Agglomerative clustering via maximum incremental path integral: an implementation

Unsupervised and Reinforcement Learning

April 13, 2025

Bernat Comas i Machuca

Contents

1 Introduction to the problem

The paper Agglomerative clustering via maximum incremental path integral (2013) [1] presents a new agglomerative clustering algorithm that computes cluster similarity based on path graphs that capture the data structure. To make this computation each cluster is treated as a dynamic system and the algorithm measures stability using the Path Integral, a concept extracted from statistical mechanics and quantum mechanics. The algorithm decides how to merge clusters based on how much their stability changes, which results in an efficient solution with linear time complexity.

To define the structural descriptors that allow to compute this concept of similarity, a neighborhood graph is created. This allows it to avoid relying as much in pairwise distances (only used for graph initialization), and instead define the affinity between clusters by how much their stability changes when merged, which is measured through the path integral.

This has several advantages:

- Works well for data lying on low-dimensional manifolds.

- It does not rely on approximation or eigen-decomposition, making it more robust to noise.

- It does not make assumptions on data distribution, offering better generalization.

- Performs well on multi-scale data

- It is efficiently computed by the presented algorithm in linear time complexity

Over the course of this project, we are going to implement the Path Integral Clustering algorithm in an efficient way, and then we are going to compare it to other well-known clustering algorithms. In this section we are going to describe the algorithm, section 2 contains a description of the implementation , in section 3 the algorithm experiments and comparison is performed, and finally the last section is devoted to describing the conclusions of the project.

1.1 Cluster initialization

Because we are working with an Agglomerative Clustering algorithm, we have to define how will the initial clusters be formed. We could initialize the samples each in its own cluster and start the iteration, but instead the paper proposes to use nearest neighbor merging.

The algorithm of nearest neighbor merging consists of having each sample in its own cluster together with its nearest neighbor and then, the clusters that share samples are merged. This results in a number of initial clusters that is at least half the number of samples.

1.2 Building the structural graph

We define the structural graph G as the directed graph where each initial sample X is a node and E the set of edges connecting them. Two samples are connected by an edge if they are in its K closest neighbors (for a predefined graph).

Then, we compute the weighted adjacency matrix W, which contains the pairwise similarities between each pair of samples. Note that we are only computing similarities between each point and its K closest neighbors, the rest of similarities will be 0. Therefore, each element $w_i j$) of the matrix W is defined as shown in Equation 1.

$$w_{ij} = \begin{cases} \exp\left(\frac{-\text{dist}(i,j)^2}{\sigma^2}\right), & \text{if } \mathbf{x}_j \in \mathcal{N}_i^K, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Where K is a free parameter to be set and σ^2 is estimated by Equation 2.

$$\sigma^2 = [\sum_{j=1}^{n} \sum_{x_j \in \mathcal{N}_i^3} dist(i,j)^2]/[3n(-ln(a))] \tag{2}$$

Where a is a parameter to be set.

We define the transition probability matrix P as the one-step transition probability from vertex i to vertex j, and we compute it using Equation 3.

$$P = D^{-1}W;$$
$$d_{ii} = \sum_{j=1}^{n} w_{ij} \text{ such that } \sum_{j=1}^{n} p_{ij} = 1 \tag{3}$$

1.3 Affinity measure: Path integral

The path integral of a cluster is computed by summing the paths in the cluster on the directed graph G, weighted by transition probabilities in P.

The path integral used is a discretization of the one seen in quantum mechanics, and is defined as a sum of the weighted contributions of all the paths in the cluster C, divided by the number of samples belonging to the cluster. This is called the path integral descriptor of a cluster C_a, which we call S_{C_a}, and is defined in Equation 4. 4.

$$S_c = \frac{1}{|C|^2} \sum_{\gamma \in \Gamma_c} \Theta(\gamma) \tag{4}$$

Where, Γ_c is the set of all paths in C and $\Theta(\gamma)$ the weight of a path.

We also define the conditional path integral descriptor $S_{C_a|C_a \cup C_b}$ between two clusters S_{C_a} and S_{C_b} as the path descriptor of the clustering resulting from the merge but only taking into account the paths that have starting and ending vertices in C_a. We compute the conditional path integral descriptor using Equation 5.

$$S_{C_a|C_a \cup C_b} = \frac{1}{|C_a|^2} \mathbf{1}_{C_a}^T (\boldsymbol{I} - z\boldsymbol{P}_{C_a \cup C_b})^{-1} \mathbf{1}_{C_a} \tag{5}$$

Now, we can compute the affinity A_{C_a,C_b} of the merging of two clusters C_a and C_b by the increment in the path integral descriptor shown in the previous equations 4 and 5, as can be seen in Equation 6. In addition to the path integral of each cluster separately S_{C_a} and S_{C_b}, the computation of the affinity uses the conditional path integral descriptor $S_{C_a|C_a \cup C_b}$, which computes the resulting path descriptor of the merge but only the ones that have starting and ending vertices in C_a.

$$A_{C_a,C_b} = (S_{C_a|C_a \cup C_b} - S_{C_a}) + (S_{C_b|C_a \cup C_b} - S_{C_b}) \tag{6}$$

1.4 Agglomerative algorithm

The input to our problem is a set of sample vectors together with the target number of clusters (that has to be predefined).

The algorithm starts by building the graph, weighted adjacency matrix W, and using it to obtain the transition probability matrix P. Then, cluster initialization is run to obtain the initial clusters.

The iterative part consists of merging the two most affine clusters we have until the number of clusters is the target one. To do so, we use the precomputed affinities between all pairs of clusters. The pseudocode of the algorithm can be seen in Algorithm 1.

Algorithm 1 Path Integral Clustering Algorithm

1: **INPUT** The set of n points to cluster $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, and the target number of clusters n_T.
2: Build the graph G with k-nearest-neighbors and compute its weighted adjacency matrix $\mathbf{W}$.
3: Get the transition probability matrix $\mathbf{P}$
4: Run nearest neighbour merging to form n_c initial clusters $\mathcal{C}_c = \{\mathcal{C}_1, \ldots, \mathcal{C}_{n_c}\}$.
5: **while** $n_c > n_T$ **do**
6:     Search two clusters $\mathcal{C}_a$ and $\mathcal{C}_b$, such that

$$\{\mathcal{C}_a, \mathcal{C}_b\} = \arg \max_{\mathcal{C}_a, \mathcal{C}_b \in \mathcal{C}_c} \mathcal{A}_{\mathcal{C}_a, \mathcal{C}_b}$$

   where $\mathcal{A}_{\mathcal{C}_a, \mathcal{C}_b}$ is the affinity measure between $\mathcal{C}_a$ and $\mathcal{C}_b$.
7:     $\mathcal{C}_c \leftarrow \{\mathcal{C}_c \setminus \{\mathcal{C}_a, \mathcal{C}_b\}\} \cup \{\mathcal{C}_a \cup \mathcal{C}_b\}$
8:     $n_c \leftarrow n_c - 1$
9: **end while**
10: **OUTPUT** $\mathcal{C}_c$

---

2 Description of the implementation

In this section we are going to describe the particularities of the implementation of the Path Integral Clustering Algorithm. The first section goes over the implementation

techniques recommended by the paper, and the second one contains the extra decisions made during the implementation.

During the implementation, efficiency was our priority. The full implementation of the algorithm is contained in the files "pic.py", "nearest_neighbour_init.py" and "path_integral.py". The rest of files are helpers that have contributed to the results obtained but are not part of the implementation itself.

2.1 Efficient computation of the path integral

The main ideas laid out by the paper about the implementation are on the computation of the path integral between two clusters. They use Theorem 1 to avoid computing a matrix inverse but instead solve a linear system.

Theorem 1. s_{ij} always converges, and $s_{ij} = \left[(I - zP_{\mathcal{C}})^{-1}\right]_{ij}$, i.e., the (i, j)-element of $(I - zP_{\mathcal{C}})^{-1}$, where $P_{\mathcal{C}}$ is the submatrix of P by selecting the samples in $\mathcal{C}$. If we define $S_{\mathcal{C}} = [s_{ij}]_{i,j \in \mathcal{C}}$, we have $S_{\mathcal{C}} = (I - zP_{\mathcal{C}})^{-1}$. Then, we can compute the path integral as the structural descriptor of cluster $\mathcal{C}$ as follows:

$$S_{\mathcal{C}} = \frac{1}{|\mathcal{C}|^2}\mathbf{1}^T S_{\mathcal{C}}\mathbf{1} = \frac{1}{|\mathcal{C}|^2}\mathbf{1}^T(I - zP_{\mathcal{C}})^{-1}\mathbf{1} \tag{7}$$

where $\mathbf{1}$ is all-one column vector.

Proposition 1 $(I - zP_{\mathcal{C}})$ is a strictly diagonally dominant matrix with the ∞-norm condition number no more than $(1 + z)/(1 - z)$.

Both Theorem 1 and Proposition 1 are proven in the original paper [1], so we will not go into more detail here. Taking them into account, we see that the computation of $S_{\mathcal{C}}$ only involves solving a linear system

$$\begin{aligned} (I - zP_{\mathcal{C}})\mathbf{y} &= \mathbf{1}, \\ S_{\mathcal{C}} &= \frac{1}{|\mathcal{C}|^2}\mathbf{1}^T\mathbf{y}. \end{aligned} \tag{8}$$

For a large cluster, $(I - zP_{\mathcal{C}})$ is sparse, which means it follows Proposition 1 and can be solved using iterative methods. This will help us compute both the incremental path integral and the path integral descriptor.

$$S_{c_a|c_a \cup c_b} = \frac{1}{|c_a|^2}\mathbf{1}_{c_a}^T(I - zP_{c_a \cup c_b})^{-1}\mathbf{1}_{c_a} \tag{9}$$

To compute the path integral descriptor as shown in Equation 7, we first compute $(I - zP_{c_a \cup c_b})$, which is straightforward. Then, following equation 8, we solve the linear system (we can use spsolve). Finally, multiplying by the transposed matrix of 1s is the same as summing all the elements in the matrix resulting from the linear equation's solving, and finally we can compute the multiplication with the fraction. Computing the incremental path descriptor takes advantage of the same trick.

2.2 Implementation decisions

Because we want our algorithm to run in linear time, we need to precompute everything we can, and that includes not only the graph and its transition matrices, but also the affinity between each pair of clusters. Because as can be seen in 1 the decision on the clusters to merge is an argmax, we need to have computed the affinities between all pairs of clusters to compare them.

In addition to this, we are using a heap to store all the cluster pairs and their affinities for fast extraction in runtime. This brought us to use a set to store the clusters that are active in a moment, to avoid having to traverse the heap to remove all pairs of clusters that contains one of the merged ones at each iteration. This means that at every iteration we are flagging the merged clusters as inactive, computing the affinities between the new cluster and all the previous ones, and adding them to the heap.

Due to the nature of our algorithm, we only implement the fit-predict method. This is because the algorithm does not learn from data but instead applies the same process to the samples to cluster them independently from the data it has seen.

All the operations that involve the clustering algorithm use numpy, using matrices as numpy arrays, and taking advantage of the different tools we have to work with sparse matrices. Because of the way the similarities are computed, being set to zero if they are not in the K nearest neighbors of every sample, and because of their enormous sizes, we use different tools from scipy.sparse, including spsolve to solve the linear systems that compute the Path Integral (see previous section), converting matrices to csc form and making computations between matrices using the sparse tools.

3 Experiments

In this section we will conduct experiments comparing different clustering algorithms on several datasets. The algorithms under evaluation include Affinity Propagation, Complete Link, Average Link, Simple Link, Zeta clustering (Zell), Difussion clustering, and of course, our implementation of PIC. Each algorithm will be tested on four datasets: MNIST, USPS, Breast Cancer (Wisconsin), and a synthetic random dataset with 1000 samples and 50 features. We are going to divide the experiments section into two subsections. In the first subsection, we are going to analyse how does the introduction of gaussian noise and structural noise affect the performance of the different algorithms, similar to what is done in the original paper. In the second one, we are going to run the algorithms on the complete datasets to observe what are the differences in their performances, using a diverse set of metrics.

The choice of the algorithms we are using is due to the fact that we only have found implementations for the Affinity Propagation, Complete, Average and Simple Links. Although in the project we only had to implement PIC, we decided to also implement some of the algorithms that get the most similar results in the paper, and the selected algorithms to implement are the diffusion Kernel (D-kernel) and Zeta function clustering (Zell), because they are shown to be better than PIC in some domains in the original paper.

The datasets chosen are also extracted from the paper: MNIST and USPS. We opted not to use the FRGC ver2.0, PubFig, and Caltech datasets for this study due to their large size, which would require significant computational resources and time for processing, apart from the approach the paper takes on loading them, which uses spatial

pyramid features. Instead, we have decided to incorporate the Breast-cancer wisconsin dataset because it is a well-known benchmark in the medical domain, containing a high-dimensional, real-world classification and being widely used. In addition, being a binary classification dataset, it stands out from the rest and means that the algorithms will have to iterate until only two clusters are remaining, apart from being something not tested in the original paper.

We don't have access to the synthetic datasets that are used in the paper, and for this reason we have decided to create our. It will be using the synthetic dataset that we will study how does structural and gaussian noise affect our implementation of the clustering algorithm. Our synthetic dataset has 1000 samples, 10 features and 10 target clusters.

Table 1: Statistics of used datasets.

Dataset	USPS	MNIST	BC-Wisconsin	Synthetic
No. of samples	11 000	5 139	569	1000
No. of clusters	10	5	2	10
Min. cluster size	1100	980	212	-
Max. cluster size	1100	1135	357	-
Noise added	No	No	No	Yes
Dimensionality	256	784	30	10

3.1 Metrics to be used

The metrics we will be using to compare the methods are the same ones that are used in the paper: Normalized Mutual Information score and Clustering error, and as both of these are external metrics we decided to incorporate some internal ones: Silhouette score, Davies-Bouldin score, and Calinski-Harabasz Index. We have selected these three internal metrics because they are the ones recommended as internal criteria for a wide range of situations by the "Clustering evaluation and validation" chapter of the material of the Unsupervised Learning course [2]. Their implementations can be seen in Equations 10, 11, 12, 13, and 14.

$$\text{NMI}(U, V) = \frac{2 \times I(U; V)}{H(U) + H(V)} \qquad (10)$$

where:

- $I(U; V)$ is the mutual information between cluster assignment U and ground truth labels V.

- $H(U)$ and $H(V)$ are the entropies of U and V.

$$\text{CE} = 1 - \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}\{\hat{y}_i = y_i\} \qquad (11)$$

where:

- N is the total number of samples.

- $\hat{y}_i$ is the predicted cluster label after optimal matching.

- y_i is the ground truth label.

- $\mathbb{1}\{\cdot\}$ is the indicator function (1 if true, 0 otherwise).

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \tag{12}$$

where:

- $a(i)$ is the average distance from point i to all other points in the same cluster.

- $b(i)$ is the minimum average distance from point i to points in a different cluster.

The overall Silhouette Score is the mean of $s(i)$ over all points.

$$\text{DB} = \frac{1}{k} \sum_{i=1}^{k} \max_{j \neq i} \left(\frac{S_i + S_j}{M_{ij}} \right) \tag{13}$$

where:

- k is the number of clusters.

- S_i is the average distance between each point in cluster i and its centroid.

- M_{ij} is the distance between the centroids of clusters i and j.

$$\text{CH} = \frac{\text{Tr}(B_k)}{\text{Tr}(W_k)} \times \frac{N - k}{k - 1} \tag{14}$$

where:

- N is the total number of points.

- k is the number of clusters.

- $\text{Tr}(B_k)$ is the trace of the between-cluster dispersion matrix.

- $\text{Tr}(W_k)$ is the trace of the within-cluster dispersion matrix.

3.2 Noise analysis

In this section we will run experiments using the previously described synthetic dataset incorporating different levels of gaussian noise and structural noise to see how do the different methods behave. Because the addition of gaussian and structural noise is done randomly, we have followed the same practices from the original paper and repeated all the experiments 20 times, showing the mean and the standard deviation at each noise level.

3.3 Algorithm comparison

...

4 Conclusions

Comment about the claims of the paper, reproducibility & issues encountered.

5 Bibliography

[1] Wei Zhang et al., *Agglomerative clustering via maximum incremental path integral*, 2013 (Link)

[2] Javier Béjar, URL - 2025 Spring Term *Clustering evaluation and validation*