Pattern Analysis and Applications (2024) 27:83
https://doi.org/10.1007/s10044-024-01298-5

THEORETICAL ADVANCES

A novel two-stage omni-supervised face clustering algorithm

Sing Kuang Tan[1] · Xiu Wang[2]

Received: 8 September 2022 / Accepted: 14 June 2024 / Published online: 9 July 2024

Abstract
Face clustering has applications in organizing personal photo album, video understanding and automatic labeling of data for semi-supervised learning. Many existing methods cannot cluster millions of faces. They are either too slow, inaccurate, or need a lot memory. In our paper, we proposed a two stage unsupervised clustering algorithm which can cluster millions of faces in minutes. A rough clustering using greedy Transitive Closure (TC) algorithm to separate the easy to locate clusters, then a more precise non-greedy clustering algorithm is used to split the clusters into smaller clusters. We also developed a set of omni-supervised transformations that can produce multiple embeddings using a single trained model as if there are multiple models trained. These embeddings are combined using simple averaging and normalization. We carried out extensive experiments with multiple datasets of different sizes comparing with existing state of the art clustering algorithms to show that our clustering algorithm is robust to differences between datasets, efficient and outperforms existing methods. We also carried out further analysis on number of singleton clusters and variations of our model using different non-greedy clustering algorithms. We did trained our semi-supervised model using the cluster labels and shown that our clustering algorithm is effective for semi-supervised learning.

Keywords Clustering · Semi-supervised learning · Face recognition · Omni-supervised learning · Label propagation · Face embedding

1 Introduction

Face clustering has applications in organizing personal photo album, video understanding and automatic labeling of data for semi-supervised learning. Many existing methods cannot cluster millions of faces. They are either too slow, inaccurate, or need a lot memory. Our method can run on CPU in minutes given the nearest neighbors graph and embedding values. Greedy clustering algorithms such as TC are fast but inaccurate. Non-greedy algorithms such as spectral clustering are slow, use a lot of memory and need to specify accurate number of clusters to produce good result. For large dataset, spectral clustering will lead to out of memory when runs on desktop computer because spectral clustering has to store and process all pairs distance information between all embeddings during clustering. Our method combines best of both worlds which our optimization based clustering consists of both greedy and non-greedy algorithms. Deep learning clustering requires learning, produces noisy results with a lot of singleton clusters whereas our method does not require learning. These deep learning clustering methods cannot scale to large number of faces (in millions), slow, need large amount of memory and accuracy drops at large number of faces.

Clustering is an important step for semi-supervised face recognition. Semi-supervised learning is to learn from both labeled and unlabeled data. The trend in deep learning face recognition models is that the larger the dataset, the better the performance. However large dataset requires more man-hours to label the data. It is difficult to label face classes because it has unlimited number of classes. This helps to circumvent the problem of large data collection and makes training state of the art face recognition model using lesser

✉ Xiu Wang
xiuwang0214@163.com

Sing Kuang Tan
singkuangtan@gmail.com

[1] Department of Industrial Systems Engineering and Management, National University of Singapore, 1 Engineering Drive 2, Singapore 117576, Singapore

[2] School of Computing and Artificial Intelligence, Southwest Jiaotong University, Xi'an Road, Chengdu 611756, Sichuan, China

labeled data possible. Semi-supervised learning is an underexplored area in face recognition. Unlimited number of classes makes fixed classes semi-supervised learning method impossible to use. Face recognition is an open set classification problem which means we can always add more face classes for learning.

Omni-supervised learning is a recently proposed algorithm to label unlabeled data using only a single trained model [1]. Originally it is applied for the human pose estimation problem where the input image is transformed multiple times, fed into the model to produce multiple labels. These labels are then combined and used as labels for semi-supervised learning. We adapt the technique to use on face recognition where the face image is transformed before the face alignment process and flipped before the computation of face embedding. This allows us to compute multiple embeddings and these embeddings are combined to form a more robust embedding that is more accurate for clustering.

The main contributions of this paper are:

- Developed an unsupervised clustering method that is very fast and accurate. It is a two stage algorithm, first a greedy clustering is performed, followed by a non-greedy clustering algorithm to tackle both easy and difficult to separate clusters
- Achieved state of the art performance on large-scale face clustering, surpass supervised deep learning clustering algorithms. Our clustering algorithm has achieved F-measure of 76.30% in 22.75 min for a 5 millions faces of the MS-Celeb-1 M dataset compared to a competing method with F-measure 71.63% in 162.27 min
- Shown that our clustered result can be fed into a face recognition algorithm to do semi-supervised learning

2 Related work

Classical clustering methods (such as K-means, DBSCAN and spectral clustering) are too slow, consume too much memory and have poor accuracy. They are not tuned to work on face recognition embedding distribution. Many of them require to specify number of clusters. Density clustering methods such as DBSCAN have low recall. An unsupervised clustering algorithm (FINCH) [2] using first neighbor relation is not able to scale to large number of face classes in both accuracy and speed.

State of the art deep learning clustering algorithms [3, 4] require training, large memory usage and they are slow. They cannot work with face embeddings of any vector length. So we came out a unsupervised two stage clustering algorithm that does not require to do learning on any dataset and it has few parameters to tune. These deep learning algorithms not only need the nearest neighbor graph between face embeddings to do clustering, they also require the embedding values for clustering. Graph convolutional network (GCN) method [4] will increase and decrease the edge distances of the embeddings so that embeddings from the same class are moved closer to each other and embeddings of different classes are moved further from each other. Then a greedy clustering algorithm is applied to segment the embeddings based on simple increasing distance threshold and breath first search. For another work that uses affinity graph (LTC) [3], it groups up the embeddings into super-vertices, generates cluster proposals, does non-maximal suppression on the proposal clusters and refines the cluster labels within each proposal similar to a Mask-RCNN [5] algorithm. By grouping into super-vertices, their method are more robust against producing small noisy clusters.

Semi-supervised algorithms are generally divided into two categories. One type of semi-supervised learning operates on classification problem with fixed number of classes (closed set classification) [6–15]. Some of them train multiple deep networks and combine output labels from multiple networks to label the unlabeled data [8, 10, 13, 14]. Some use label propagation to spread labels from labeled samples to unlabeled samples [6, 12]. One method uses graph filtering similar to clustering [9]. Some use specialized loss function to propagate labels during deep learning [11, 16]. One very innovative work uses learning speed to determine the labels [17]. Since face recognition has unlimited number of classes, these fixed classes semi-supervised learning methods are not suitable for face recognition and they are difficult to modify to work on face recognition problem.

The second type of semi-supervised learning algorithm is designed to work on variable number of classes (open set classification). One subcategory of this type requires the unlabeled data to be clustered [3, 4] to become labeled data in order to do the learning. A simple model is learned from the labeled data. Then clustering is performed using the trained model on the unlabeled data to do labeling. The combined data from both labeled and unlabeled data is used to train a final model for face recognition.

Another subcategory of this type does not require clustering and labeling is done during learning using a semi-supervised loss function [18]. This approach learns and labels at the same time using a single loss function and does not output clustering labels. Our semi-supervised learning using our clustering algorithm can use existing face recognition deep learning networks and loss functions without the use of complex loss function as in theirs.

Some semi-supervised face recognition algorithms take into the context of how the photos are taken [19–21]. Our method does not need any context information, and can work on random faces crawled from the internet.

CDP [22] is a recently proposed semi-supervised learning algorithm. The disadvantage is it requires to train multiple committee models in order to achieve higher clustering accuracy. However training many models is time consuming. Omni-supervised learning [1] has recently been proposed to generate multiple labels of a single input image by doing multiple transforms of the input image. The different transformations are fed into a single model to produce multiple labels. The multiple labels are then combined similar to how the labels are combined by multiple committee models. This avoids the need to train multiple models. As deep network model has redundancy representation of weights where a slight transformation of the input should result in the same label. If the output label of a transformation is incorrect, it is detected as having different label from other transformations and the labels of multiple transformations can be combined to create a more accurate label. The model is thought to be 'self-ensembled' with different models of the ensemble are expressed when different transformations are applied to its input.

As for the recent deep learning approaches [23, 24] to clustering, clustering is inherently an optimization problem. Deep learning clustering is to try to learn the optimization function of clustering. Deep learning clustering is non-interpretable and explainable, so it does not add knowledge to the science of clustering. Deep learning has drawbacks. It does not scale well to extremely large datasets, so it took very long and consumed a lot of memory to do clustering. An analogy is the use of linear programming or reinforcement learning to solve the Travelling Salesman Problem. Reinforcement learning does not scale well in computational performance to the number of cities.

3 The proposed clustering algorithm

3.1 Overview

Figure 1 shows the framework for our clustering algorithm with the last step doing a semi-supervised face recognition learning. Step 1 is to train a shallow model using only the labeled data. We will use a shallow network to prevent overfitting. Step 2 is to apply multiple transforms to each input face image. These transformed face images are fed into the shallow network to produce multiple embeddings. These embeddings are averaged and normalized to produce the final embedding for each face image to be used for clustering at later stages. The transformed embeddings should complement each other mispredictions. Step 3 is to label the unlabeled data using the TC algorithm [22]. Step 4 is to refine the clusters by splitting or merging the clusters. We will explain that it is very unlikely that the clusters need to merge at this step to produce better result. Therefore in our algorithm, only splitting is carried out at this step. Step 5 is to propagate the labels to neighboring face embeddings that have distances smaller than a threshold. Step 6 is to train the final semi-supervised model using both the labeled and unlabeled data with the cluster labels.

It is true that two-stage clustering, as well as TC clustering and K-Means, have been introduced and adopted in the literature, our approach combines these methods in a novel way to address a specific problem. The combination of both clustering has not been experimented in the literature.

3.2 Omni-supervised model with multiple transformations

Figure 2 shows how four transforms are carried out to produce four different embeddings using only one model in step 2 of our framework. The four transformations are made up of simpler transformations namely left and right flips, align and scale face image up two times then align. These embeddings produced by running different transformations are fed to the same model, later combined using average operation and then normalized. We notice that there are errors in the alignment stage. By scaling up the face image then run an align operation, this returns a slightly different alignment keypoint positions. This helps

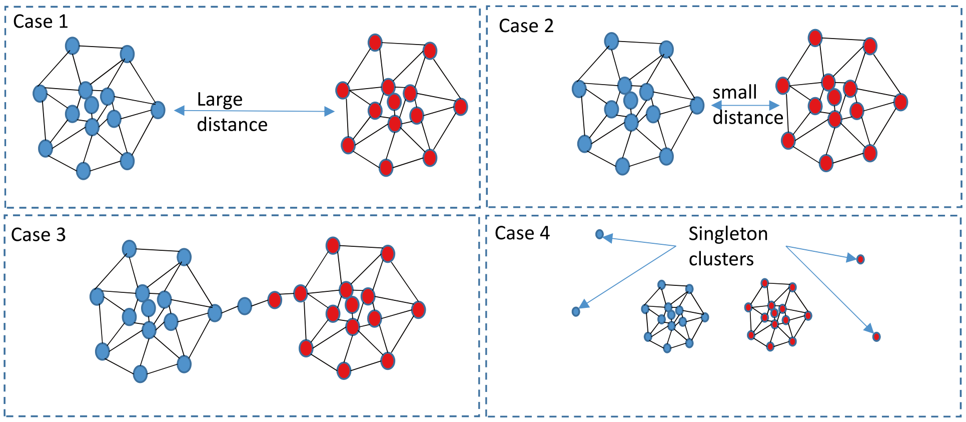


Fig. 1 Our face clustering framework

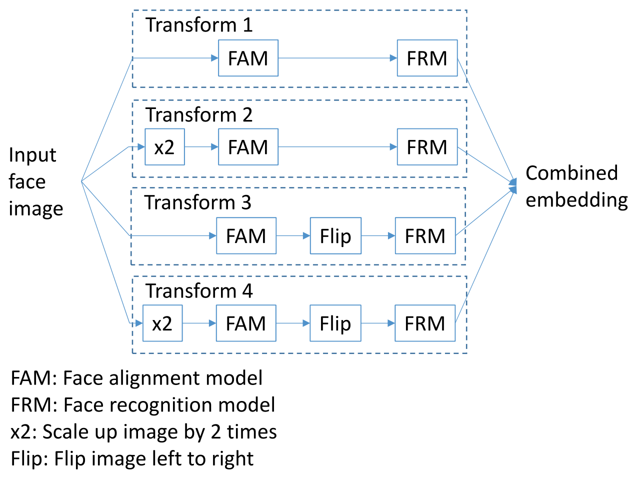


Fig. 2 The four transforms for Omni-supervised learning

as the original alignment without scaling may be incorrect. This averages out the errors in embeddings due to wrong alignment. The left to right flipping also averages out the errors in the embedding. We apply another alignment transformation [25] instead of using the alignment information in the dataset. We use the alignment information in the data only if the alignment transform fails to find an alignment for a face image. We assume the unlabeled data has no alignment information. Other transformations such as rotation can also be used, however we did not investigate them in this work.

These transformations act like image augmentations in deep learning. We have shown that the use of all transformations will lead to the best result. These transformations tackle the face key points misalignment problem, but did not tackle blurry or color distortion due to surveillance camera. Blurry and color distortion problem can be addressed in the future work.

3.3 Greedy clustering and non-greedy cluster splitting

An additional clustering is performed at step 4 to further improve on the clustering result. As the TC is a greedy clustering approach, it will result in clusters that geometrically contain two or more clusters. These clusters can be further split at step 4 by non-greedy clustering algorithm such as K-means, hierarchy clustering, spectral clustering etc. For detail, implementation of our clustering algorithm can be found in the supplementary sections of our paper.

D is a distance matrix where each element d_{ij} represents the distance between embedding i and embedding j, likewise S is a similarity matrix with s_{ij} is a reciprocal of d_{ij},

$$\begin{aligned} d_{ij} &= \|e_i - e_j\| \\ s_{ij} &= 1/d_{ij}. \end{aligned} \tag{1}$$

$S^{(2)} = \delta(S, t)$ is the threshold of the distances s_{ij} in S element-wise, if smaller than threshold is 0, else 1. $s^{(2)}_{ij} = 1$ if embedding i is link to embedding j. The threshold-ed matrix $S^{(2)}$ is the adjacency matix that represents the graph that connects the datapoints. TC clustering is to do transitive closure on the adjacency matrix. TC of the graph is the same as Floyd Warshall Algorithm with path algebra, which the algorithm can be represented in matrix form.

$$\delta((S^{(2)})^2, 0) \tag{2}$$

is the linking of the embeddings that are 2 edges apart where the matrix $S^{(2)}$ is raised to the power of 2. This is the addition of edges that are connected by a path of 2 edges in the adjacency matrix.

$$S^{(3)} = \delta((S^{(2)})^\infty, 0) \tag{3}$$

is the linking of embeddings that are many (large finite positive) edges apart, or we can say it is transitive closure of the edges using path algebra. The matrix $S^{(2)}$ is raised to the power of ∞. To subdivide the clusters using non-greedy clustering, let

$$\begin{aligned} &\min_T \{|T - S^{(3)}|\}_{\geq 0} \\ &\quad \text{such that} \quad S^{(4)} = \delta(S, T) \\ &\quad S^{(4)} \leq S^{(3)} \\ &\quad v_i = \lambda_i(S^{(4)}) \\ &\quad v_i \in \{0, 1\} \\ &\quad S^{(4)}\mathbf{1} < t_{max\_size}. \end{aligned} \tag{4}$$

$\mathbf{1}$ is a vector of ones and $S^{(4)}$ is a matrix of thresholds which is a block diagonal matrix, each block represents 1 cluster. Because each cluster should be fully conntected with every other datapoint in the cluster. Basically it means that we will repeatedly breaks down a large cluster by increasing the threshold until the size of each cluster is below e.g. $\mathbf{t}_{max\_size} = 600$. Note that different clusters will have different thresholds. In equation 4, the matrix S is thresholded by a matrix of thresholds T where each individual element of $S^{(4)}$ has a different threshold. $v_i = \lambda_i(S^{(4)})$ returns a eigen vector i of matrix $S^{(4)}$ multipled by its eigen value. The ones in v_i represents which datapoints (indexed by the ones in the vector) are in cluster i. The clustering result is in $S^{(4)} = \delta(S^{(3)}, T)$.

Clustering result of TC and non-greedy clustering are represented in an adjacency matrices in $S^{(3)}$ and $\delta(S^{(3)}, T)$. Examples of how the matrices will look like is shown in Figs. 3 and 4. In the examples, $S^{(3)}$ clustered into 2 clusters with data points in each clusters fully connected with each

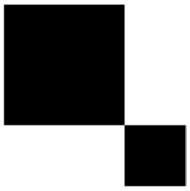


Fig. 3 Example of adjacency matrix with 2 clusters after TC

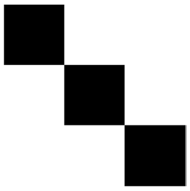


Fig. 4 Example of adjacency matrix with 3 clusters after non-greedy clustering

other by TC. $\delta(S^{(3)}, T)$ further breaks down the cluster into 3 clusters so that each cluster is below a maximum size by non-greedy clustering. $min_T|T - S^{(3)}|_{>=0}$ is to maximise the size of each cluster as much as possible, but the clusters should be subdivision of the TC clusters as shown in $S^{(4)} \leq S^{(3)}$. $|T - S^{(3)}|_{>=0}$ is the same as $|max(T - S^{(3)}, 0)|$ where we sum up the difference of T and $S^{(3)}$ when T is larger than $S^{(3)}$.

The TC clustering algorithm consists of two parts. First a 15 nearest neighbors of all face embeddings of unlabeled data are computed and these neighbors are used to create a graph that connects the nearby embeddings if their similarities are above a threshold. Then the graph is partitioned into connected components and each component forms a cluster. This is the first step of this clustering algorithm. Next the connected components or clusters are repeatedly broken down if the cluster sizes are larger than a specified size (e.g. 600 embeddings) and the similarity threshold is increased by a small amount to break down the clusters into smaller connected components or clusters.

Ideally, the face embeddings of the face recognition model are trained to have a fixed distance between pairs of face embeddings from different face identities and a very small distance between pairs of face embeddings from same face identity. A greedy algorithm with a fixed similarity threshold that links up embeddings above the similarity threshold is able to produce the clusters. This is true for small number of face embeddings in the test set. However for large number of faces in the test set, the distances between pairs of face embeddings from different identities may be higher than a fixed similarity threshold. This is because many of these new faces in the test set are very different from the training set, and the face recognition algorithm is confused whether some pairs of faces are from the same or different persons using a fixed similarity threshold. The greedy algorithm needs to use adaptive similarity threshold (which is increased gradually over time) to further break down the face embedding clusters if they are larger than a predefined number of face embeddings (e.g. 600).

After the face embeddings are broken down into clusters by TC, some of these clusters are still large and contain two or more face classes. For these clusters of face embeddings, a greedy clustering algorithm using connected components is not able to break them and a non-greedy algorithm at step 4 is needed to break them down. The non-greedy algorithm looks at all pairs of distances between the face embeddings in these large clusters and still able to figure out the sub-clusters within these large clusters using between class distances and within class distances. In practice, if the number of face embeddings in a cluster is smaller than a predefined number (e.g. 150), we ignore this cluster and it will not be further split by our non-greedy algorithm in step 4 into smaller clusters as the cluster relationship is unsure for small number of embeddings. As the TC algorithm is greedy when it does clustering, the algorithm will always separate easy to separate clusters and therefore the algorithm will very unlikely oversplit the datapoints (there will be very unlikely a case where two or more resulted clusters are part of an actual face class). We have verified experimentally that the TC algorithm has high recall and low precision, therefore it is unlikely that it will oversplit the datapoints. The difficult clusters will be left for the non-greedy clustering algorithm at step 4 to tackle.

Non-greedy clustering algorithm can be implemented using k-means algorithm. The loss function for the k-means algorithm is

$$J(C_{ij}) = \sum_{(i,j)\in\{(m,n)\|C_{mn}=1\}} \|x_i - u_j\|^2 \tag{5}$$

where x_i is the embedding i and u_j is the centroid of cluster j. x_i belong to cluster u_j. Non-greedy clustering algorithm is based on centroid clustering.

If we use the non-greedy algorithm directly on the face embeddings without doing TC greedy clustering, this will result in poor clustering performance. It is because it is difficult to determine the number of clusters and cluster size of each cluster using non-greedy clustering algorithm. Table 1 shows the results of spectral clustering directly on the face

Table 1 Spectral clustering results with different number of clusters

Spectral clustering	F-measure (%)
With 2000 clusters	84.56
wWth 2577 clusters (actual # of classes)	**97.22**
With 3000 clusters	93.83

Bold indicates the best performance/results

embeddings with 2577 (exact ground truth number of clusters), 2000 and 3000 clusters. We can see that if we wrongly estimate the number of clusters by a small fraction, the clustering performance (F-measure) will differ a lot. Beside that, non-greedy clustering algorithm looks at all pair distances of face embeddings, it will take up a lot of memory and makes it impossible to cluster millions of faces. For a small set of about 600 face embeddings, greedy clustering algorithm will be able to execute very fast using small amount of memory.

Figure 5 shows the cascade clustering process of our algorithm. A quick greedy clustering algorithm will split easy to separate large clusters (each may contain a few subclusters) and the non-greedy clustering algorithm will further decompose these large clusters into small clusters.

Figure 6 shows the types of clusters that exist in the face embeddings. For case 1, a simple fixed threshold is able to separate the clusters. For case 2, although the fixed threshold cannot separate them, the between class distances are larger than within class distances. By increasing the similarity threshold, the greedy clustering algorithm is still able to separate them. For case 3, if there exists a bridge between two clusters, then the greedy clustering algorithm will not be able to separate them. A non-greedy clustering algorithm such as K-means can be used to separate them. For case 4, although the two clusters can be split by a threshold and greedy clustering algorithm, some small clusters are produced as a side effect of the algorithm. These small clusters (e.g. cluster with only one embedding) can be merged to the nearest clusters using step 5 of our framework. For more information on the distance distribution of each case can be found in the supplementary section.

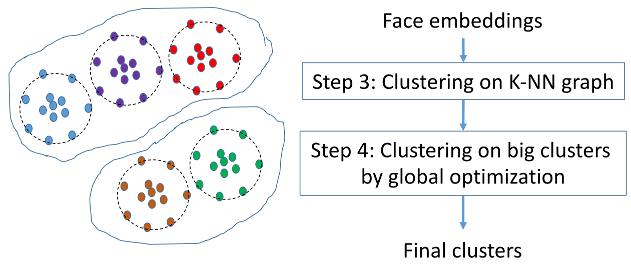

Face embeddings
Step 3: Clustering on K-NN graph
Step 4: Clustering on big clusters by global optimization
Final clusters

Fig. 5 Our two stage clustering algorithm

3.4 Label propagation of remaining unlabeled face embeddings

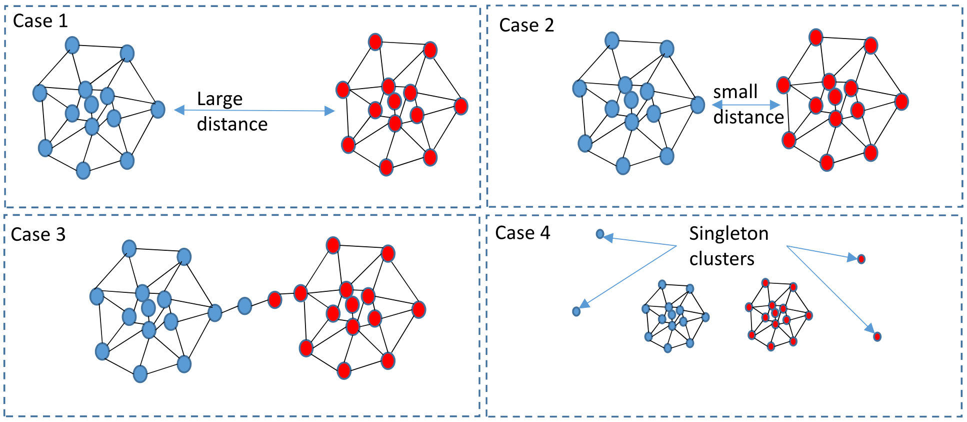
Label propagation is performed at step 5 of our algorithm. The TC algorithm will label some face embeddings as noise (unlabeled). These face embeddings each forms a singleton cluster with one embedding in size. We can also change the labels of small clusters (e.g. size$\leq$ 3) same as singleton clusters as these labels are noisy. They are separated as clusters by the TC clustering algorithm. These singleton clusters are far away from every other face embeddings and so they are labeled as noise. These noisy face embeddings can be ignored during training of the final face recognition model after the unlabeled data is labeled as they make up a small proportion of the total number of unlabeled face embeddings. But for clustering purposes, we can assign them to the nearest clusters if their distances are smaller than a threshold. This will improve the overall F-measure of the clustering result when compared to the ground truth labels of the unlabeled faces. These singleton clusters may be formed during the greedy clustering process and they most likely belong to the nearest clusters.

For a small cluster k smaller a certain size,

$$t_{ij} = \delta(C(i), k) \times (1 - \delta(C(j), k)) \times \delta_{\text{top } 3}(i,j) \times c_j^{(size)}$$
$$c_k = C(arg\,max_j \bigcup_{C(i)=k} s_{ij}) \tag{6}$$

where t_{ij} is an element of a matrix, $t_{ij} > 0$ if embedding i is in cluster k and embedding j is not in cluster k and t_{ij} has the cluster size of where embedding j is located with additional top3 constraint, else it is 0. $\delta_{\text{top } 3}()$ returns a 1 if j is top 3 nearest embedding to embedding i (in terms of distance), else it is 0. $\delta(C(i), k)$ and $1 - \delta(C(j), k))$ make sure that the

Fig. 6 Four types of clusters



matrix is zero when either embedding i does not belong to cluster k or embedding j belongs to cluster k. $c_j^{(size)}$ is the size of cluster that contains embedding j. The function $C()$ returns the cluster id of embedding i or j. c_k is the cluster id where the cluster k is finally assigned to. Basically it means that considering all the embeddings in small cluster k, find the largest cluster that the cluster k is connected to based on top 3 distances of each embeddings and merge cluster k with the largest cluster.

The labels are propagated synchronously to the unlabeled embeddings by finding the most frequently occurring labels of the 3 nearest neighbors of these unlabeled embeddings. Note that the unlabeled embeddings that are far away from their nearest embeddings (e.g. further than a 0.4 threshold) are ignored and new labels will not be assigned to them.

4 Experiment results

In this section, we carried out experiments to validate the effectiveness of our clustering algorithm. We tested our algorithm on the IJB-B 1845 [26] data and MS-Celeb-1 M [27] data which are commonly used by the face clustering community for validation. We used the Folkes and Mallows F-measure [28–30] to evaluate the pairwise performance of clustering,

$$\text{Avg Recall} = \sum_{i,j} N_{i,j} / \sum_{i} N_i \tag{7}$$

$$\text{Avg Precision} = \sum_{i,j} N_{i,j} / \sum_{j} N_j \tag{8}$$

$$\text{Avg F-Measure} = \frac{2 \times \text{Avg Recall} \times \text{Avg Precision}}{\text{Avg Recall} + \text{Avg Precision}} \tag{9}$$

where N_j is the number of possible pairs of embeddings in cluster j and N_i is the number of possible pairs of embeddings in class i. For class i and cluster j, N_{ij} is the number of possible pairs of embeddings of class i in cluster j. Class i is the ground truth label of an embedding and cluster j is a cluster label from the clustering algorithm.

4.1 Omni-supervised clustering results

In this subsection, we use 0.5 million data partition of MS-Celeb-1 M dataset from github site [30] to investigate on the use of different transformations of our omni-supervised face clustering.

Table 2 shows that using two weak transformations to generate two embeddings, then combine the embeddings leads to better result than using each original embedding itself. The two weak transformations are 3×3 median filter and scale down by 4 pixels. They are called weak transformations because they produce embedding values that produce slightly higher performance on test data compared to using the original image without tranformation. These two transformations are weaker than using the transformations of original image and original image flipped left to right. However the combined of the two transformations embeddings (by averaging) leads to better clustering result than using only each transformation. This is in analogy to the idea of weak classifiers where a combination of the classifiers leads to a stronger classifier.

Table 3 shows that by applying two or four best transformations for omni-supervised clustering, we are able to achieve 1% and 5% improvements respectively. This is considered large improvement using only one trained model instead of multiple trained models as in CDP algorithm. The results shown in this subsection used embeddings generated

Table 2 Combine two weak transformations becomes a strong transformation

Performance (F-measure)	Transformation 3×3 median filter (%)	Transformation: Scale down by 4 pixels on width and height (%)	2 Transformations: Scale down by 4 pixels on width and height 3×3 median filter (%)
Step 3 [22]	76.62	76.25	**78.84**
Step 4	80.94	81.63	**83.06**
Step 5	86.95	86.49	**87.82**

Bold indicates the best performance/results

Table 3 Combine four input transformations to generate robust embeddings for clustering

Performance (F-measure)	No transformation (%)	Apply 2 transformations (flip face image left and right) (%)	Apply 4 transformations (as in previous section) (%)
Step 3 [22]	75.45	77.20	**81.61**
Step 4	78.70	81.91	**87.17**
Step 5	86.17	87.82	**91.80**

Bold indicates the best performance/results

using our trained model on labeled data. We did not use existing embeddings from the github site as they do not provide their deep network model for us to experiment with different input transformations. Our omni-supervised transformations can act like multiple committee models, without the need to train multiple models.

4.2 Compare with CDP multiple committee models

We experimented with CDP algorithm using their committee models instead of using our omni-supervised transformations to generate and combine the embeddings. Then the cluster results from CDP are further refined using our step 4 cluster splitting and step 5 label propagation algorithms. We used the embeddings from the CDP github site [29]. From Table 4, we can see that on 200k MS-Celeb-1 M dataset, using our clustering method with 1 committee model (top right entry in the table) has close to the same performance as using CDP with 4 committee models (bottom left entry in the table). Using 4 committee models requires much more training time and therefore it is not worth it to train 4 models to cluster this small dataset. Using 4 committee models with our clustering algorithm (bottom right entry in the table) has reached almost the same F-measure as spectral clustering (97.22%, see Table 1) with actual number of ground truth clusters specified as input.

4.3 Compare with state of the art clustering algorithms

Table 5 shows the step 3, 4 and 5 results of our framework on the 1.7M and 5 M data. The 1.7 and 5 millions faces datasets are partitions of a large MS-Celeb-1 M dataset provided in github site [30]. The embeddings are also downloaded from this github site. Our algorithm has significant performance improvement over the CDP algorithm. Step 5 has made some improvement to step 4 of our framework. The timings shown in this table exclude the time taken to find the 15 k-nearest neighbors graphs for the TC algorithm in step 3. It takes 15.97 and 59.83 min to compute the 15 k-nearest neighbors graph for 1.7 and 5 millions data respectively. The k-nearest neighbors graph computation is a bottleneck in the clustering algorithms, which is also needed for the deep learning clustering algorithms. In fact, the deep learning clustering

Table 5 Clustering results on 1.7 and 5 millions faces of the MS-Celeb-1 M dataset

Step	1.7M data		5 M data	
	F-measure (%)	Time taken (min)	F-measure (%)	Time taken (min)
Step 3 [22]	67.34	0.92	63.85	4.67
Step 4	78.98	2.86	75.38	12.64
Step 5	**81.31**	**0.26**	**76.30**	**5.44**

Bold indicates the best performance/results

algorithms LTC and GCN require 80 and 200 nearest neighbors respectively.

In Tables 5 and 6, we use only one face recognition model without omni-supervised tranformations and without multiple models as in the multiple committee models case in previous subsection. We used the embeddings provided at the LTC paper github site [30] and the GCN paper github site [31]. We used spectral clustering for step 4 of our framework. Spectral clustering is applied only to clusters larger than a predefined number (e.g. 150 embeddings). We select number of clusters in spectral clustering using the number of eigenvalues greater than a threshold, with a maximum of 5 clusters. The spectral clustering is repeated again if the broken down clusters are still larger than the predefined number.

From Table 6, we can see that our clustering algorithm has outperformed LTC [3] and GCN [4] clustering algorithms even on the large dataset of 1.7 and 5 millions faces. Both of these algorithms use deep learning network. Our algorithm is much faster. This enables the user more time to vary with the parameters of our clustering algorithm to further improve the performance. Note that all clustering algorithms are run on CPU and the timings of the deep learning clustering algorithms are inference times only without training times. The GCN, LTC and our clustering timings in the table do not include nearest neighbors computation as k-nearest neighbors are computed separately. As for the FINCH method, it requires only one nearest neighbor and it is computed together with the clustering process. We can also see that GCN algorithm works well on the small 66k IJB-B 1845 data but it performs poorly on the 1.7M data. This algorithm does not scale well to large dataset. FINCH algorithm [2] is a hierarchy clustering technique which

Table 4 Clustering results on 200k dataset

Performance (F-measure)	Step 3 [22] (%)	Step 4 (%)	Step 5 (%)
1 Base model and 1 committee models, use voting	88.17	94.66	95.19
1 Base model and 2 committee models, use voting	91.70	95.62	96.35
1 Base model and 4 committee models, use voting	92.86	95.226	96.67
1 Base model and 4 committee models, use mediator	**95.59**	**96.10**	**97.05**

Bold indicates the best performance/results

Table 6 Our clustering results comparing to the state of the art algorithms

Clustering algorithms	IJB-B 1845 data		1.7M MS-Celeb-1 M data		5 M MS-Celeb-1 M data	
	F-measure	Time taken	F-measure	Time taken	F-measure	Time taken
FINCH [2]	26.81%	1.17 mins	50.489%	9.7 mins	35.78%	34.96 mins
GCN [4]	67.72%	2.09 mins	15.72%	52.48 mins	3.25%	150.01 mins
LTC [3] (0.7,0.75)	0.4296%	1.37 mins	75.72%	50.67 mins	71.63%	162.27 mins
LTC [3] (0.6, 0.65, 0.7, 0.75)[a]	–	–	77.63%	–	72.77%	–
Ours	**82.13%**	**0.07 mins**	**81.31%**	**4.04 mins**	**76.30%**	**22.75 mins**

Bold indicates the best performance/results

[a]Taken from github site [30] "mins" means minutes

Table 7 Compare our clustering algorithm (with and without omni-supervision) and the best algorithm VEGCN [32]

Clustering algorithms	F-measure (%)
VEGCN [32]	75
Ours without Omni-supervision	84.55
Ours with Omni-supervision	**86.29**

Bold indicates the best performance/results

Table 8 Percentage of singleton clusters by different clustering algorithms on the 1.7M MS-Celeb-1 M data

Clustering algorithms	% Of singleton clusters	# Of singleton clusters
FINCH [2]	0	0
GCN [4]	87.35	1,520,191
LTC [3] (0.7,0.75)	15.71	273,466
Ours	**1.82**	**31,692**

Bold indicates the best performance/results

returns a set of different numbers of clusters. By choosing the best number of clusters and compute the F-measure, our clustering algorithm still outperforms FINCH. LTC algorithm performs poorly on IJB-B 1845 data as the algorithm is not trained to cluster effectively on that data.

Table 7 compares the best clustering algorithm VEGCN [32] with our algorithm with and without omni-supervision. It has clearly shown that our algorithm outperforms VEGCN even without omni-supervision. The VEGCN algorithm reported here is slightly less than the original paper as we do not know the exact hyperparameters and we reduced the length of each embeddings from 512 dimensions to 256 dimensions so that the embeddings can be input into the deep learning network of VEGCN. The embeddings are reduced from 512 to 256 dimensions simply by adding the first 256 dimensions with the next 256 dimensions and normalizing the resultant embeddings to unit length.

Table 8 shows the percentage of singleton clusters labeled by the different clustering algorithms. Although FINCH has

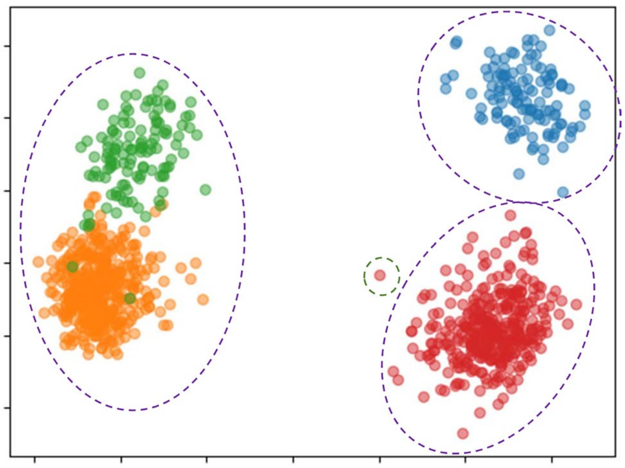


Fig. 7 Example of embeddings distribution of four face identities in first two principal components space

0 singleton cluster, it has poor clustering performance and it is not able to identify noisy face images in the data. We can see that our clustering algorithm is more robust, produces lesser singleton clusters and much higher clustering F-measure compared to the deep learning clustering algorithms. Higher clustering performance is needed for semi-supervised learning.

Figure 7 shows an example of ground truth cluster distribution of four classes of face embeddings. The purple dotted circles are the large clusters returned by step 3 (greedy clustering) of our clustering algorithm. These large clusters can be further broken down into small clusters by step 4 (non-greedy clustering). The green dotted circle is a singleton cluster created by first stage of our clustering algorithm which can be re-merged to the nearest cluster at step 5.

4.4 Different variations of our clustering algorithm

Table 9 shows the results of our clustering algorithm at step 5 if we assume clusters of sizes smaller than or equal to 3 instead of 1 as noisy clusters. We can see that this

Table 9 Clustering performance of our algorithm with assumption of noisy clusters of different sizes

Noisy cluster size	F-measure for 1.7M MS-Celeb-1 M data (%)	F-measure for 5 M data (%)
1	82.48	78.54
2	83.63	**79.17**
3	**83.76**	79.11

Bold indicates the best performance/results

assumption helps to improve the recall and therefore also improves the F-measure of the clustering results.

We used the embeddings provided at the LTC paper github site [30]. Table 10 investigates the use of different non-greedy clustering algorithms in step 4 in terms of execution speed and F-measure performance. We used squared error,

$$SE = \frac{1}{N}\sum_{1}^{N} \|x_i - C(x_i)\|^2 \tag{10}$$

to select the number of clusters for K-means, hierarchy clustering and Birch algorithms. N is the number of face embeddings, x_i is face embedding i and $C(x_i)$ is the centroid of cluster of face embedding x_i. In this experiment, a face embedding cluster is repeatedly broken down by 2 if it contains 2 or more clusters. A cluster is broken down by 2 if

$$SE_2 < SE_1/1.1 \tag{11}$$

Table 10 Clustering results of our algorithm using different non-greedy clustering algorithms at step 4 on the 1.7M MS-Celeb-1 M data

Clustering algorithms	F-measure (%)	Time taken (min)
K-means (init=10)	78.64	1.37
K-means (init=1)	78.42	0.6
Hierarchy clustering (ward)	78.88	0.76
Hierarchy clustering (single)	68.90	0.43
Hierarchy clustering (complete)	78.41	0.76
Hierarchy clustering (average)	77.80	0.75
Hierarchy clustering (weighted)	77.75	0.77
Hierarchy clustering (centroid)	68.30	0.42
Hierarchy clustering (median)	68.26	0.42
DBSCAN (eps=0.75, min_samples=2)	72.20	2.02
Birch (branching_factor=50)	75.41	0.93
Spectral clustering	78.98	2.86
Hierarchy clustering (ward with selection of number of clusters)	**79.27**	**0.73**

Bold indicates the best performance/results

where SE_1 and SE_2 are the square errors when using one and two clusters respectively. This method is similar to elbow method to select number of clusters, where we continue to increase the number of clusters if there is large decrease in error. DBSCAN algorithm needs to specify eps (neighbourhood distance) and F-measure is sensitive to this eps parameter. Spectral clustering selects number of clusters by looking at the eigenvalues. It looks at how many of these eigenvalues are larger than a threshold. For detail information of which python package is used to implement each type of non-greedy clustering algorithm can be found in the supplementary section of our paper.

Using K-means with 1 random initialization is much faster than 10 random initializations with slight loss of F-measure. K-means is the fastest of all algorithms. DBSCAN is sensitive to the choice of eps parameter. Result will differ a lot for slight change in the parameter. Hierarchical clustering works best if we use the 'Ward' merging criteria. 'Ward' is also the best overall algorithm. 'Single' criteria leads to greedy merging and has tendency to form long chain cluster, therefore leads to poor performance. 'Complete', 'Average' and 'Weighted' are slightly inferior to 'Ward', although they are similar to 'Ward'. 'Centroid' and 'Median' perform poorly as they disregard the sizes of the clusters when they try to merge clusters. Spectral clustering is too time consuming. Although 'Ward' has the best performance and is slightly better than spectral clustering, we think that spectral clustering is theoretically better which can partition non-circular and irregular shaped clusters. Therefore for Table 6, we choose to use spectral clustering.

We can also select the number of clusters in 'Ward' hierarchy clustering by using the dendrogram (see row 'Hierarchy clustering (ward with selection of number of clusters)' in Table 10). We search between 1 to 14 clusters and cut the dendrogram where the distance between the nearest split and merge operations is larger than 1.1 using largest number of clusters. The clusters are repeatly broken down for a few iterations. Using this approach, we have achieved F-measure of 79.27% in 0.73 min. It is about 0.4% more accurate than using ward with repeated splitting into 1 or 2 clusters at each iteration, but has about the same speed compared to repeated splitting.

Table 11 shows the optimal clustering results at step 4. This is done by using the cluster results at step 3, then

Table 11 Optimal clustering results at step 4 on the 1.7M MS-Celeb-1 M data

Optimal clustering algorithms	F-measure (%)
Consider clusters of size> 150	80.58
Consider all clusters	**82.44**

Bold indicates the best performance/results

split them according to the ground truth at each cluster to obtain the optimal results. If we do not do additional clustering on the clusters at step 4 that are smaller than or equal to a predefined number (150 samples), the optimal F-measure is 80.58%. Our result is 1.3% lower than this optimal result. If we do additional clustering on all clusters at step 4, the optimal F-measure is 82.44%. Our result is 3.2% lower than this optimal result. As for small clusters at step 3, it is difficult to obtain accurate clustering of them at step 4 so these small clusters are ignored at step 4.

The value of 150 is chosen because the number of images in an identity Celeb-1 M is about 100 images, which is slightly less than 150. 600 is chosen for transitive closure because it is a multiple of 100. So that after transitive closure clustering, there is still room for K-means for further fine-tuning breakdown into smaller clusters of the clusters.

4.5 Semi-supervised face recognition results

We trained an initial 14 layers shallow model which is a modified version of ResNeXt [33] and used it to label the unlabeled data. Then a final model using the ResNeXt 50 layers model is used to train on all the labeled and unlabeled data. Note that the face identities in the labeled and unlabeled data can overlap. We can use separate classifier heads to each part of the data to overcome the problem of data overlap. In our case, we assume that the identities in the labeled and unlabeled data do not overlap so only one classifier head using ArcFace loss function is used. The results are shown in Table 12. Supervised model on labeled data is to train a model on the labeled data only. Supervised model on all data is to train on both labeled and unlabeled data assuming we have the labels of unlabeled data. We can see that our semi-supervised model achieves good performance (almost the same identification performance on the MegaFace dataset) compared to fully supervised model using ground truth labels for the unlabeled data for training. We use the labeled data and unlabeled data provided in the github site [30].

Table 12 Semi-supervised face recognition results

Models	Identification rate on MegaFace dataset (%)
Supervised model on labeled data only	62.74
Supervised model on all data	76.47
Our semi-supervised model	74.94
Our semi-supervised model with removal of small clusters with size$\leq$ 4	**76.22**

Bold indicates the best performance/results

Our semi-supervised model with removal of clusters of size$\leq$ 4, has achieved nearly the same identification rate as our supervised model on whole data validated using the MegaFace dataset. Without removal of small clusters of size$\leq$ 4, our semi-supervised model result is clearly less than our model with removal of small clusters. Our semi-supervised model has greatly outperformed supervised model on labeled data only by more than 10%.

5 Conclusion

In this paper, we have shown that combining two weak tranformations leads to strong clustering result, in similar analogy to combining weak classifiers leads to strong classifier. Our omni-supervised clustering has led to 5% improvement in our clustering algorithm compared to the case with no transformation. For clustering of 200k dataset, we have shown that using one committee model has about the same performance of using four committee models if step 4 and 5 step of our clustering are performed after step 3 TC. We have tried substituting step 4 of our clustering algorithm with many other classical clustering algorithms and have shown that K-means, hierarchy clustering (ward) and spectral clustering have performed similarly well. We have shown that if our step 4 clustering is perfect, it is only 4% away from our hierarchy clustering (ward) result. We have trained a semi-supervised model using both labeled and unlabeled data (labeled by clustering). It has almost the same performance with the model trained on all ground truth data without stripping the labels from unlabeled data.

Although our method is an optimization technique, our method is interpretable and explainable than the deep learning approach, which can be run in much shorter time using less memory. In the future work, we will add in convex optimization clustering to perfect the art of semi-supervised learning.

Appendix 1: Implementation details of our clustering algorithm

We use the TC clustering algorithm as the backbone algorithm in our clustering algorithm. Our clustering algorithm first does a greedy clustering (TC clustering), next a non-greedy algorithm and lastly propagation of cluster labels from labeled embeddings to unlabeled embeddings (developed by us). Our whole algorithm is written entirely in python and we released the codes in our github site (state the site). We used the data from the papers [3] and [4]. All clustering experiments are ran on a desktop machine with Intel Core i7-7700 CPU @3.60GHz. We use only one core (without any parallel processing) in our experiments. Our clustering algorithm is simple and can run on most CPU using only single core.We use only one core (without any parallel processing) in our experiments. Our clustering algorithm is simple and can run on most CPU using only single core. We have uploaded our codes to github. It can be accessed through the link https://github.com/singkuangtan/face-clustering.

For Table 6, we use embeddings trained using softmax loss function and as for semi-supervised learning (Table 12), we use embeddings trained using ArcFace loss function.

Appendix 2: Python functions and packages

Table 13 shows the python functions and packages we use for the experiments.

Appendix 3: Relationship of the distances in four cases

We begin by describing a set of properties,

$$\begin{aligned} d_{min}(C_0, C_1) &= \min_{i\in C_0, j\in C_1, n(i,j)=1} \|e_i - e_j\| \\ d_{max}(C_0) &= \max_{i\in C_0, j\in C_0, n(i,j)=1} \|e_i - e_j\| \\ d_{max}(C_1) &= \max_{i\in C_1, j\in C_1, n(i,j)=1} \|e_i - e_j\| \\ d_{avg}(C_0, C_1) &= \frac{1}{|C_0||C_1|} \sum_{i\in C_0, j\in C_1} \|e_i - e_j\| \\ d_{avg}(C_0) &= \frac{1}{|C_0|} \sum_{i\in C_0} \|e_i - \sum_{j\in C_0} e_j\| \\ d_{avg}(C_1) &= \frac{1}{|C_1|} \sum_{i\in C_1} \|e_i - \sum_{j\in C_1} e_j\| \end{aligned} \tag{12}$$

where C_0 is the set of embedding indices for cluster 0 and likewise C_1 is the set of embedding indices for cluster 1. $n(i,j) = 1$ if embedding i and j are neighbors else it is a 0. e_i or e_j is an embedding with index i or j.

For Case 0, a greedy clustering algorithm with a large threshold can separate the two clusters. Mathematically, it is

$$d_{min}(C_0, C_1) >> max(d_{max}(C_0), d_{max}(C_1)) \tag{13}$$

where max is a maximum function of the two input values and >> means much greater than (by a few times).

For case 1, a greedy clustering algorithm can separate the two clusters, but the gap between the clusters is smaller and therefore a smaller threshold is used. Mathematically, it is

$$d_{min}(C_0, C_1) > max(d_{max}(C_0), d_{max}(C_1)). \tag{14}$$

For case 2, there is a bridge that connects nearest neighbor embeddings from the two clusters. Therefore no threshold using a greedy algorithm is able to separate them. However, the mean interclass distance is still larger than the mean intraclass distance. This property enables the clusters to be separated by non-greedy clustering algorithm such as Kmeans. Mathematically, it is

$$\begin{aligned} &d_{min}(C_0, C_1) < max(d_{max}(C_0), d_{max}(C_1)) \\ &d_{avg}(C_0, C_1) > d_{avg}(C_0) \\ &d_{avg}(C_0, C_1) > d_{avg}(C_1). \end{aligned} \tag{15}$$

For case 3, although the singleton cluster 0 is separated from main cluster 1 using a threshold and greedy clustering algorithm, the ground truth class of cluster 0 is the same as cluster 1 due to random outlier noise. So the singleton cluster 0 should be combined with cluster 1. Mathematically, it is

$$\begin{aligned} &d_{min}(C_0, C_1) > d_{max}(C_1) \\ &|C_0| = 1 \\ &|C_1| >> 1 \end{aligned} \tag{16}$$

where >> means it is much greater (a few times greater).

Table 13 List of python functions and packages

Clustering algorithm	Python function	Python Package
Kmeans	KMeans	sklearn.cluster
Hierarchy clustering (ward)	linkage	scipy.cluster.hierarchy
Hierarchy clustering (single)	linkage	scipy.cluster.hierarchy
Hierarchy clustering (complete)	linkage	scipy.cluster.hierarchy
Hierarchy clustering (average)	linkage	scipy.cluster.hierarchy
Hierarchy clustering (weighted)	linkage	scipy.cluster.hierarchy
Hierarchy clustering (centroid)	linkage	scipy.cluster.hierarchy
Hierarchy clustering (median)	linkage	scipy.cluster.hierarchy
DBSCAN	DBSCAN	sklearn.cluster
Birch	Birch	sklearn.cluster
Spectral clustering	SpectralClustering	sklearn.cluster

Funding The authors did not receive support from any organization for the submitted work

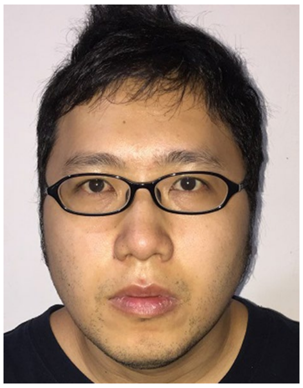
Data availability The datasets generated during and/or analysed during the current study are available from the corresponding author on reasonable request

Declarations

Conflict of interest The authors declare they have no financial interests

References

1. Radosavovic I, Dollár P, Girshick R, Gkioxari G, He K (2018) Data distillation: towards omni-supervised learning. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 4119–4128
2. Sarfraz S, Sharma V, Stiefelhagen R (2019) Efficient parameter-free clustering using first neighbor relations. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 8934–8943
3. Yang L, Zhan X, Chen D, Yan J, Loy CC, Lin D (2019) learning to cluster faces on an affinity graph. in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 2298–2306
4. Wang Z, Zheng L, Li Y, Wang S (2019) Linkage based face clustering via graph convolution network. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 1117–1125
5. He K, Gkioxari G, Dollár P, Girshick R (2017) Mask R-CNN. In: Proceedings of the IEEE international conference on computer vision, pp 2961–2969
6. Iscen A, Tolias G, Avrithis Y, Chum O (2019) Label propagation for deep semi-supervised learning. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 5070–5079
7. Wang S, Meng J, Yuan J, Tan Y-P (2019) Joint representative selection and feature learning: a semi-supervised approach. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 6005–6013
8. Wu S, Li J, Liu C, Yu Z, Wong H-S (2019) Mutual learning of complementary networks via residual correction for improving semi-supervised classification. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 6500–6509
9. Li Q, Wu X-M, Liu H, Zhang X, Guan Z(2019) Label efficient semi-supervised learning via graph filtering. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 9582–9591
10. Wu S, Deng G, Li J, Li R, Yu Z, Wong H-S (2019) Enhancing triplegan for semi-supervised conditional instance synthesis and classification. In: proceedings of the IEEE conference on computer vision and pattern recognition, pp 10091–10100
11. Yu B, Wu J, Ma J, Zhu Z (2019) Tangent-normal adversarial regularization for semi-supervised learning. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 10676–10684
12. Jiang B, Zhang Z, Lin D, Tang J, Luo B (2019) Semi-supervised learning with graph learning-convolutional networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 11313–11320
13. Qiao S, Shen W, Zhang Z, Wang B, Yuille A (2018) Deep co-training for semi-supervised image recognition. In: Proceedings of the European conference on computer vision (ECCV), pp 135–152
14. Robert T, Thome N, Cord M (2018) Hybridnet: classification and reconstruction cooperation for semi-supervised learning. In: Proceedings of the European conference on computer vision (ECCV), pp 153–169
15. Chen Y, Zhu X, Gong S (2018) Semi-supervised deep learning with memory. In: Proceedings of the European conference on computer vision (ECCV), pp 268–283
16. Shi W, Gong Y, Ding C, MaXiaoyu Tao Z, Zheng N (2018) Transductive semi-supervised deep learning using min-max features. In: Proceedings of the European conference on computer vision (ECCV), pp 299–315
17. Cicek S, Fawzi A, Soatto S (2018) Saas: speed as a supervisor for semi-supervised learning. In: Proceedings of the European conference on computer vision (ECCV), pp 149–163
18. Liu Y, Song G, Shao J, Jin X, Wang X (2018) Transductive centroid projection for semi-supervised large-scale recognition. In: Proceedings of the European conference on computer vision (ECCV), pp 70–86
19. Coelho de Castro D, Nowozin S (2018) From face recognition to models of identity: a bayesian approach to learning about unknown identities from unsupervised data. In: Proceedings of the European conference on computer vision (ECCV), pp 745–761
20. Kumar V, Namboodiri A, Jawahar C (2018) Semi-supervised annotation of faces in image collection. Signal Image Video Process 12(1):141–149
21. Sharma V, Tapaswi M, Sarfraz MS, Stiefelhagen R (2019) Self-supervised learning of face representations for video face clustering. arXiv preprint arXiv:1903.01000
22. Zhan X, Liu Z, Yan J, Lin D, Change Loy C (2018) Consensus-driven propagation in massive unlabeled data for face recognition. In: Proceedings of the European conference on computer vision (ECCV), pp 568–583
23. Shen S, Li W, Zhu Z, Huang G, Du D, Lu J, Zhou J(2021) Structure aware face clustering on a large-scale graph with 107 nodes. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, IEEE, pp 9085–9094
24. Nguyen XB, Bui DT, Duong CN, Bui TD, Luu K (2021) Clusformer: a transformer based clustering approach to unsupervised large-scale face and visual landmark recognition. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, IEEE, pp 10847–10856
25. Zhang K, Zhang Z, Li Z, Qiao Y (2016) Joint face detection and alignment using multitask cascaded convolutional networks. IEEE Sign Process Lett 23(10):1499–1503
26. Whitelam C, Taborsky E, Blanton A, Maze B, Adams J, Miller T, Kalka N, Jain AK, Duncan JA, Allen K, et al. (2017) Iarpa janus benchmark-b face dataset. In: Proceedings of the IEEE conference on computer vision and pattern recognition workshops, pp. 90–98
27. Guo Y, Zhang L, Hu Y, He X, Gao J(2016) Ms-celeb-1m: a dataset and benchmark for large-scale face recognition. In: European conference on computer vision, Springer, pp 87–102
28. Amigó E, Gonzalo J, Artiles J, Verdejo F (2009) A comparison of extrinsic clustering evaluation metrics based on formal constraints. Inf Retriev 12(4):461–486
29. Zhan X (2019) Implementation of “Consensus-Driven Propagation in Massive Unlabeled Data for Face Recognition” (CDP). GitHub
30. Yang L (2019) Learning to cluster faces on an affinity graph (CVPR 2019). GitHub
31. Wang Z (2019) Linkage-based face clustering via graph convolution network. GitHub
32. Yang L, Chen D, Zhan X, Zhao R, Loy CC, Lin D (2020) Learning to cluster faces via confidence and connectivity estimation. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp 13369–13378
33. Liu Y, Zhang G, Wang H, Zhao W, Zhang M, Qin H (2019) An efficient super-resolution network based on aggregated residual transformations. Electronics 8(3):339



Sing Kuang Tan

Xiu Wang