists of the following steps:

1. Initialize the $d \times k$ projection matrix $R = [\mathbf{r}_1, \mathbf{r}_2, \cdots, \mathbf{r}_k]$ following Achlioptas' distribution (3) and compute $\mathcal{L}(R)$ on the available set of n samples.
2. Generate a random projection direction $\mathbf{w} \in \{1, 0, -1\}^{d\times 1}$ following Achlioptas' distribution (3) and a random integer c from the discrete uniform distribution $\mathcal{U}(1, k)$.
3. Generate a tentative improved random projection matrix by replacing the c-th column of R with $\mathbf{w}$. Formally, the tentative random projection matrix is $R^\star = [\mathbf{r}_1, \cdots, \mathbf{r}_{c-1}, \mathbf{w}, \mathbf{r}_{c+1}, \cdots, \mathbf{r}_k]$.
4. Compute $\mathcal{L}(R^\star)$. Then, if $\mathcal{L}(R^\star) < \mathcal{L}(R)$, take $R^\star$ as the newly selected projection matrix (i.e., $R \leftarrow R^\star$).
5. Repeat steps 2-4 a desired number of times, as specified by the hyper-parameter N_{iter}

While this formulation of the is easy to understand and to implement, it incurs in several inefficiencies that severely limit its applicability. Particularly, naively computing $\mathcal{L}(R^\star)$ at each iteration results in a cost of $\mathcal{O}(ndk + n^2d + n^2k)$ per iteration, where the $\mathcal{O}(ndk)$ term comes from computing the projection of samples with the tentative projection matrix (i.e., $\frac{\sqrt{s}}{\sqrt{k}}XR^\star$), and the $\mathcal{O}(n^2d + n^2k)$ is for the computation of the pairwise distances among the d-dimensional (input) and k-dimensional (output) data samples.

To mitigate this problem, the can be re-formulated to yield the same results while avoiding unnecessary computations. First, we observe that the computation of $\frac{\sqrt{s}}{\sqrt{k}}XR^\star$ is highly redundant with $\frac{\sqrt{s}}{\sqrt{k}}XR$, where R is the best matrix found so far by the . This is because R and $R^\star$ only differ in the c-th column. As a consequence, if $\frac{\sqrt{s}}{\sqrt{k}}XR$ is stored, the computation of $\frac{\sqrt{s}}{\sqrt{k}}XR^\star$ at each iteration can be done in $\mathcal{O}(nd)$ time. With this modification, each iteration of the takes $\mathcal{O}(nd + n^2d + n^2k)$. However, the complexity can be further simplified by analyzing the computation of the pairwise distances.

[2]Simulated annealing s differ from the proposed method in that these often include a simulated temperature variable which encourages the exploration of non-improving solutions to avoid getting stuck in local minima. However, preliminary experiments showed that using this approach only slowed down the convergence of the without improving the performance.

Let us define the function $D : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times n}$ which computes the matrix of pairwise squared Euclidean distances for a data matrix as

$$D(X) = \begin{bmatrix} ||\mathbf{x}_1 - \mathbf{x}_1||^2 & \cdots & ||\mathbf{x}_1 - \mathbf{x}_n||^2 \\ \vdots & \ddots & \vdots \\ ||\mathbf{x}_n - \mathbf{x}_1||^2 & \cdots & ||\mathbf{x}_n - \mathbf{x}_n||^2 \end{bmatrix}, \tag{6}$$

where $X = [\mathbf{x}_1, \cdots, \mathbf{x}_n]^\top$.

Intuitively, $D(X)_{ij}$ corresponds to the squared Euclidean distance between the i-th and the j-th samples in X. Then, the error function defined in (5) for a tentative projection matrix $R^\star$ can be expressed as

$$\mathcal{L}(R^\star) = \frac{1}{\binom{n}{2}} \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{\left|D(X)_{ij} - D(\frac{\sqrt{s}}{\sqrt{k}} X R^\star)_{ij}\right|}{D(X)_{ij}}. \tag{7}$$

At this point, we see that the computation of $\mathcal{L}(R^\star)$ at each iteration can be easily accelerated by pre-computing and storing $D(X)$, which will not change throughout the iterations of the . This further reduces the cost of each iteration from $\mathcal{O}(nd + n^2d + n^2k)$ to $\mathcal{O}(nd + n^2k)$. Now, the single most expensive computation at each iteration is the computation using $D(\cdot)$ of the pairwise distance matrix for the data samples projected by $R^\star$, which takes $\mathcal{O}(n^2k)$. To accelerate this step, we will rely on a helpful property of squared Euclidean distances. Consider two arbitrary data samples

$$\begin{aligned} \mathbf{x} &= [x_1, x_2, \cdots, x_i, \cdots, x_d], \\ \mathbf{y} &= [y_1, y_2, \cdots, y_i, \cdots, y_d]. \end{aligned} \tag{8}$$

Then, perform an arbitrary modification in one feature of each sample, yielding

$$\begin{aligned} \mathbf{x}^\star &= [x_1, x_2, \cdots, x_i', \cdots, x_d], \\ \mathbf{y}^\star &= [y_1, y_2, \cdots, y_i', \cdots, y_d]. \end{aligned} \tag{9}$$

Conveniently, if the squared Euclidean distance between the original samples is known, the squared Euclidean distance between $\mathbf{x}^\star$ and $\mathbf{y}^\star$ can be computed in a time independent of their dimensionality using:

$$||\mathbf{x}^\star - \mathbf{y}^\star||^2 = ||\mathbf{x} - \mathbf{y}||^2 - (x_i - y_i)^2 + (x_i' - y_i')^2, \tag{10}$$

since the total squared Euclidean distance between two samples can be expressed as the sum of the squared differences between individual features. As a consequence, when evaluating $\mathcal{L}(R^\star)$ with (7), the computation of $D(\frac{\sqrt{s}}{\sqrt{k}} X R^\star)$ can be done using the following formula[3]

$$\begin{aligned} D(\tfrac{\sqrt{s}}{\sqrt{k}} X R^\star) = D(\tfrac{\sqrt{s}}{\sqrt{k}} X R) & - D(\tfrac{\sqrt{s}}{\sqrt{k}} X R_{[:,\,c]}) \\ & + D(\tfrac{\sqrt{s}}{\sqrt{k}} X R^\star_{[:,\,c]}), \end{aligned} \tag{11}$$

[3]In this context, the notation $R_{[:,\,c]}$ denotes the c-th column of R, considering it as a $d \times 1$ column vector.

where, as mentioned before, R is the best projection matrix found so far, which only differs from $R^\star$ in the values of the c-th column. Note that, conveniently, applying $D(\cdot)$ to $XR_{[:,\,c]}$ and $XR^\star_{[:,\,c]}$ takes only $\mathcal{O}(n^2)$ since they are of shape $n \times 1$. Also, note that $R^\star_{[:,\,c]} = \mathbf{w}$. Therefore, provided that the projection of data samples and the pairwise distance matrix corresponding to the best projection matrix found so far (i.e., $\frac{\sqrt{s}}{\sqrt{k}} XR$ and $D(\frac{\sqrt{s}}{\sqrt{k}} XR)$) are stored and updated at each iteration, $D(\frac{\sqrt{s}}{\sqrt{k}} XR^\star)$ can always be computed in $\mathcal{O}(nd + n^2)$ by applying (11). This final modification reduces the complexity of each iteration from $\mathcal{O}(nd + n^2k)$ to $\mathcal{O}(nd + n^2)$.

Algorithm 1 provides a self-contained description of the proposed with the implementation details described above. The final computational complexity is as follows: steps 1-5 of 1, which are executed only once, have a complexity of $\mathcal{O}(ndk + n^2(d + k))$. Then, steps 6-14 which are repeated iteratively N_{iter} times have a complexity of $\mathcal{O}(nd + n^2)$. The complete complexity of the is hence $\mathcal{O}(ndk + n^2(d + k) + N_{iter}(nd + n^2))$.

Regarding the parametrization of Algorithm 1, the proposed has three hyper-parameters, two of which are inherited from the standard Random Projection method and have well known semantics and effects:

- Output dimension of the projection matrix (k): Determines the output dimension of the projected samples. Using a larger k results in a better performance, at the cost of having a bigger projection matrix and a larger dimension of output data samples. Tables 2 and 3 present results for a wide range of values of k, showing that the improvements in performance obtained by using Algorithm 1 are consistent regardless of the selected output dimensionality.
- Sparsity level (s): Determines the fraction of zero-valued entries in the projection matrix. Can be used to reduce the storage and computational costs while preserving most of the performance. This hyper-parameter was introduced in [2], and further studied in [3]. Following the recommendation of [3], we use $s = \sqrt{d}$ in all the experiments presented in Section 4, where d is the dimension of input data samples.
- Number of iterations (N_{iter}): Determines the number of iterations to be run by the optimization . In essence, using more iterations will result in a better performance, at the cost of longer execution times. The experimental results presented in Section 4 suggest that the proposed is fairly robust regarding the selection of this hyperparameter, as using 3×10^3 or 4×10^3 resulted in a good balance between performance and efficiency in all the experiments.

Algorithm 1 Data-tuned random projection (DT-P).

```
Require: The initial $d \times k$ projection matrix $R$, a $n \times d$ matrix $X$ containing the $n$ available data samples, the number of iterations to run, which is determined by
Ensure: Returns an optimized sparse projection matrix $R \in \{0, 1, -1\}^{d \times k}$ which preserves distances more accurately than the initial projection matrix on data distributed
 1: $D_d \leftarrow D(X)$    ▷ Compute the pairwise distance matrix of input data
 2: $X' \leftarrow \frac{\sqrt{s}}{\sqrt{k}} X R$    ▷ Project data using the current projection matrix
 3: $D_k \leftarrow D(X')$    ▷ Compute the pairwise distance matrix of the projected data
 4: $D_k^\star \leftarrow \{0\}^{n \times n}$    ▷ Initialize a variable to hold tentative distance matrices
 5: $\mathcal{L} \leftarrow \frac{1}{\binom{n}{2}} \sum_{i=1}^{n} \sum_{j=i+1}^{n} \left| D_d[i,j] - D_k[i,j] \right| / D_d[i,j]$    ▷ Compute loss for the current projection matrix
 6: for $counter = 1, \cdots, N_{iter}$ do
 7:     Generate a $d \times 1$ vector $\mathbf{w}$ with entries drawn from $\{-1, 0, 1\}$ w.p. $\frac{1}{2s}$, $1 - \frac{1}{s}$ and $\frac{1}{2s}$ respectively    ▷ Generate new projection direction
 8:     Generate a random integer $c \sim \mathcal{U}(1, k)$    ▷ Select which column of R will be substituted
 9:     $D_k^\star \leftarrow D_k - D(X'_{[:,c]}) + D(\frac{\sqrt{s}}{\sqrt{k}} X \mathbf{w})$    ▷ Apply (11) to compute the tentative distance matrix
10:     $\mathcal{L}^\star \leftarrow \frac{1}{\binom{n}{2}} \sum_{i=1}^{n} \sum_{j=i+1}^{n} \left| D_d[i,j] - D_k^\star[i,j] \right| / D_d[i,j]$    ▷ Loss for the tentative projection matrix
11:     if $\mathcal{L}^\star < \mathcal{L}$ then    ▷ If loss decreased...
12:         $\mathcal{L} \leftarrow \mathcal{L}^\star$    ▷ Update best loss variable
13:         $D_k \leftarrow D_k^\star$    ▷ Update current distance matrix
14:         $X'_{[:,c]} \leftarrow \frac{\sqrt{s}}{\sqrt{k}} X \mathbf{w}$    ▷ Update current projected representation of data
15:         $R_{[:,c]} \leftarrow \mathbf{w}$    ▷ Update current projection matrix
16: return R
```

4 Experimental results

In this section, we use different evaluation tools to measure the improvement in the performance of RP matrices obtained by means of Algorithm 1. For the experiments, we selected six public datasets from a wide range of domains. In particular, the data used in the experiments ranges from raw image data (Trevi), to audio features (Isolet), including hand-crafted image features (GIST). In this regard, the results presented in this section suggest that the proposed algorithm performs well on a wide range of data distributions. This is true likely because of the generality of Algorithm 1, which makes no assumption about the distribution of data, but instead evaluates different random variations of the projection matrix in an efficient manner. Table 1 describes their main features. Throughout this section, we will refer to the proposed algorithm as Data-Tuned Random Projection (DT-RP).

4.1 Recall on different datasets

To compare the performance of DT-RP to the standard Random Projection method, we first use the Recall measure, very common in the literature of approximate nearest neighbor search [40]. For a given query point, the Recall is the proportion of true K-nearest neighbors returned by an algorithm, in this case, K-NN in the projected version of the data. Formally, the recall[4] is computed as

$$\text{Recall} = \frac{\left|\nu \cap \nu'\right|}{K}, \tag{12}$$

where ν' is the set of K-approximate nearest neighbors, ν is the set containing the real K-nearest neighbors and $\left|\nu \cap \nu'\right|$ is the cardinality of the intersection of the two sets. In the experiments, we considered the 5-nearest neighbors ($K = 5$) to compute the Recall, which is a typical value, for instance, in distance-based classification applications.

The experimental protocol for all datasets was as follows: first, 500 samples were selected at random from the training set, and provided to DT-RP for training. To compute the Recall, 1000 samples were selected as the query points from the test set. The remaining points in the test set were used as the database to perform the query. Each algorithm was evaluated 500 times to study the stochastic nature of both RP and DT-RP. Figure 1 shows the results of this experiment as a pair of Box-plots for each dataset. In this case, the output dimension was fixed at $k = 200$.

As we can observe, the proposed DT-RP algorithm outperforms RP for all datasets. The average Recall

[4]Note that the recall metric used here is different from the common classification recall (sensitivity, true positive rate) commonly used in categorization problems together with the precision score. For more details about the recall score used in approximate nearest neighbor search see [40].

Table 1 Summary of the six public datasets used in the experiments

Dataset	Reference	Description	Classes	Feature number	Test set size
MNIST	[33]	28×28 Images of handwritten digits	10	784	10,000
ISOLET	[34]	Spoken words (audio features)	26	617	3,898
CIFAR10	[35]	32×32 Color images of objects	10	3072	10,000
GIST (subset)	[36]	Global image descriptors (GIST)	N/A	960	50,000
Trevi	[37]	64×64 Gray-scale image patches	N/A	4096	50,450
FMA	[38]	Song's features extracted with librosa [39]	161	518	53,287

For Isolet, Trevi and FMA datasets, a random train/test split was arranged with a 50%/50% proportion. A similar random split was used on a subset of the GIST dataset with 100,000 samples

obtained with DT-RP was in all cases superior to the best score obtained by any instance of RP. Furthermore, the worst Recall obtained by DT-RP for four of the six datasets, was very close to and in some cases above the best score obtained by RP in the 500 runs. In addition, when comparing RP to DT-RP, a reduction in the standard deviation of Recall scores among runs was registered for all datasets. These reductions ranged from $\sim$11% for GIST to $\sim$62% for FMA.

To further assess the performance of the proposed method, we ran a number of experiments following the same protocol, but with different output dimension values. In this case, 50 runs of each algorithm were executed. The resulting Recall values and standard deviations are provided in Table 2. This table also displays the approximate running time for each algorithm[5], computed as the lowest running time out of ten executions on an Intel i7-6700K CPU with 16GB of RAM memory.

Analyzing Table 2, the proposed algorithm achieves consistent improvements across the different datasets and output dimensions. We can also observe that the running time for DT-RP lies in the 4 to 5 seconds range for all datasets and output dimensions. As mentioned above, the computational complexity of DT-RP includes a $\mathcal{O}(N_{iter} \cdot n^2)$ term, which dominated the computation time in this case. This explains why the times were similar for all datasets, since n and N_{iter} were not changed throughout the experiments. Of course, standard RP is several orders of magnitude faster, as it only requires drawing the $d \cdot k$ entries of the projection matrix from a discrete random distribution. Nevertheless, since in many application scenarios the projection matrix will remain in use for long periods of time after its initialization, spending these extra seconds on the initialization process is worthwhile in many cases.

[5]Note that the provided times correspond to the initialization/tuning of the projection matrices. Comparing test times is not necessary because once the projection matrix has been constructed, both standard RP and DT-RP are identical.

4.2 Comparison with an improved baseline

While the standard RP is the most obvious baseline for comparison, another natural reference method can be used to validate DT-RP. As detailed in Section 2, most JL-lemma proofs show that a randomly initialized projection matrix has, for a sufficiently large k, a probability greater than $1/n$ of succeeding in its low distance distortion goal. Therefore, a rarely employed in practice but natural approach would involve generating and evaluating a number of projection matrices, keeping the one that achieves the lowest average distortion for the available training samples. We shall refer to this approach as "Best of n RPs".

Table 3 compiles the results of the experiments, comparing DT-RP to the Best of n RPs approach. Again, we used the Recall measure for different output dimensions and datasets. Each experiment was executed 50 times, so the table reports the average and standard deviation of the results. In this case, we selected a more conservative value for the number N_{iter} of iterations of DT-RP. Namely, we used $N_{iter} = 3 \times 10^3$, to highlight how this would reduce execution times with little loss in performance. As expected, selecting the best RP matrix out of a number of initializations resulted in a slight increase in the accuracy as compared to the basic RP approach. However, this small improvement came along with vast increases in the execution time. In fact, while the Best of n RPs approach was outperformed by DT-RP all across the board, execution times of the former were longer in most cases. This suggests that the naive "Best of n RPs" approach is a much less efficient alternative for data-aware RP matrix selection than DT-RP.

4.3 Influence of the training set size

As previously mentioned, one of the advantages of DT-RP is that it can improve the performance of RP matrices even if only a small number of training samples are available. In all the experiments presented so far, DT-RP was provided with only 500 training samples, which is a small number

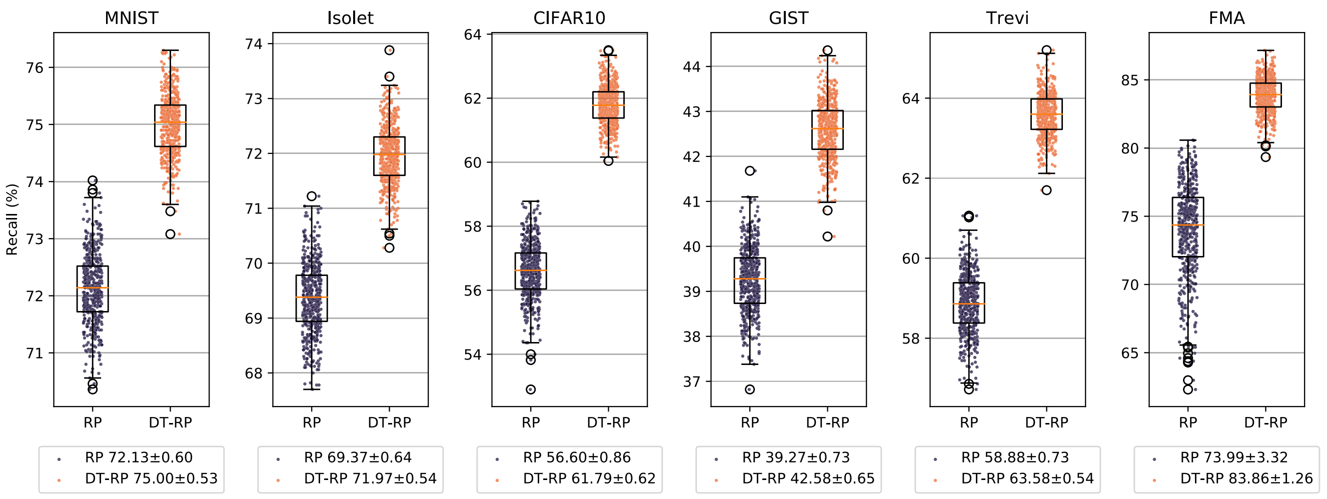

Fig. 1 Box-plots of Recall values for 500 runs of standard sparse Random Projection (RP) and the proposed algorithm (DT-RP). In all experiments, the sparsity degree was set at $s = \sqrt{d}$ and the output dimension at $k = 200$. Iteration number for DT-RP was set at $N_{iter} = 4 \times 10^3$, and 500 training samples were employed. Points have been spread along the horizontal axis to ease visualization. Below each plot, the average Recall is depicted for both RP and DT-RP

by today's standards. The ability to work with little training data explains why DT-RP's training time is in the order of seconds. In this subsection, we explore the limits of this feature by repeating the experiment in Fig. 1 with a

Table 2 Average ± standard deviation of the Recall values and training times for 50 runs of standard sparse Random Projection (RP) and the proposed algorithm (DT-RP)

Output dim.	MNIST		Isolet		CIFAR10	
	RP	DT-RP	RP	DT-RP	RP	DT-RP
k =25	33.51 ± 0.99	36.99 ± 0.72	31.15 ± 1.38	34.75 ± 1.09	14.15 ± 1.12	17.75 ± 0.60
	0.001s	3.959s	0.001s	3.932s	0.002s	4.240s
k =50	49.16 ± 0.76	53.10 ± 0.66	45.95 ± 1.18	49.36 ± 0.73	27.81 ± 1.01	32.78 ± 0.67
	0.002s	3.969s	0.002s	3.962s	0.003s	4.251s
k =100	62.24 ± 0.66	65.60 ± 0.62	58.78 ± 0.67	62.04 ± 0.64	42.80 ± 0.94	48.31 ± 0.66
	0.003s	4.001s	0.003s	3.937s	0.006s	4.294s
k =200	72.11 ± 0.54	74.91 ± 0.49	69.33 ± 0.61	72.03 ± 0.57	56.42 ± 0.93	61.67 ± 0.61
	0.007s	4.006s	0.006s	3.960s	0.011s	4.268s
k =400	79.41 ± 0.38	81.95 ± 0.40	77.12 ± 0.51	79.48 ± 0.32	67.55 ± 0.63	72.08 ± 0.49
	0.013s	4.064s	0.012s	3.987s	0.023s	4.354s
Output dim.	GIST		Trevi		FMA	
	RP	DT-RP	RP	DT-RP	RP	DT-RP
k =25	7.98 ± 0.40	9.00 ± 0.38	17.42 ± 0.83	20.33 ± 0.67	29.84 ± 11.05	66.03 ± 7.27
	0.001s	3.986s	0.002s	4.372s	0.001s	3.933s
k =50	15.39 ± 0.56	17.36 ± 0.47	30.90 ± 0.81	35.54 ± 0.75	50.15 ± 9.11	69.93 ± 4.54
	0.002s	4.014s	0.003s	4.402s	0.002s	3.944s
k =100	26.03 ± 0.85	29.07 ± 0.62	45.67 ± 0.84	50.55 ± 0.71	63.07 ± 6.49	77.75 ± 2.34
	0.004s	4.046s	0.006s	4.408s	0.003s	3.953s
k =200	39.32 ± 0.73	42.42 ± 0.67	58.76 ± 0.65	63.64 ± 0.56	73.97 ± 3.84	83.95 ± 1.32
	0.007s	4.028s	0.013s	4.439s	0.006s	3.981s
k =400	52.36 ± 0.67	55.64 ± 0.64	69.60 ± 0.64	73.52 ± 0.51	81.49 ± 1.98	88.25 ± 0.81
	0.014s	4.081s	0.025s	4.472s	0.011s	3.996s

In all experiments, the sparsity degree was set at $s = \sqrt{d}$. Iteration number for DT-RP was set at $N_{iter} = 4 \times 10^3$, and 500 training samples were provided

Table 3 Average ± standard deviation of the Recall values and training times for 50 runs of the "Best of n RPs" approach and the proposed algorithm (DT-RP)

Output dim.	MNIST		Isolet		CIFAR10	
	Best of n RP	DT-RP	Best of n RP	DT-RP	Best of n RP	DT-RP
k =200	72.61 ± 0.44	74.99 ± 0.57	69.69 ± 0.48	71.81 ± 0.57	57.60 ± 0.56	61.56 ± 0.57
	$3.478s$	$3.034s$	$3.045s$	$3.018s$	$7.933s$	$3.280s$
k =400	79.70 ± 0.38	81.72 ± 0.35	77.44 ± 0.44	79.34 ± 0.41	68.54 ± 0.61	71.80 ± 0.45
	$5.437s$	$3.056s$	$4.762s$	$3.014s$	$13.993s$	$3.292s$
Output dim.	GIST		Trevi		FMA	
	Best of n RP	DT-RP	Best of n RP	DT-RP	Best of n RP	DT-RP
k =200	39.77 ± 0.62	42.68 ± 0.60	59.90 ± 0.65	63.50 ± 0.61	78.58 ± 2.23	83.32 ± 1.10
	$3.826s$	$3.052s$	$9.900s$	$3.375s$	$2.875s$	$3.005s$
k =400	52.82 ± 0.64	55.56 ± 0.59	69.99 ± 0.53	73.33 ± 0.38	84.40 ± 1.20	87.99 ± 0.87
	$6.054s$	$3.080s$	$17.825s$	$3.426s$	$4.397s$	$3.040s$

In all experiments, the sparsity degree was set at $s = \sqrt{d}$. Iteration number for DT-RP was set at $N_{iter} = 3 \times 10^3$, and 500 training samples were provided to both algorithms

decreasing number of training samples. Specifically, DT-RP was executed 300 times, with only 500, 400, 300, 200, 100 and 50 training samples, respectively. The resulting Recalls for MNIST, Isolet and CIFAR10 datasets are depicted in Fig. 2.

DT-RP's performance continues to be superior to that of standard RP even for the instances of DT-RP trained with the least training samples. The results for the remaining datasets listed in Table 1 are similar and are not shown due to space limitations.

The 500 runs of RP and DT-RP for each training set size make it possible to compute statistical tests and assess whether the average performances of RP and DT-RP are significantly different according to the Recall measure. Two-tailed unpaired t-tests are employed if the 500 values for RP and for each training set size of DT-RP can be considered as being normally distributed according to D'Agostino and Pearson's omnibus normality test [41]. Otherwise, Mann-Whitney U tests are employed. The significance level was set at $\alpha = 0.01$. Since comparing the performances of RP and DT-RP for each training set size involves multiple hypothesis tests, the Holm-Bonferroni correction has been applied to bound by α the probability of considering at least one non-significant difference as statistically significant [42]. As suggested by Fig. 2, the dominance of DT-RP over RP is statistically significant

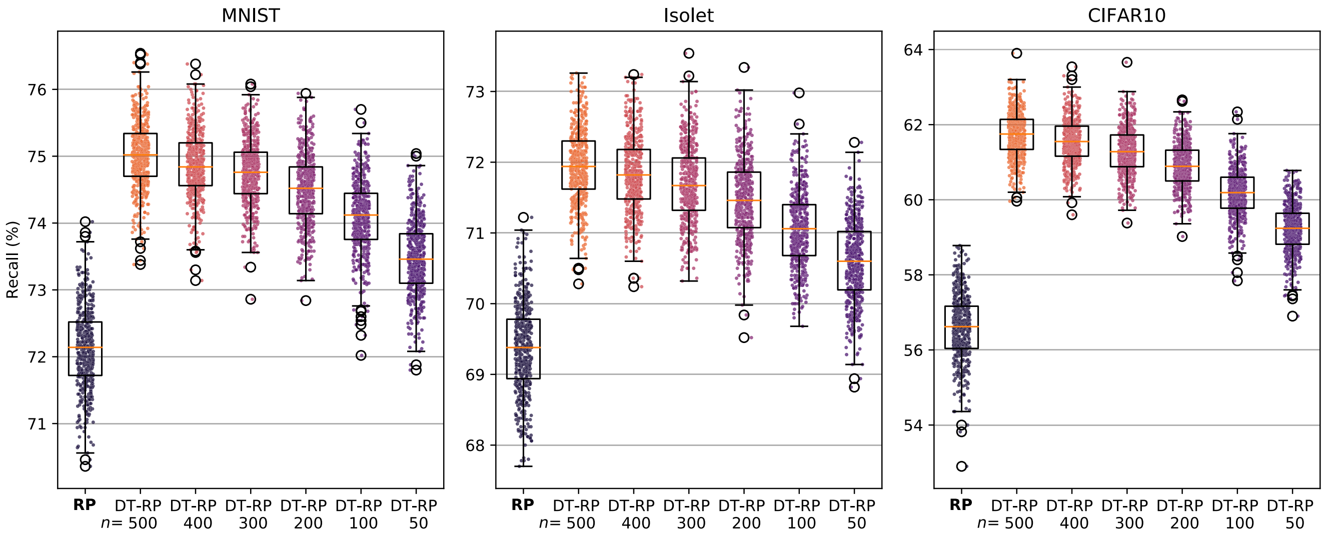

Fig. 2 Box-plots of Recall values for 500 runs of standard sparse Random Projection (RP) and of the proposed algorithm (DT-RP) for different numbers n of training samples. In all experiments, the sparsity degree was set at $s = \sqrt{d}$ and the output dimension at $k = 200$. Iteration number for DT-RP was set at $N_{iter} = 4 \times 10^3$. Points have been spread along the horizontal axis to facilitate visualization

for all training set sizes and all datasets listed in Table 1, even when only 50 training samples were employed. This supports the claim that DT-RP is useful even if only a small amount of data samples are available when initializing the projection matrix.

4.4 Neighborhood preservation assessment

Despite its convenience and widespread usage, the Recall measure is limited by the fact that it only quantifies the neighborhood preservation for a single, fixed neighborhood size. This feature hinders the ability to analyze whether the neighborhoods are reproduced at different data scales and does not highlight the local and global properties of the mapping. For these reasons, some studies developed dimensionality reduction (DR) quality criteria which measure the high-dimensional neighborhood preservation in the projection [43], becoming generally adopted in several publications [44–46]. This neighborhood preservation principle is indeed considered as the driving factor in the DR quality [47]. Denoting the sets of the K nearest neighbors of $\mathbf{x}_i$ and $\mathbf{x}_i$ in the high-dimensional space and in the projection by v_{iK} and v'_{iK} respectively, their average normalized agreement can be computed as

$$Q_{\mathrm{NX}}(K) = \frac{1}{n}\sum_{i=1}^{n} \frac{\left|v_{iK} \cap v'_{iK}\right|}{K} \in [0, 1]. \qquad (13)$$

It corresponds to the recall measure with neighborhood size K. As $\mathbb{E}\left[Q_{\mathrm{NX}}(K)\right]$ equals $K/(n-1)$ for uniformly distributed samples in the output space,

$$R_{\mathrm{NX}}(K) = \frac{(n-1)\,Q_{\mathrm{NX}}(K) - K}{n-1-K} \qquad (14)$$

rescales $Q_{\mathrm{NX}}(K)$ to enable the comparison of different neighborhood sizes [48]. The $R_{\mathrm{NX}}(K)$ curves are typically displayed with a log-scale for K as local neighborhoods usually prevail. The area under the resulting curve, computed as

$$\mathrm{AUC} = \sum_{K=1}^{n-2} \frac{R_{\mathrm{NX}}(K)}{K} \Bigg/ \left(\sum_{K=1}^{n-2} \frac{1}{K}\right) \in [-1, 1], \qquad (15)$$

increases with the DR quality, quantified at all scales with an emphasis on the small ones [49].

Figure 3 depicts the $R_{\mathrm{NX}}(K)$ curves, which are computed using an experimental protocol similar to the one used in Section 4.1. The main difference is that, thanks to the $R_{\mathrm{NX}}(K)$ curves, we can actually visualize how the behavior of the compared algorithms changes depending on the neighborhood size. Note that in this case, each curve corresponds to one run of one of the algorithms, enabling us to observe the typical behavior of each method over 50 executions. As we can observe, the general shape of the curves greatly depends on the dataset. This is due to the fact that the high-dimensional neighborhood structure can change radically depending on the dataset, and this greatly influences the performances of RP as a function of the neighborhood size K. Nevertheless, in agreement with the results previously presented in this section, DT-RP outperforms standard RP for all datasets. Moreover, this holds for all the neighborhood sizes.

4.5 Performance stability of RP matrices across datasets

One may argue that DT-RP is in fact optimizing an intrinsic property of the projection matrix, i.e., that it just improves the projection matrix in general without really adapting it to a specific dataset. For instance, one might think that DT-RP simply reduces the level of sparsity of the projection matrix, whereas in fact no significant decrease in the sparsity level

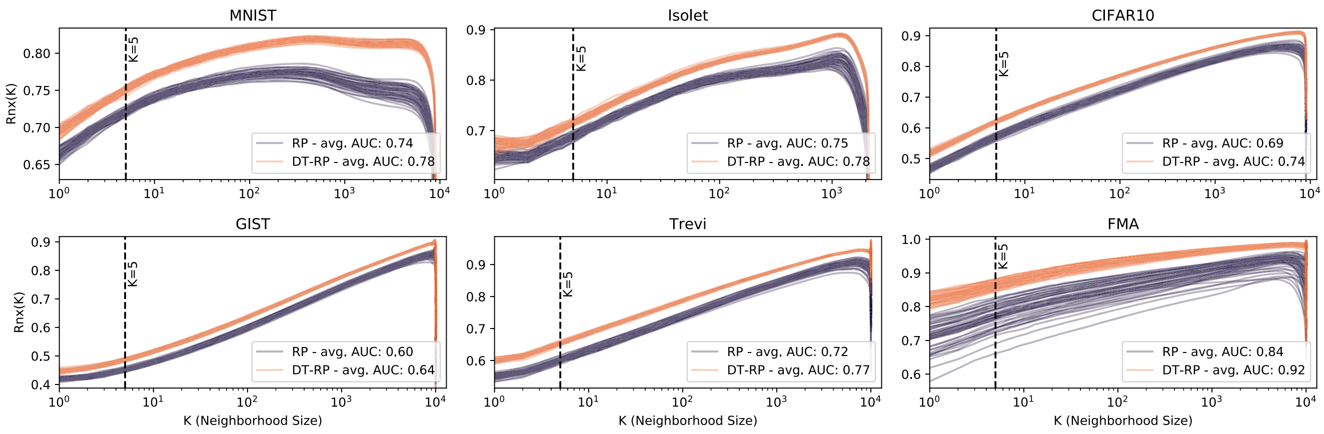

Fig. 3 Obtained $R_{\mathrm{NX}}(K)$ curves for 50 runs of standard sparse Random Projection (RP) and the proposed algorithm (DT-RP). In all experiments, the sparsity degree was set to $s = \sqrt{d}$ and the output dimension to $k = 200$. Iteration number for DT-RP was set to $N_{iter} = 4 \times 10^3$, and 500 training samples were provided to DT-RP. The neighborhood size $K = 5$ employed for the Recall measure in Fig. 1 is highlighted by a dashed line

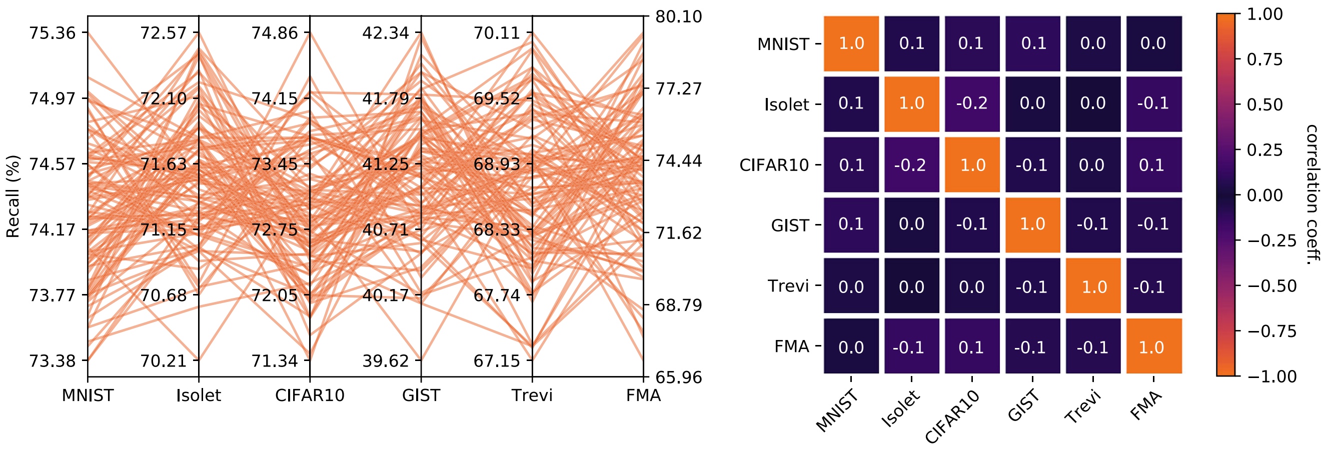

Fig. 4 (Left) Parallel-coordinate plot in which each line links the Recall scores of a single RP matrix instantiation on the six datasets. Only the first 518 features of each data set were kept to make it possible to employ the same RP matrix for each one of them. A total of 100 different RP matrices were used. (Right) Heat-map representation of the correlation matrix for the 100 Recall scores obtained on the different data sets. In all experiments, the sparsity degree was set at $s = \sqrt{518}$ and the output dimension at $k = 200$

of the projection matrices was registered in the experiments. Nevertheless, DT-RP might very well alter other intrinsic properties of the projection matrix. If that were the case, it would mean that it is feasible to improve RP matrices independently of the specific data to be projected. Then, a better data-independent tuning method could in principle be devised.

This subsection aims to evidence empirically that the performances are in fact determined by the combination of the RP matrix and the distribution of the data to be projected, rather than some property of the standalone projection matrix. To achieve this, we analyzed the correlation of the performance of RP matrices on different datasets. One major problem in this case is that, in order to use the same projection matrix on different datasets, they must have the same dimensionality. To manage this, we kept only the first 518 features of each dataset, as that is the smallest number of features among the datasets we employed. Then, 100 matrices were generated and evaluated in terms of Recall on each dataset. This was done following a protocol that was analogous to the one used in Section 4.1, except for the fact that, this time, the same RP matrix was used for all datasets at each run. Figure 4 shows a parallel-coordinates visualization of the results, where each line represents one of the 100 projection matrices generated. This figure also displays the correlation matrix for the results obtained with different datasets. As we can observe, no significant correlation exists between the results, in spite of the fact that the same RP matrix was used for all the datasets. In plain words, an RP matrix which performs well on one dataset, might perform poorly on another. This supports the claim that it is the combination of the RP matrix and the dataset that determines the performance, justifying the need for a data-dependent method for improving RP matrices, as the one proposed in this paper.

5 Conclusions

This paper has introduced the Data-Tuned Random Projection (DT-RP) method. The proposed algorithm aims to improve the performance of Random Projection matrices on specific datasets, assuming that at least a small number of samples are available at the moment of initializing the projection matrix. Essentially, DT-RP performs a guided random search by iteratively re-generating the projection directions that form the projection matrix and checking whether each modification decreases the average distortion of the pairwise distances. Therefore, the proposed method sacrifices the data-independence of RP to achieve better performance with samples following a specific distribution. Conveniently, once the projection matrix is selected, the computational cost of projecting data samples is not altered at all. This is because the projection matrices tuned by DT-RP consist of integer numbers and remain sparse. In addition, we have introduced some mathematical tricks that, thanks to the properties of squared Euclidean distances, allowed us to rapidly evaluate modifications in the projection matrix. In particular, DT-RP ran in less than five seconds in all of the experiments.

When comparing the standard RP method with DT-RP, the experimental results show consistent improvements in the performance as measured by the different neighborhood preservation criteria. While these improvements in performance are affected by the number of available training samples, the results show that statistically significant gains in performance can be achieved with as little as fifty training samples. This makes DT-RP a good option when some samples are available at the moment of initializing the projection matrix and when we want to obtain well-performing, database-friendly RP matrices, especially in cases where they are to remain in use over an extended time period. A potential drawback of DT-RP is that, as opposed to standard RP, it is not robust to changes in the distribution of samples after the projection matrix has been initialized. In the future, we intend to assess how the boost in the distance preservation properties of projection matrices can impact tasks such as distance-based classification, document retrieval or clustering.

In addition, the applicability of different heuristic optimization methods should be explored in the future, assessing if they can enable improvements in the performance or efficiency of the optimization problem described in this paper. For instance, simulated annealing [32] and genetic algorithms have been successfully applied in the literature to manage working with high-dimensional data [50].

Finally, it is worth noting how the field of machine learning has recently experienced significant breakthroughs thanks to the contributions of the deep learning paradigm. In this regard, deep learning models are being successfully applied in a wider range of domains, including dimensionality reduction [51] and representation learning [52, 53], thanks to their ability to model very complex properties of data. Moreover, random projection techniques have been recently applied in conjunction with certain deep learning models to make them more efficient [54]. In this regard, we can expect future research to continue bridging the gap between these two fields.

Acknowledgements The research of Daniel López-Sánchez has been financed by the Ministry of Education, Culture and Sports of the Spanish Government (University Faculty Training (FPU) programme, reference number FPU15/02339). Cyril de Bodt has been a Research Fellow of the Fonds de la Recherche Scientifique - FNRS, and is a post-doctoral fellow of the Belgian American Educational Foundation. John A. Lee is a Senior Research Associate with the Belgian F.R.S.-FNRS.

References

1. Vempala SS (2005) The random projection method, vol 65. American Mathematical Society
2. Achlioptas D (2003) Database-friendly random projections: Johnson-lindenstrauss with binary coins. J Comput Syst Sci 66(4):671–687
3. Li P, Hastie TJ, Church KW (2006) Very sparse random projections. In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, pp 287–296
4. Grellmann C, Neumann J, Bitzer S, Kovacs P, Tönjes A, Westlye LT, Andreassen OA, Stumvoll M, Villringer A, Horstmann A (2016) Random projection for fast and efficient multivariate correlation analysis of high-dimensional data: A new approach. Front Genet 7:102
5. Zhao K, Alavi A, Wiliem A, Lovell BC (2016) Efficient clustering on riemannian manifolds: A kernelised random projection approach. Pattern Recogn 51:333–345
6. Ye M, Liu W, Wei J, Hu X (2016) Fuzzy-means and cluster ensemble with random projection for big data clustering. Math Probl Eng 2016
7. Alzu'bi A, Abuarqoub A (2020) Deep learning model with low-dimensional random projection for large-scale image search. Eng Sci Technol Int J 23(4):911–920
8. Qiao Y, Zhao Y (2015) Rotation invariant texture classification using principal direction estimation and random projection. J Inf Hiding Multimed Sig Process 6(3):534–543
9. López-Sánchez D (2017) Improving random projection with genetic algorithms: student research abstract. In: Proceedings of the Symposium on Applied Computing. ACM, pp 828–829
10. Johnson WB, Lindenstrauss J (1984) Extensions of lipschitz mappings into a hilbert space. Contemp Math 26(189-206):1
11. Dasgupta S, Gupta A (2003) An elementary proof of a theorem of johnson and lindenstrauss. Random Struct Algorithm 22(1):60–65
12. Frankl P, Maehara H (1988) The johnson-lindenstrauss lemma and the sphericity of some graphs. J Comb Theory Ser B 44(3):355–362
13. Indyk P, Motwani R (1998) Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the thirtieth annual ACM symposium on Theory of computing. ACM, pp 604–613
14. Matoušek J (2008) On variants of the johnson–lindenstrauss lemma. Random Struct Algorithm 33(2):142–156
15. Achlioptas D (2001) Database-friendly random projections. In: Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. ACM, pp 274–281
16. Li P, Mitzenmacher M, Slawski M (2016) Quantized random projections and non-linear estimation of cosine similarity. In: Advances in Neural Information Processing Systems, pp 2756–2764
17. Valsesia D, Coluccia G, Bianchi T, Magli E (2015) Compressed fingerprint matching and camera identification via random projections. IEEE Trans Inf Forensic Secur 10(7):1472–1485
18. Jacques L (2017) Small width, low distortions: quantized random embeddings of low-complexity sets. IEEE Trans Inf Theory 63(9):5477–5495
19. Rachkovskij DA, Misuno IS, Slipchenko SV (2012) Randomized projective methods for the construction of binary sparse vector representations. Cybern Syst Anal 48(1):146–156
20. Wimalajeewa T, Varshney PK (2015) Wireless compressive sensing over fading channels with distributed sparse random projections. IEEE Trans Signal Inf Process Over Netw 1(1): 33–44

21. Rachkovskij DA (2015) Formation of similarity-reflecting binary vectors with random binary projections. Cybern Syst Anal 51(2):313–323
22. Cheng L, You C, Guan Y (2016) Random projections for non-linear dimensionality reduction. Int J Mach Learn Comput 6(4):220–225
23. López-Sánchez D, Arrieta AG, Corchado JM (2018) Data-independent random projections from the feature-space of the homogeneous polynomial kernel. Pattern Recogn 82:130–146
24. Ailon N, Chazelle B (2009) The fast johnson–lindenstrauss transform and approximate nearest neighbors. SIAM J Comput 39(1):302–322
25. Ailon N, Liberty E (2013) An almost optimal unrestricted fast johnson-lindenstrauss transform. ACM Trans Algorithm (TALG) 9(3):1–12
26. Bamberger S, Krahmer F (2021) Optimal fast johnson–lindenstrauss embeddings for large data sets. Sampling Theory Signal Process Data Anal 19(1):1–23
27. Binjubeir M, Ahmed AA, Ismail MAB, Sadiq AS, Khan MK (2019) Comprehensive survey on big data privacy protection. IEEE Access 8:20067–20079
28. Soliman RF, Amin M, Abd El-Samie FE (2019) A modified cancelable biometrics scheme using random projection. Ann Data Sci 6(2):223–236
29. Tasoulis S, Cheng L, Välimäki N, Croucher NJ, Harris SR, Hanage WP, Roos T, Corander J (2014) Random projection based clustering for population genomics. In: 2014 IEEE international conference on big data (big data). IEEE, pp 675–682
30. Wan S, Kim J, Won KJ (2020) Sharp: hyperfast and accurate processing of single-cell rna-seq data via ensemble random projection. Genome Res 30(2):205–213
31. Carraher LA, Wilsey PA, Moitra A, Dey S (2016) Random projection clustering on streaming data. In: 2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW). IEEE, pp 708–715
32. Delahaye D, Chaimatanan S, Mongeau M (2019) Simulated annealing: From basics to applications. Springer
33. Deng L (2012) The mnist database of handwritten digit images for machine learning research [best of the web]. IEEE Signal Proc Mag 29(6):141–142
34. Fanty M, Cole R (1991) Spoken letter recognition. In: Advances in Neural Information Processing Systems, pp 220–226
35. Krizhevsky A, Hinton G (2009) Learning multiple layers of features from tiny images. Technical Report. Citeseer
36. Jegou H, Douze M, Schmid C (2011) Product quantization for nearest neighbor search. IEEE Trans Pattern Anal Mach Intell 33(1):117–128
37. Winder SAJ, Brown M (2007) Learning local image descriptors. In: Computer Vision and Pattern Recognition, 2007. CVPR’07. IEEE Conference on. IEEE, pp 1–8
38. Defferrard M, Mohanty SP, Carroll SF, Salathé M (2018) Learning to recognize musical genre from audio: *Challenge overview*. In: The 2018 Web Conference Companion. ACM Press
39. McFee B, Raffel C, Liang D, Ellis DanielPW, McVicar M, Battenberg E, Nieto O (2015) librosa: Audio and music signal analysis in python. In: Proceedings of the 14th python in science conference, pp 18–25
40. Hyvönen V, Pitkänen T, Tasoulis S, Jääsaari E, Tuomainen R, Wang L, Corander J, Roos T (2016) Fast nearest neighbor search through sparse random projections and voting. In: Big Data (Big Data), 2016 IEEE International Conference on. IEEE, pp 881–888
41. d’Agostino RB (1971) An omnibus test of normality for moderate and large size samples. Biometrika 58(2):341–348
42. Shaffer JP (1995) Multiple hypothesis testing. Ann Rev Psychol 46(1):561–584
43. Lee JA, Verleysen M (2009) Quality assessment of dimensionality reduction: Rank-based criteria. Neurocomputing 72(7):1431–1443
44. de Bodt C, Mulders D, Verleysen M, Lee JA (2019) Nonlinear dimensionality reduction with missing data using parametric multiple imputations. IEEE Trans Neural Netw Learn Syst 30(4):1166–1179
45. Mokbel B, Lueks W, Gisbrecht A, Hammer B (2013) Visualizing the quality of dimensionality reduction. Neurocomputing 112:109–123
46. de Bodt C, Mulders D, Verleysen M, Lee JA (2020) Fast multiscale neighbor embedding. IEEE Trans Neural Netw Learn Syst:1–15
47. Venna J, Peltonen J, Nybo K, Aidos H, Kaski S (2010) Information retrieval perspective to nonlinear dimensionality reduction for data visualization. J Mach Learn Res 11:451–490
48. Lee JA, Renard E, Bernard G, Dupont P, Verleysen M (2013) Type 1 and 2 mixtures of kullback–leibler divergences as cost functions in dimensionality reduction based on similarity preservation. Neurocomputing 112:92–108
49. Lee JA, Peluffo-Ordóñez DH, Verleysen M (2015) Multi-scale similarities in stochastic neighbour embedding: Reducing dimensionality while preserving both local and global structure. Neurocomputing 169:246–261
50. Tran B, Xue B, Zhang M (2019) Genetic programming for multiple-feature construction on high-dimensional classification. Pattern Recogn 93:404–417
51. Bhatt G, Jha P, Raman B (2019) Representation learning using step-based deep multi-modal autoencoders. Pattern Recogn
52. Chen B, Deng W (2019) Deep embedding learning with adaptive large margin n-pair loss for image retrieval and clustering. Pattern Recogn 93:353–364
53. Zhe X, Chen S, Yan H (2019) Directional statistics-based deep metric learning for image classification and retrieval. Pattern Recogn 93:113–123
54. López-Sánchez D, Arrieta AG, Corchado JM (2019) Compact bilinear pooling via kernelized random projection for fine-grained image categorization on low computational power devices. Neurocomputing