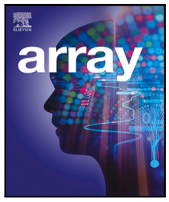
Array 17 (2023) 100274

Contents lists available at ScienceDirect

Array

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



Random projection tree similarity metric for SpectralNet

Mashaan Alshammari [a,*], John Stavrakakis [b], Adel F. Ahmed [c], Masahiro Takatsuka [b]

[a] Independent Researcher, Riyadh, Saudi Arabia
[b] School of Computer Science, The University of Sydney, NSW, Australia
[c] Information and Computer Science Department, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia

ARTICLE INFO

Keywords:
k-nearest neighbor
Random projection trees
SpectralNet
Graph clustering
Unsupervised learning

ABSTRACT

SpectralNet is a graph clustering method that uses neural network to find an embedding that separates the data. So far it was only used with *k*-nn graphs, which are usually constructed using a distance metric (e.g., Euclidean distance). *k*-nn graphs restrict the points to have a fixed number of neighbors regardless of the local statistics around them. We proposed a new SpectralNet similarity metric based on random projection trees (rpTrees). Our experiments revealed that SpectralNet produces better clustering accuracy using rpTree similarity metric compared to *k*-nn graph with a distance metric. Also, we found out that rpTree parameters do not affect the clustering accuracy. These parameters include the leaf size and the selection of projection direction. It is computationally efficient to keep the leaf size in order of $\log(n)$, and project the points onto a random direction instead of trying to find the direction with the maximum dispersion.

1. Introduction

Graph clustering is one of the fundamental tasks in unsupervised learning. The flexibility of modeling any problem as a graph has made graph clustering very popular. Extracting clusters' information from graph is computationally expensive, as it usually done via eigen decomposition in a method known as spectral clustering. A recently proposed method, named as SpectralNet [1], was able to detect clusters in a graph without passing through the expensive step of eigen decomposition.

SpectralNet starts by learning pairwise similarities between data points using Siamese nets [2]. The pairwise similarities are stored in an affinity matrix A, which is then passed through a deep network to learn an embedding space. In that embedding space, pairs with large similarities fall in a close proximity to each other. Then, similar points can be clustered together by running k-means in that embedding space. In order for SpectralNet to produce accurate results, it needs an affinity matrix with rich information about the clusters. Ideally, a pair of points in the same cluster should be connected with an edge carrying a large weight. If the pair belong to different clusters, they should be connected with an edge carrying a small weight, or no weight which is indicated by a zero entry in the affinity matrix.

SpectralNet uses Siamese nets to learn informative weights that ensure good clustering results. However, the Siamese nets need some information beforehand. They need some pairs to be labeled as negative and positive pairs. Negative label indicates a pair of points belonging to different clusters, and a positive label indicates a pair of points in the same cluster. Obtaining negative and positive pairs can be done in a semi-supervised or unsupervised manner. The authors of SpectralNet have implemented it as a semi-supervised and an unsupervised method. Using the ground-truth labels to assign negative and positive labels, makes the SpectralNet semi-supervised. On the other hand, using a distance metric to label closer points as positive pairs and farther points as negative pairs, makes the SpectralNet unsupervised. In this study, we are only interested in an unsupervised SpectralNet.

Unsupervised SpectralNet uses a distance metric to assign positive and negative pairs. A common approach is to get the nearest k neighbors for each point and assign those neighbors as positive pairs. A random selection of farther points are labeled as negative pairs. But this approach restricts all points to have a fixed number of positive pairs, which is unsuitable if clusters have different densities. In this work, we proposed a similarity metric based on random projection trees (rpTrees) [3,4]. An example of an rpTree is shown in Fig. 1. rpTrees do not restrict the number of positive pairs, as this depends on how many points in the leaf node.

The main contributions of this work can be summarized in the following points:

- Proposing a similarity metric for SpectralNet based on random projection trees (rpTrees) that does not restrict the number of positive pairs and produces better clustering accuracy.

* Corresponding author.
E-mail addresses: mashaan.awad1930@alum.kfupm.edu.sa (M. Alshammari), john.stavrakakis@sydney.edu.au (J. Stavrakakis), adelahmed@kfupm.edu.sa (A.F. Ahmed), masa.takatsuka@sydney.edu.au (M. Takatsuka).

https://doi.org/10.1016/j.array.2022.100274
Received 2 November 2022; Received in revised form 17 December 2022; Accepted 17 December 2022
Available online 21 December 2022

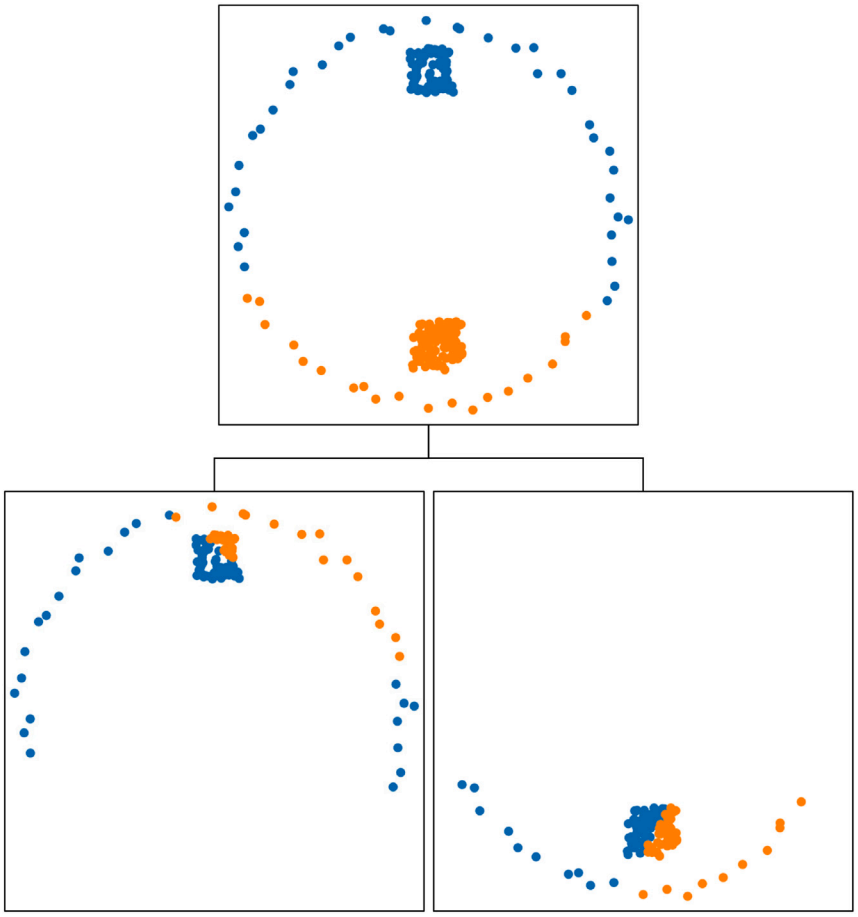


Fig. 1. An example of rpTree; points in blue are placed in the left branch and points in orange are placed in the right branch. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

- Investigating the influence of the leaf size parameter n_0 on the clustering accuracy.
- Performing an in-depth analysis of the projection direction in rpTrees, and examine how it influences the clustering accuracy of SpectralNet.

2. Related work

2.1. Graph neural networks (GNNs)

GNNs became researchers' go-to option to perform graph representation learning. Due to its capability in fusing nodes' attributes and graph structure, GNN has been widely used in many applications such as knowledge tracing [5] and sentiment analysis [6]. The most well-known form of GNN is graph convolutional network (GCN) [7].

Researchers have been working on improving GCN. Franceschi et al. [8] have proposed to learn the adjacency matrix A by running GCN for multiple iterations and adjusting the graph edges in A accordingly. Another problem with GCN is its vulnerability to adversarial attack. Yang et al. used GCN with domain adaptive learning [9]. Domain adaptive learning attempts to transfer the knowledge from a labeled source graph to unlabeled target graph. Unseen nodes from the target graph can later be used for node classification.

2.2. Graph clustering using deep networks

GCN performs semi-supervised node classification. Due to limited availability of labeled data in some applications, researchers developed graph clustering using deep networks. Yang et al. developed a deep model for network clustering [10]. They used graph neural network (GCN) to encode the adjacency matrix A and the feature matrix X. They also used multilayer perceptron (MLP) to encode the feature matrix X. The output is clustered using Gaussian mixture model (GMM), where GMM parameters are updated throughout training. A similar approach was used by Wang et al. [11], where they used autoencoders to learn latent representation. Then, they deploy the manifold learning technique UMAP [12] to find a low dimensional space. The final clustering assignments are given by k-means. Affeldt et al. used autoencoders to obtain m representations of the input data [13]. The affinity matrices of these m representations are merged into a single matrix. Then spectral clustering was performed on the merged matrix. One drawback with this approach is that it still needs eigen decomposition to find the embedding space.

SpectralNet is another approach for graph clustering using deep networks, which was proposed by Shaham et al. [1]. They used Siamese nets to construct the adjacency matrix A, which is then passed through a deep network. Nodes in the embedding space can be clustered using k-means. An extension to SpectralNet was proposed by Huang et al. [14], where multiple Siamese nets are trained on multiple views. Each view is passed into a neural network to find an embedding space. All embedding spaces are fused in the final stage, and k-means was run to find the cluster labels. Another approach to employ deep learning for spectral clustering was introduced by Wada et al. [15]. Their method starts by identifying hub points, which serve as the core of clusters. These hub points are then passed to a deep network to obtain the cluster labels for the remaining points.

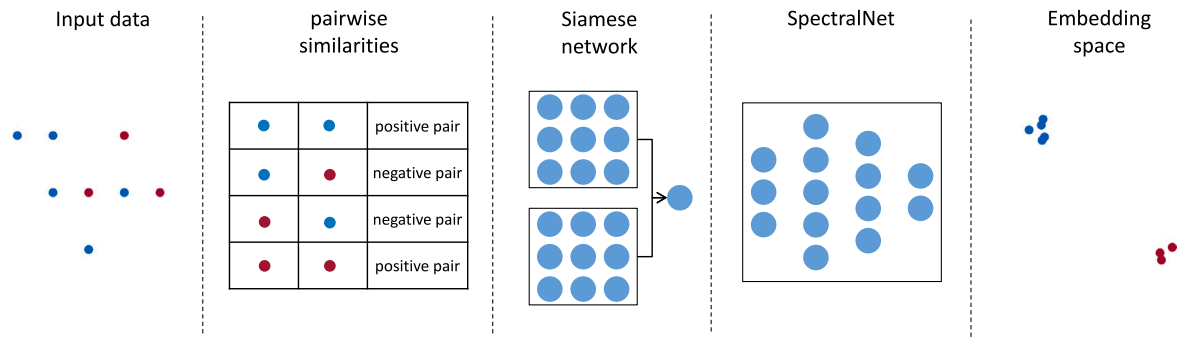


Fig. 2. An outline of the used algorithm. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

2.3. Graph similarity metrics

Every graph clustering method needs a metric to construct pairwise similarities. A shared neighbor similarity was introduced by Zhang et al. [16]. They applied their method to *attributed graphs*, a special type of graph where each node has feature attributes. They used shared neighbor similarities to highlight clusters' discontinuity. The concept of shared neighbors could be traced back to Jarvis–Patrick algorithm [17]. It is important to mention the higher cost associated with shared neighbor similarity. Because all neighbors have to be matched, instead of computing one value such as the Euclidean distance.

Another way of constructing pairwise similarities was introduced by Wen et al. [18], where they utilized Locality Preserving Projection (LPP) and hypergraphs. First, all points are projected onto a space with reduced dimensionality. The pairwise similarities are constructed using a heat kernel (Eq. (1)). Second, a hypergraph Laplacian matrix L_H is used to replace the regular graph Laplacian matrix L. Hypergraphs would help to capture the higher relations between vertices. Two things needed to be considered when applying this method: (1) the σ parameter in the heat kernel needs careful tuning [19], and (2) the computational cost for hypergraph Laplacian matrix L_H. Density information were incorporated into pairwise similarity construction by Kim et al. [20]. The method defines (locally dense points) that are separated from each other by (locally sparse points). This approach falls under the category of DBSCAN clustering [21]. These methods are iterative by nature and need a stopping criterion to be defined.

$$A_{i,j} = exp^{\frac{\|x_i - x_j\|_2^2}{2\sigma^2}} \tag{1}$$

Considering the literature on graph representation learning, it is evident that SpectralNet [1]: (1) offers a cost-efficient method to perform graph clustering using deep networks and (2) it does not require labeled datasets. The problem is that it uses k-nearest neighbor graph with distance metric. This restricts points from pairing with more neighbors if they are in a close proximity. A suitable alternative would be a similarity metric based on random projection trees [3,4]. rpTrees similarity were already used in spectral clustering by [22,23]. But they are yet to be extended to graph clustering using deep networks.

3. SpectralNet and pairwise similarities

The proposed rpTree similarity metric was used in SpectralNet alongside the distance metric that was used for k-nearest neighbor graph. The SpectralNet algorithm consists of four steps: (1) identifying positive and negative pairs, (2) running Siamese net using positive and negative pairs to construct the affinity matrix A, (3) SpectralNet that maps points onto an embedding space, and (4) clusters are detected by running k-means in the embedding space. An illustration of these steps is shown in Fig. 2. The next subsection explains the used neural networks (Siamese and SpectralNet). The discussion of similarity metrics and their complexity is introduced in the following subsections.

3.1. SpectralNet

The first step in SpectralNet is the Siamese network, which consists of two or more neural networks with the same structure and parameters. These networks has a single output unit that is connected to the output layers of the networks in the Siamese net. For simplicity let us assume that the Siamese net consists of two neural networks N_1 and N_2. Both networks received inputs x_1 and x_2 respectively, and produce two outputs z_1 and z_2. The output unit compared the two outputs using the Euclidean distance. The distance should be small if x_1 and x_2 are a positive pair, and large if they are a negative pair. The Siamese net is trained to minimize contrastive loss, that is defined as:

$$L_{\text{contrastive}} = \begin{cases} \|z_1 - z_2\|^2, & \text{if } (x_1, x_2) \text{ is a positive pair} \\ max(c - \|z_1 - z_2\|, 0), & \text{if } (x_1, x_2) \text{ is a negative pair,} \end{cases} \tag{2}$$

where c is a constant that is usually set to 1. Then the Euclidean distance obtained via the Siamese net $\|z_1 - z_2\|$ is used in the heat kernel (see Eq. (1)) to find the similarities between data points and construct the affinity matrix A.

The SpectralNet uses a gradient step to optimize the loss function $L_{SpectralNet}$:

$$L_{SpectralNet} = \frac{1}{m^2} \sum_{i,j=1}^{m} a_{i,j} \left\| y_i - y_j \right\|^2 \tag{3}$$

where m is the batch size; a of size $m \times m$ is the affinity matrix of the sampled points; y_i and y_j are the expected labels of the samples x_i and x_j. But the optimization of this functions is constrained, since the last layer is set to be a constraint layer that enforces orthogonalization. Therefore, SpectralNet has to alternate between orthogonalization and gradient steps. Each of these steps uses a different random batch m from the original data X. Once the SpectralNet is trained, all samples $x_1, x_2, \ldots, x_n$ are passed through network to get the predictions $y_1, y_2, \ldots, y_n$. These predictions represent coordinates on the embedding space, where k-means operates and finds the clustering.

3.2. Constructing pairwise similarities using k-nn

The original algorithm of SpectralNet [1] has used k-nearest neighbor graph with distance metric to find positive and negative pairs. The positive pairs are the nearest neighbors according to the selected value of k, the original method has k set to be 2. The negative pairs were selected randomly from the farther neighbors. An illustration of this process is shown in Fig. 3

Restricting the points to have a fixed number of positive pairs can be a disadvantage of using k-nn. That is a problem we are trying to overcome by using rpTrees to construct positive and negative pairs. In rpTrees, there is no restriction on how many number of pairs for individual points. It depends on how many points ended up in the same leaf node.

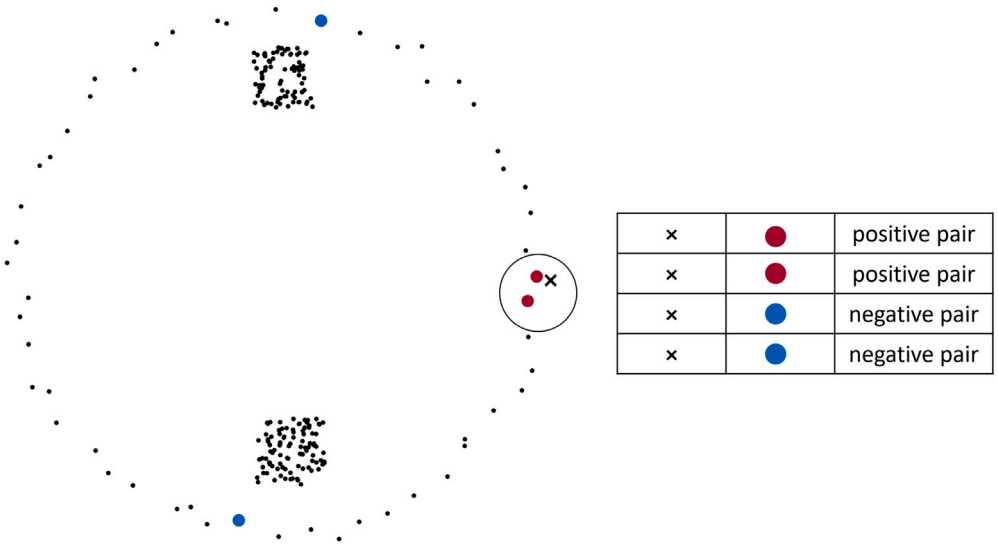


Fig. 3. Constructing positive and negative pairs using k-nn search; red points are the nearest neighbors when $k = 2$; blue points are selected randomly. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

3.3. Constructing pairwise similarities using rpTrees

rpTrees start by choosing a random direction $\vec{r}$ from the unit sphere S^{D-1}, where D is the number of dimensions. All points in the current node W are projected onto $\vec{r}$. On that reduced space $\mathbb{R}^{D-1}$, the algorithm picks a dimension uniformly at random and chooses the split point c randomly between $[\frac{1}{4}, \frac{3}{4}]$. The points less than the split point $x < c$ are placed in the left child W_L, and the points larger than the split point $x > c$ are placed in the right child W_R. The algorithm continues to partition the points recursively, and stops when the split produces a node with points less than the leaf size parameter n_0.

To create positive pairs for the Simese net, we pair all points in one leaf node. So, points that fall onto the same leaf node are considered similar, and we mark them as positive pairs. For negative pairs, we pick one leaf node W_x, and from the remaining set of leaf nodes we randomly pick W_y. Then, we pair all points in W_x with the points in W_y, and mark them as negative pairs (Eq. (4)). An illustration of this process is shown in Fig. 4.

$$\begin{aligned}(p,q) \in E(positive) &\Leftrightarrow p \in W_x \text{ and } q \in W_x \\ (p,q) \in E(negative) &\Leftrightarrow p \in W_x \text{ and } q \in W_y.\end{aligned} \tag{4}$$

3.4. Complexity analysis for computing pairwise similarities

We will use the number of positive pairs to analyze the complexity of the similarity metric used in the original SpectralNet method and the metric proposed in this paper. The original method uses the nearest k neighbors as positive pairs. This is obviously grows linearly with n, since we have to construct $n \times k$ pairs and pass them to the Siamese net [2].

Before we analyze the proposed metric, we have to find how many points will fall into a leaf node of an rpTree. This question is usually asked in proximity graphs [24]. If we place a squared tessellation T on top of n data points (please refer to section 9.4.1 by Barthelemy [25] for more elaboration). T has an area of n and a side length of $\sqrt{n}$. Each small square s in T has an area of $\log(n)$. The probability of having more than k neighbors in s is $P(l > k)$, where $l = k + 1, \ldots, n$. The probability $P(l > k)$ follows the homogeneous Poisson process. This probability approximately equals $\frac{1}{n}$, which is very small, suggesting there is a significant chance of having at most $\log(n)$ neighbors in a square s. Since rpTrees follow the same approach of partitioning the search space, it is safe to assume that each leaf node would have at most $\log(n)$ data points.

The proposed metric depends on the number of leaf nodes in the rpTree and the number of points in each leaf node (n_0). The leaf size n_0 is a parameter that can be tuned before running rpTree. It also determines how many leaf nodes in the tree. Because the method will stop partitioning when the number of points in a leaf node reaches a minimum limit. Then we have to pair all the points in the leaf node.

To visualize this effect, we have to fix all parameters and vary n. In Fig. 5, we set k to be 2 and 10. The leaf size n_0 was set to 20 and 100. The number of points n was in the interval $[100, 100000]$. With k-nn graph we need $n \times k$ positive pairs, and in rpTree similarity we need ${n_0}^2 \times \frac{n}{n_0} = n \times n_0$ positive pairs. So, both similarity metrics grow linearly with n. The main difference is how the points are partitioned. k-nn graph uses kd-tree which produces the same partition with each run making the number of positive pairs fixed. But rpTrees partitions points randomly, so the number of positive pairs will deviate from $n \times n_0$.

4. Experiments and discussions

In our experiments we compared the similarity metrics using k-nearest neighbor and rpTree, in terms of: (1) clustering accuracy and (2) storage efficiency. The clustering accuracy was measured using Adjusted Rand Index (`ARI`) [26]. Given the true grouping T and the predicted grouping L, `ARI` is computed using pairwise comparisons. n_{11} if the pair belong to the same cluster in T and L groupings, and n_{00} if the pair in different clusters in T and L groupings. n_{01} and n_{10} if there is a mismatch between T and L. `ARI` is defined as:

$$ARI(T, L) = \frac{2(n_{00}n_{11} - n_{01}n_{10})}{(n_{00} + n_{01})(n_{01} + n_{11}) + (n_{00} + n_{10})(n_{10} + n_{11})}. \tag{5}$$

The storage efficiency was measured by the number of total pairs used. We avoid using machine dependent metrics like the running time.

We also run additional experiments to investigate how the rpTrees parameters are affecting the similarity metric based on rpTree. The first parameter was the leaf size parameter n_0, which determines the minimum number of points in a leaf node. The second parameter was how to select the projections direction. There are a number of methods to choose the random direction. We tested these methods to see how they would affect the performance.

The two dimensional datasets used in our experiments are shown in Fig. 6. The remaining datasets were retrieved from scikit-learn library [27,28], except for the `mGamma` dataset which was downloaded from UCI machine learning repository [29]. All experiments were coded in python 3 and run on a machine with 20 GB of memory and a 3.10 GHz Intel Core i5-10500 CPU. The code can be found on https://github.com/mashaan14/RPTree.

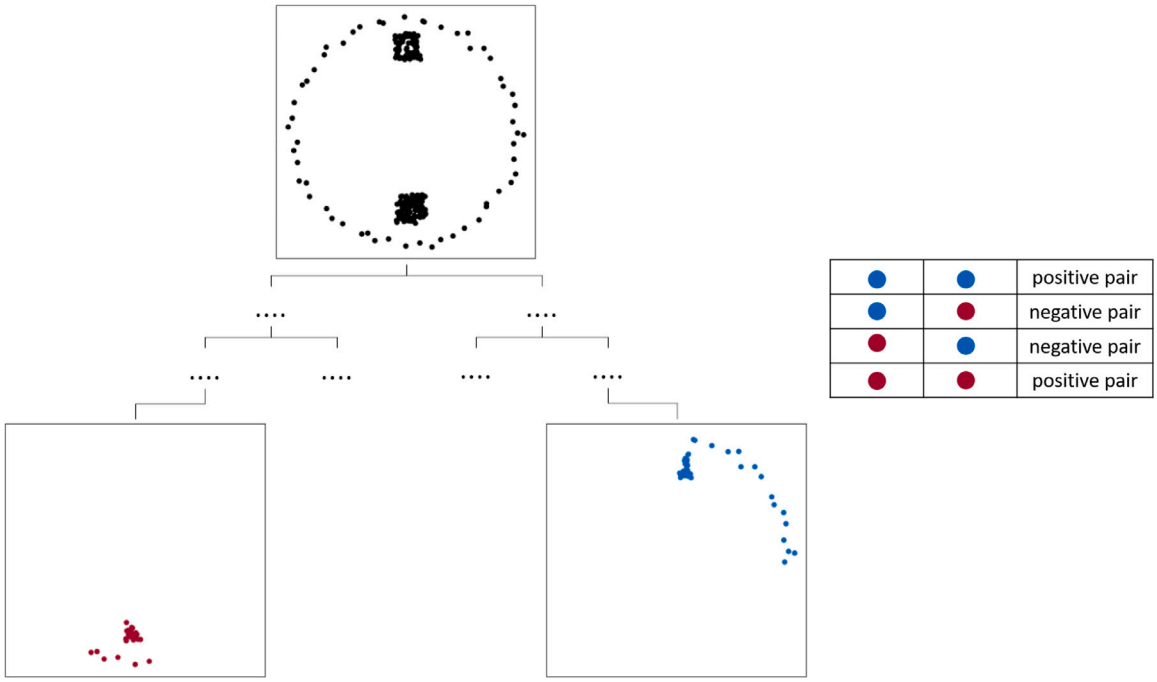


Fig. 4. Constructing positive and negative pairs using rpTree. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

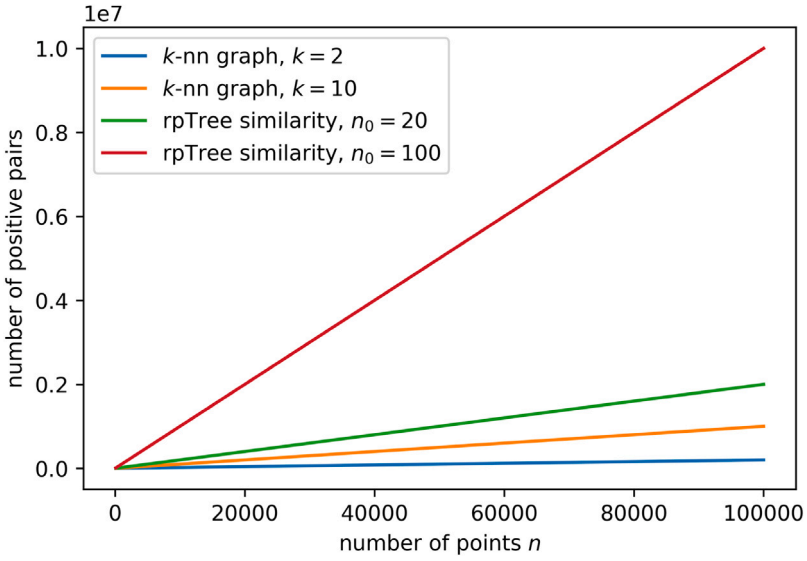


Fig. 5. The expected number of positive pairs using k-nn and rpTree similarities. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

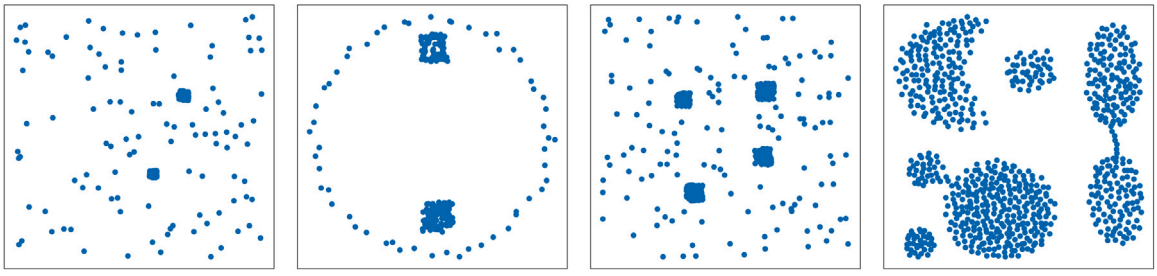


Fig. 6. Synthetic datasets used in the experiments; from left to right `Dataset 1` to `Dataset 4`.

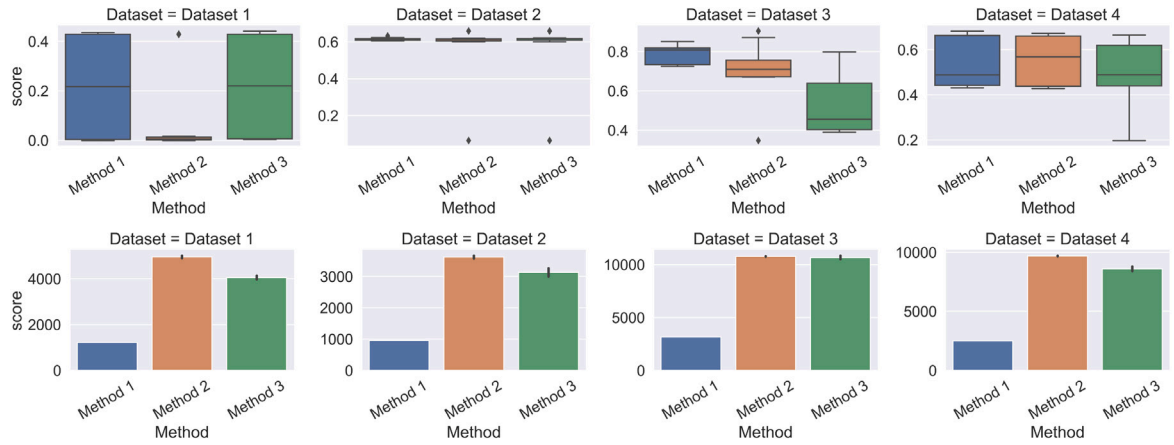


Fig. 7. Experiments with synthetic datasets; `Method 1` is k-nn graph with $k = 2$, `Method 2` is k-nn graph with varying k, and `Method 3` is rpTree similarity with $n_0 = 20$; (top) ARI scores for 10 runs, (bottom) number of total pairs. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

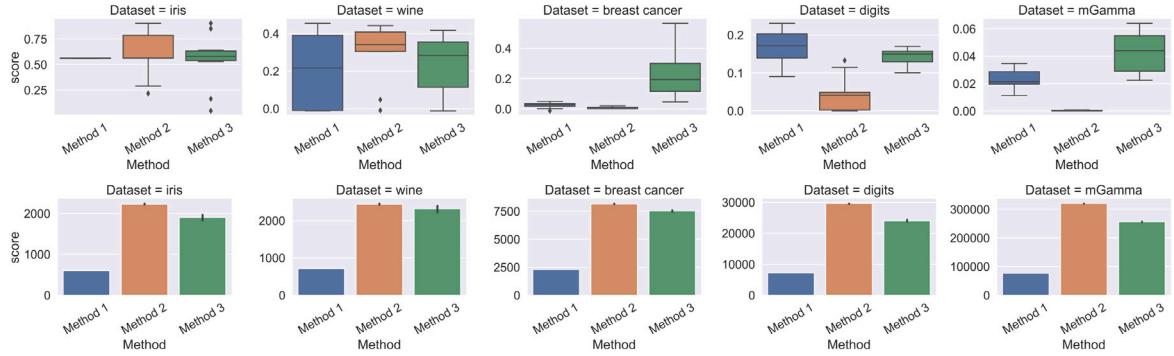


Fig. 8. Experiments with real datasets; `Method 1` is k-nn graph with $k = 2$, `Method 2` is k-nn graph with varying k, and `Method 3` is rpTree similarity with $n_0 = 20$; (top) ARI scores for 10 runs, (bottom) number of total pairs. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

4.1. Experiments using k-nn and rpTree similarity metrics

Three methods were used in this experiment. `Method 1` is the original SpectralNet method by Shaham et al. [1]. `Method 2` was developed by Alshammari et al. [30], it sets k dynamically based on the statistics around the points. `Method 3` is the proposed method which uses an rpTree similarity instead of k-nn graph.

With the four synthetic datasets, all three methods delivered similar performances shown in Fig. 7. Apart from `Dataset 3`, where rpTree similarity performed lower than other methods. This could be attributed to how the clusters are distributed in this dataset. The rpTree splits separated points from the same cluster, which lowers the `ARI`. `Method 2` has the maximum number of pairs over all three methods. The number of pairs in `Method 2` and `Method 3` deviated slightly from the mean, unlike `Method 1` which has the same number of pairs with each run because k was fixed ($k = 2$).

rpTrees similarity outperformed other methods in three out of the five real datasets `iris`, `breast cancer`, and `mGamma` as shown in Fig. 8. k-nn with Euclidean distance performed poorly in `breast cancer`, which suggests that connecting to two neighbors was not enough to accurately detect the clusters. Yan et al. reported a similar finding where clustering using rpTree similarity was better than clustering using Gaussian kernel with Euclidean distance [23]. They showed the heatmap of the similarity matrix generated by the Gaussian kernel and by rpTree.

As for the number of pairs, the proposed similarity metric was the second lowest method that used total pairs across all five datasets. Because of the randomness involved in rpTree splits, the proposed similarity metric has a higher standard deviation for the number of total pairs.

4.2. Investigating the influence of the leaf size parameter n_0

One of the important parameters in rpTrees is the leaf size n_0. It determines when the rpTree stops growing. If the number of points in a leaf node is less than the leaf size n_0, that leaf node would not be split further.

By looking at the clustering performance in synthetic datasets shown in Fig. 9 (top), we can see that we are not gaining much by increasing the leaf size n_0. In fact, increasing the leaf size n_0 might affect the clustering accuracy like what happened in `Dataset 3`. The number of pairs is also related with the leaf size n_0, as it grows with n_0. This is shown in Fig. 9 (bottom).

Increasing the leaf size n_0 helped us to get higher `ARI` with `breast cancer` and `mGamma` as shown in Fig. 10. With other real datasets it was not improving the clustering accuracy measured by `ARI`. We also observed that the number of pairs increases as we increase n_0.

Ram and Sinha [31] provided a discussion on how to set the parameter n_0. They stated that n_0 controls the balance between global search and local search. Overall, they stated that n_0 effect on search accuracy is "quite benign".

4.3. Investigating the influence of the dispersion of points along the projection direction

The original algorithm of rpTrees [3] suggests using a random direction selected at random. But a recent application of rpTree by Yan et al. [32] recommended picking three random directions ($nTry = 3$) and use the one that provides the maximum spread of data points. To investigate the effect of this parameter, we used four methods for picking a projection direction: (1) picking one random direction, (2)

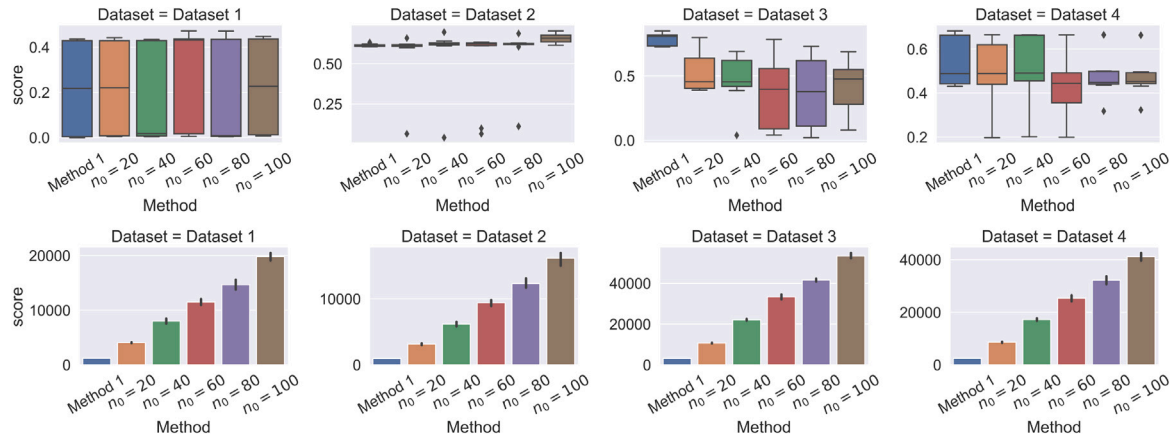


Fig. 9. Experiments with synthetic datasets; `Method 1` is k-nn graph with $k = 2$, other methods use rpTree similarity with varying n_0; (top) ARI scores for 10 runs, (bottom) number of total pairs. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

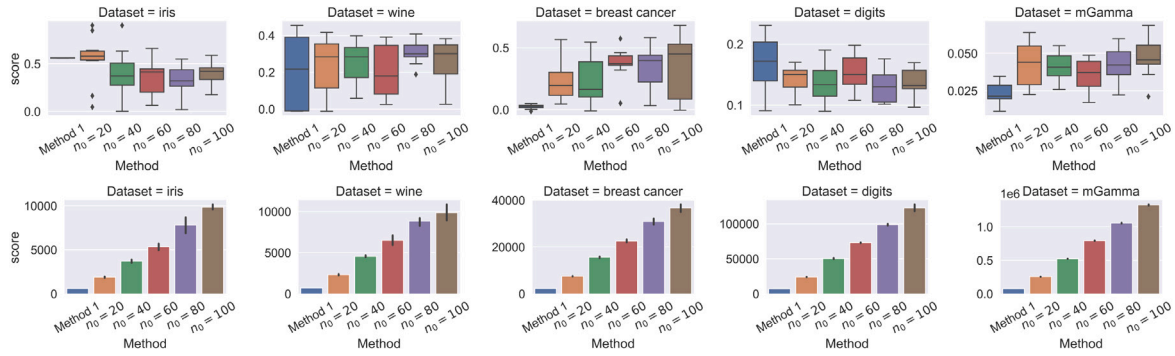


Fig. 10. Experiments with real datasets; `Method 1` is k-nn graph with $k = 2$, other methods use rpTree similarity with varying n_0; (top) ARI scores for 10 runs, (bottom) number of total pairs. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

picking three random direction ($nTry = 3$) and use the one with maximum spread, (3) picking nine random directions ($nTry = 9$), and (4) using principal component analysis (PCA) to find the direction with the maximum spread.

By looking at `ARI` numbers for synthetic datasets (Fig. 11) and real datasets (Fig. 12), we observed that we are not gaining much by trying to maximize the spread of projected points. This parameter has very little effect. Also, all methods with different strategies to pick the projection direction have used the same number of pairs.

In a final experiment, we measured the accuracy differences between a choosing random projection direction against an ideal projection direction. As there are infinite number of projection directions, we instead sample up to 1000 different directions uniformly in the unit sphere, and then pick the best performing among those (see Fig. 13). For the tested datasets, we compared the best performing direction against the random direction. We found no significant difference among mean of those 100 or 1000 samples with the random vector as shown in Figs. 14 and 15. Our finding is supported by the recent efforts in the literature [33] to limit the number of random directions to make rpTrees more storage efficient.

5. Conclusion

The conventional way for graph clustering involves the expensive step of eigen decomposition. SpectralNet presents a suitable alternative that does not use eigen decomposition. Instead the embedding is achieved by neural networks.

The similarity metric that was used in SpectralNet was a distance metric for k-nearest neighbor graph. This approach restricts points from being paired with further neighbors because k is fixed. A similarity metric based on random projection trees (rpTrees) eases this restriction and allows points to pair with all points falling in the same leaf node. The proposed similarity metric improved the clustering performance on the tested datasets.

There are number of parameters associated with rpTree similarity metric. Parameters like the minimum number of points in a leaf node n_0, and how to select the projection direction to split the points. After running experiments while varying these parameters, we found that rpTrees parameters have a limited effect on the clustering performance. So we recommend keeping the number of points in a leaf node n_0 in order of $log(n)$. Also, it is more efficient to project the points onto a random direction, instead of trying to find the direction with the maximum dispersion. We conclude that random projection trees (rpTrees) can be used as a similarity metric, where they are applied efficiently as described in this paper.

This work can be extended by changing how the pairwise similarity is computed inside the Siamese net. Currently it is done via a heat kernel. Also, one could use other random projection methods such as random projection forests (rpForest) or rpTrees with reduced space complexity. It would be beneficial for the field to see how these space-partitioning trees perform with clustering in deep networks.

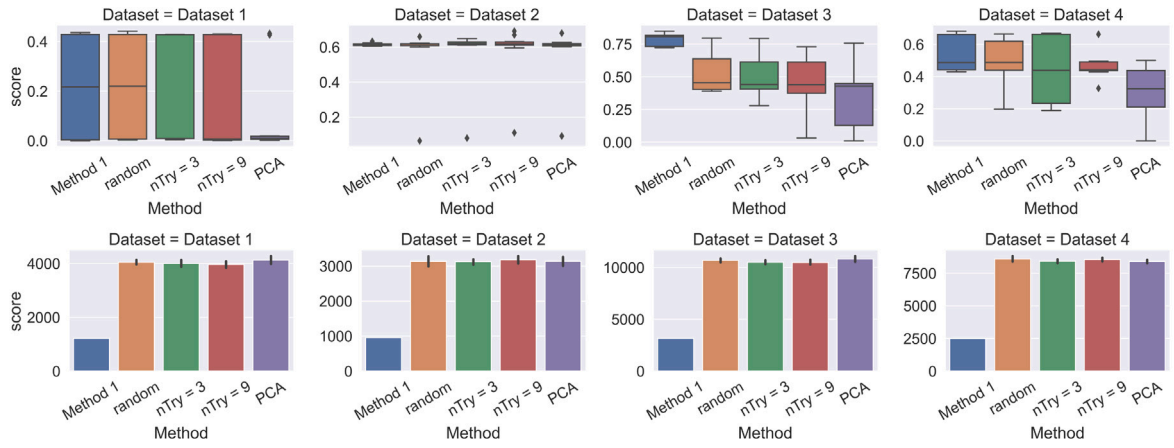


Fig. 11. Experiments with synthetic datasets; `Method 1` is k-nn graph with $k = 2$, other methods use different strategies to pick the projection direction; (top) ARI scores for 10 runs, (bottom) number of total pairs. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

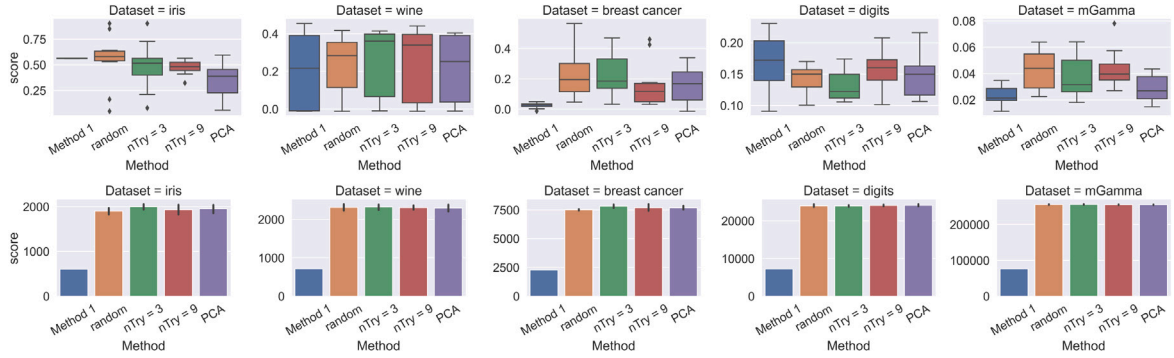


Fig. 12. Experiments with real datasets; `Method 1` is k-nn graph with $k = 2$, other methods use different strategies to pick the projection direction; (top) ARI scores for 10 runs, (bottom) number of total pairs. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

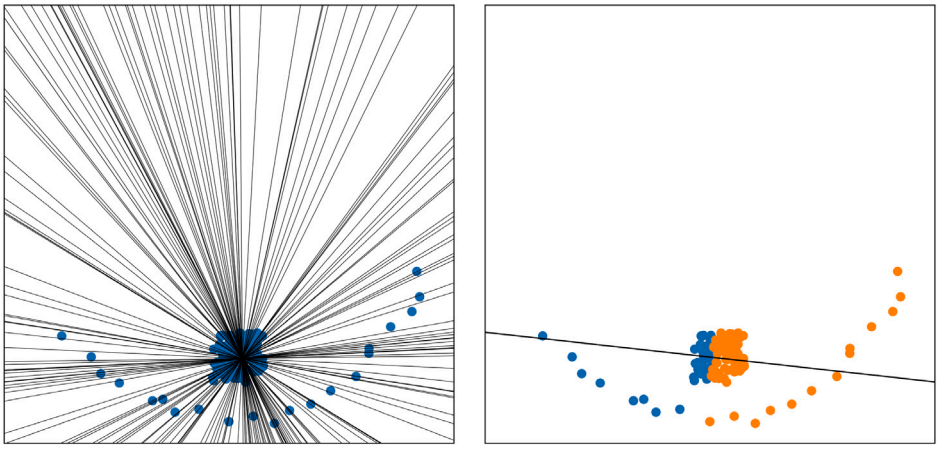


Fig. 13. (left) Sampling 100 projection directions with maximum orthogonality between them; (right) splitting points along the direction with maximum dispersion. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

CRediT authorship contribution statement

Mashaan Alshammari: Conceptualization, Formal analysis, Investigation, Methodology, Project administration, Software, Visualization, Writing – original draft. **John Stavrakakis:** Conceptualization, Formal analysis, Investigation, Methodology, Validation, Writing – review & editing. **Adel F. Ahmed:** Conceptualization, Formal analysis, Supervision, Validation, Writing – review & editing, Funding acquisition. **Masahiro Takatsuka:** Conceptualization, Formal analysis, Supervision, Validation, Writing – review & editing.

Declaration of competing interest

No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work.

Data availability

I have shared the link to my data/code.

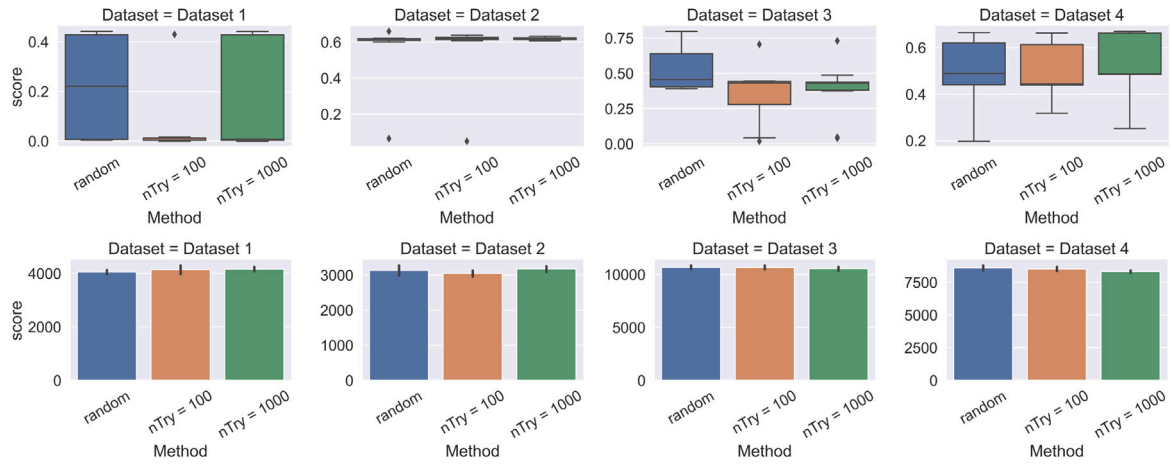


Fig. 14. Experiments with synthetic datasets; `random` represents picking one random projection direction, $nTry = 100$ and $nTry = 1000$ is the number of sampled directions; (top) ARI scores for 10 runs, (bottom) number of total pairs. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

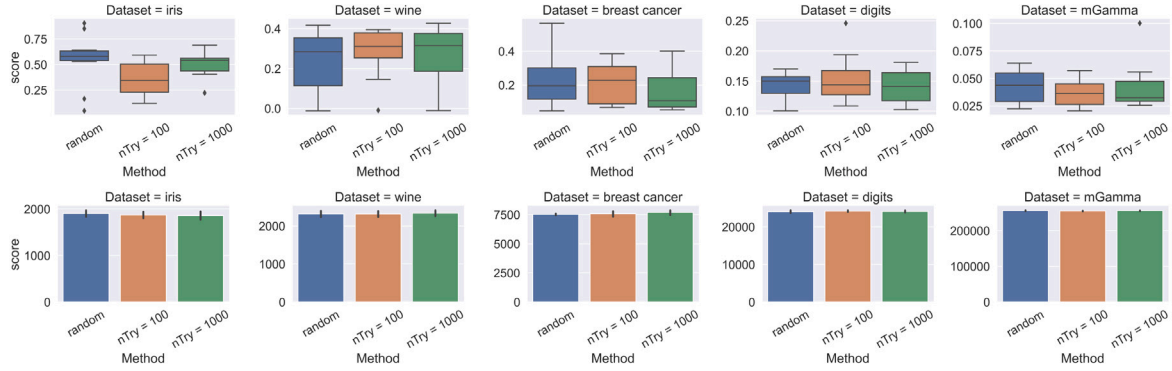


Fig. 15. Experiments with real datasets; `random` represents picking one random projection direction, $nTry = 100$ and $nTry = 1000$ is the number of sampled directions; (top) ARI scores for 10 runs, (bottom) number of total pairs. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

References

[1] Shaham U, Stanton K, Li H, Nadler B, Basri R, Kluger Y. SpectralNet: Spectral clustering using deep neural networks. In: 6th international conference on learning representations, ICLR 2018 - conference track proceedings. 2018, URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85083950872&partnerID=40md5=a1d859bb0bc2080faa2ee428a30201b1.

[2] Bromley J, Guyon I, LeCun Y, Säckinger E, Shah R. Signature verification using a "Siamese" time delay neural network. In: Proceedings of the 6th international conference on neural information processing systems. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.; 1993, p. 737–44.

[3] Dasgupta S, Freund Y. Random projection trees and low dimensional manifolds. In: Proceedings of the fortieth annual ACM symposium on theory of computing. New York, NY, USA: Association for Computing Machinery; 2008, p. 537–46. http://dx.doi.org/10.1145/1374376.1374452.

[4] Freund Y, Dasgupta S, Kabra M, Verma N. Learning the structure of manifolds using random projections. In: Platt J, Koller D, Singer Y, Roweis S, editors. Advances in neural information processing systems, Vol. 20. Curran Associates, Inc.; 2008, URL https://proceedings.neurips.cc/paper/2007/file/9fc3d7152ba9336a670e36d0ed79bc43-Paper.pdf.

[5] Song X, Li J, Cai T, Yang S, Yang T, Liu C. A survey on deep learning based knowledge tracing. Knowl-Based Syst 2022;258:110036. http://dx.doi.org/10.1016/j.knosys.2022.110036, URL https://www.sciencedirect.com/science/article/pii/S0950705122011297.

[6] Zhou J, Huang JX, Hu QV, He L. SK-GCN: Modeling syntax and knowledge via graph convolutional network for aspect-level sentiment classification. Knowl-Based Syst 2020;205:106292. http://dx.doi.org/10.1016/j.knosys.2020.106292.

[7] Kipf TN, Welling M. Semi-supervised classification with graph convolutional networks. In: International conference on learning representations. 2017.

[8] Franceschi L, Niepert M, Pontil M, He X. Learning discrete structures for graph neural networks. In: Proceedings of the 36th international conference on machine learning. 2019.

[9] Yang S, Cai B, Cai T, Song X, Jiang J, Li B, et al. Robust cross-network node classification via constrained graph mutual information. Knowl-Based Syst 2022;257:109852. http://dx.doi.org/10.1016/j.knosys.2022.109852, URL https://www.sciencedirect.com/science/article/pii/S0950705122009455.

[10] Yang S, Verma S, Cai B, Jiang J, Yu K, Chen F, et al. Variational co-embedding learning for attributed network clustering. 2021, http://dx.doi.org/10.48550/ARXIV.2104.07295, arXiv.

[11] Wang Y, Chang D, Fu Z, Zhao Y. Learning a Bi-directional discriminative representation for deep clustering. Pattern Recognit 2022;109237. http://dx.doi.org/10.1016/j.patcog.2022.109237.

[12] McInnes L, Healy J, Melville J. UMAP: Uniform manifold approximation and projection for dimension reduction. 2018, http://dx.doi.org/10.48550/ARXIV.1802.03426, arXiv.

[13] Affeldt S, Labiod L, Nadif M. Spectral clustering via ensemble deep autoencoder learning (SC-EDAE). Pattern Recognit 2020;108:107522. http://dx.doi.org/10.1016/j.patcog.2020.107522, URL https://www.sciencedirect.com/science/article/pii/S0031320320303253.

[14] Huang S, Ota K, Dong M, Li F. MultiSpectralNet: Spectral clustering using deep neural network for multi-view data. IEEE Trans Comput Soc Syst 2019;6(4):749–60. http://dx.doi.org/10.1109/TCSS.2019.2926450.

[15] Wada Y, Miyamoto S, Nakagama T, Andéol L, Kumagai W, Kanamori T. Spectral embedded deep clustering. Entropy 2019;21(8). http://dx.doi.org/10.3390/e21080795, URL https://www.mdpi.com/1099-4300/21/8/795.

[16] Zhang X, Liu H, Wu X-M, Zhang X, Liu X. Spectral embedding network for attributed graph clustering. Neural Netw 2021;142:388–96. http://dx.doi.org/10.1016/j.neunet.2021.05.026, URL https://www.sciencedirect.com/science/article/pii/S0893608021002227.

[17] Jarvis R, Patrick E. Clustering using a similarity measure based on shared near neighbors. IEEE Trans Comput 1973;C-22(11):1025–34. http://dx.doi.org/10.1109/T-C.1973.223640.

[18] Wen G, Zhu Y, Zheng W. Spectral representation learning for one-step spectral rotation clustering. Neurocomputing 2020;406:361–70. http://dx.doi.org/10.1016/j.neucom.2019.09.108, URL https://www.sciencedirect.com/science/article/pii/S0925231220303477.

[19] Zelnik-Manor L, Perona P. Self-tuning spectral clustering. Adv Neural Inf Process Syst 2005;1601–8.
[20] Kim J-H, Choi J-H, Park Y-H, Leung CK-S, Nasridinov A. KNN-SC: Novel spectral clustering algorithm using k-nearest neighbors. IEEE Access 2021;9:152616–27. http://dx.doi.org/10.1109/ACCESS.2021.3126854.
[21] Ester M, Kriegel H-P, Sander J, Xu X. A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of the second international conference on knowledge discovery and data mining. AAAI Press; 1996, p. 226–31.
[22] Yan D, Huang L, Jordan MI. Fast approximate spectral clustering. In: Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining. New York, NY, USA: Association for Computing Machinery; 2009, p. 907–16. http://dx.doi.org/10.1145/1557019.1557118.
[23] Yan D, Gu S, Xu Y, Qin Z. Similarity kernel and clustering via random projection forests. 2019, CoRR abs/1908.10506, arXiv:1908.10506.
[24] Gilbert EN. Random plane networks. J Soc Ind Appl Math 1961;9(4):533–43. http://dx.doi.org/10.1137/0109045.
[25] Barthelemy M. Morphogenesis of spatial networks. Lecture notes in morphogenesis, Springer International Publishing; 2017.
[26] Hubert L, Arabie P. Comparing partitions. J Classification 1985;2(1):193–218. http://dx.doi.org/10.1007/BF01908075.
[27] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine learning in Python. J Mach Learn Res 2011;12:2825–30.
[28] Buitinck L, Louppe G, Blondel M, Pedregosa F, Mueller A, Grisel O, et al. API design for machine learning software: Experiences from the scikit-learn project. In: ECML PKDD workshop: languages for data mining and machine learning. 2013, p. 108–22.
[29] Dua D, Graff C. UCI machine learning repository. University of California, Irvine, School of Information and Computer Sciences; 2017, URL http://archive.ics.uci.edu/ml.
[30] Alshammari M, Stavrakakis J, Takatsuka M. Refining a k-nearest neighbor graph for a computationally efficient spectral clustering. Pattern Recognit 2021;114:107869. http://dx.doi.org/10.1016/j.patcog.2021.107869, URL https://www.sciencedirect.com/science/article/pii/S003132032100056X.
[31] Ram P, Sinha K. Revisiting kd-tree for nearest neighbor search. New York, NY, USA: Association for Computing Machinery; 2019, p. 1378–88. http://dx.doi.org/10.1145/3292500.3330875.
[32] Yan D, Wang Y, Wang J, Wang H, Li Z. K-nearest neighbor search by random projection forests. IEEE Trans Big Data 2021;7(1):147–57. http://dx.doi.org/10.1109/TBDATA.2019.2908178.
[33] Keivani O, Sinha K. Random projection-based auxiliary information can improve tree-based nearest neighbor search. Inform Sci 2021;546:526–42. http://dx.doi.org/10.1016/j.ins.2020.08.054.