Incremental Ng-Jordan-Weiss Algorithm Group_404

Keerti Harpavat
170101031
keert170101031@iitg.ac.in

Priyanshu Singh
170101049
singh170101049@iitg.ac.in

Aman Mishra
170101005
aman170101005@iitg.ac.in

Aniket Rajput
170101007
anike170101007@iitg.ac.in

Abstract

Incremental clustering algorithms aim to use approximations to avoid re-computing the clustering for a dataset, due to small change in the dataset. Ng-Jordan-Weiss(**NJW**) algorithm is a simple spectral clustering algorithm, which has shown to work well across different domains of clustering problems. In this paper, we propose an incremental version of this algorithm. Owing to matrices and their eigensystems being the central focus in the static version, the key concept we use in the incremental version is **Eigen Perturbation Theory**. We evaluate our proposed algorithm on several datasets and report the results. Our code is publicly available here.

CCS Concepts: • **Spectral Clustering → Matrices**; • **Eigensystem → Eigen Perturbation Theory**.

Keywords: Laplacian Matrix, Eigenvalues, Eigenvectors, Eigen Perturbation

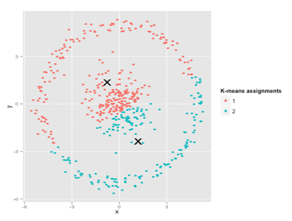
ACM Reference Format:
Keerti Harpavat, Priyanshu Singh, Aman Mishra, and Aniket Rajput. 2021. Incremental Ng-Jordan-Weiss Algorithm Group_404. In *Data Mining '21: Incremental Algorithms, Jan-May, 2021, Guwahati, India.* 5 pages.

1 Introduction

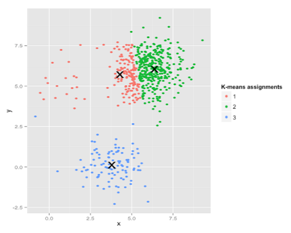
Clustering is a very fundamental problem in computer science, which attempts to group together similar data points into clusters, segregating points of different nature. Traditional clustering methods range from generative models like EM to K-means and spectral graph partitioning approaches. We briefly discuss these below.

For clustering points in $\mathbb{R}^n$, one standard approach is based on generative models, in which algorithms such as EM are used to learn a mixture density. The major drawback of such algorithms is the necessity of simplifying assumptions, like Gaussian distribution, to use parametric density estimators. Another popular algorithm is the K-means algorithm which aims to partition the n data points into k clusters, using an iterative method to identify clusters and their centroids. K-means fails to accurately find the clusters in cases when the data is non-spherical and when the clusters are of uneven sizes (Fig. 5). In case of multiple local minima, it may also require multiple restarts. A promising alternative that has recently emerged in a number of fields is to use **spectral methods** for clustering. Here, one uses the top eigenvectors of a matrix derived from the pairwise distances between points. Different spectral clustering algorithms differ in the way in which they use the eigenvectors ([5]). These algorithms build a weighted graph in which nodes correspond to data points and edges are related to the distance between the points. One such clustering algorithm looks at the second eigenvector of the graph's *Laplacian matrix* to define a semi-optimal cut which partitions the graph into two clusters. This method of making cuts can be applied recursively to generate the k clusters from the data points [4]. Experimentally it has been observed that using more eigenvectors and directly computing a k-way partitioning is better that using the above recursive method. Meila and Shi algorithm [1] uses k eigenvectors simultaneously to find the k clusters. However, this method might be susceptible to bad clusterings when the degree to which different clusters are connected varies substantially across clusters. NJW algoritm builds upon the Meila and Shi algoritm with some cosmetic changes in which normalization and eigenvectors are used differently. It's a method that is robust to a very large variety of clustering problems and generally performs better compared to its counterparts.

These different approaches are suitable for different domains of data. But there is a different paradigm to clustering, irrespective of the type of data. This is the alteration of data -



(a) Non-spherical Data



(b) Uneven sized clusters

Figure 1. Fail cases of k-means clustering

insertion or deletion or updation of few data points. In many cases, the fraction of points altered is very small compared to the whole dataset. In such cases, re-computing the clustering of the whole dataset again is a cumbersome and sometimes even impractical process. To circumvent the re-computation, with some reasonable margin of error, incremental methods are used. These methods leverage approximations to speed up the computation. Here we propose an incremental version for the NJW algorithm using the approximations detailed in the **Eigen Perturbation Theory**.

2 Base Static Algorithm

As we saw earlier, spectral clustering algorithms first convert the data points clustering problem to a graph cut problem. On the same lines, the NJW algorithm defines a matrix which pertains to the distances between the points, known as the Affinity matrix, $A \in \mathbb{R}^{n \times n}$ defined as $A_{ij} = e^{\frac{-\|s_i - s_j\|^2}{2\sigma^2}}$ for $i \neq j$ and $A_{ii} = 0$. Then it defines the graph Laplacian matrix L using A and the diagonal degree matrix D. Then the eigenvectors corresponding to the k largest eigenvalues are stacked up to form the matrix X, normalizing whose rows gives Y. Now each row of Y is treated as a point in $\mathbb{R}^k$, and clustered using any standard clustering algorithm like K-means. Finally, the original points are assigned a cluster iff the corresponding row of Y is assigned that cluster. The algorithm is described in Fig. 2.

2.1 Implementation

We found an implementation of NJW algorithm in MATLAB here. We ran this and were able to generate the clusters for a random data distribution.

We implemented the NJW algorithm in C++ from scratch, taking reference from the above mentioned MATLAB code whenever needed. For finding eigenvalues and eigenvectors of Laplacian matrix, we used the **eigen3** library available for C++. K largest eigenvectors were obtained by sorting eigenvectors w.r.t eigenvalues using *pair* STL. Finally matrix X was constructed by stacking K largest eigenvectors as columns, which were then L2-normalized to form matrix Y.

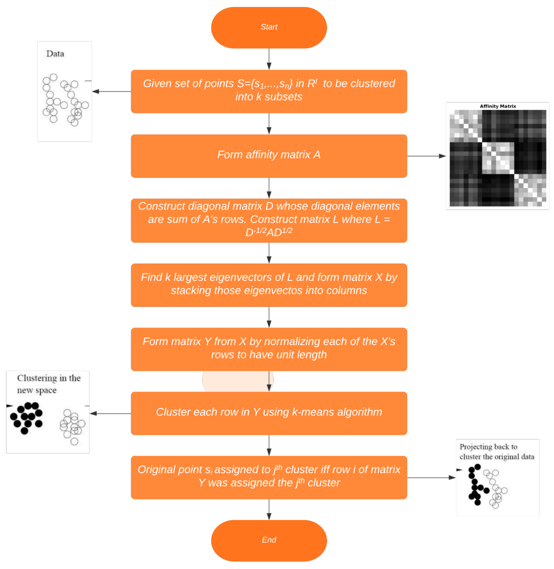

Start
Given set of points S={s1,...,sn} in Rl to be clustered into k subsets
Data
Form affinity matrix A
Affinity Matrix
Construct diagonal matrix D whose diagonal elements are sum of A's rows. Construct matrix L where L = D-1/2AD1/2
Find k largest eigenvectors of L and form matrix X by stacking those eigenvectos into columns
Form matrix Y from X by normalizing each of the X's rows to have unit length
Clustering in the new space
Cluster each row in Y using k-means algorithm
Original point si assigned to jth cluster iff row i of matrix Y was assigned the jth cluster
Projecting back to cluster the original data
End

Figure 2. NJW Algorithm

We applied k-means on matrix Y using open-sourced code found here.

3 Related incremental algorithms

To the best of our knowledge, there is no existing work on incremental NJW algorithm. There are works related to incremental spectral clustering algorithms [2][3], but they are based on the general formulation of spectral clustering, where the Laplacian L is defined as $D - A$, where D is the degree matrix, and A is the affinity matrix as defined above. These works provide insights on some possible directions towards incremental spectral clustering, but since they use some results derived from the definition of L as $D - A$, it is not suitable to apply them for incremental NJW algorithm, where L has a quite different formulation as $D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$.

4 Our Approach

4.1 Intuition

For converting the existing NJW algorithm into an incremental algorithm, we follow the approach discussed in [2][3] and Eigenvalue Perturbation Theory. [2] and [3] work on incremental spectral clustering, providing some intuition for following this approach in our case also. For applying the ideas to our work, Eigenvalue Perturbation Theory provides the required mathematical background.

Algorithm 1: Proposed Incremental NJW Algorithm(Insertion)

```
**Input:** Point $p_{n+1}$ inserted in the dataset
1 Construct matrix $L_{pad}$ of dimension $(n+1)\times(n+1)$
    by padding a row and column of $0s$ to $L$.
2 Compute matrix $A'$ of dimension $(n+1)\times(n+1)$
    using $A$ and calculating the distances of $p_{n+1}$ with
    each of $p_1, p_2, ..., p_n$. Compute $D'$ and $L'$ using $A'$.
3 $n$ eigenvalues of $L_{pad}$ are $\{\lambda_{01}, \lambda_{02}, ..., \lambda_{0n}\}$, with
    corresponding eigenvectors $\mathbf{x_{pad_i}} = [\mathbf{x_{0i}}^T|0]^T$ (i.e. an
    element 0 padded at (n+1) th dimension in
    $\mathbf{x_{0i}}$)$\forall i \in \{1, 2, ..., n\}$.
4 Since $L_{pad}$ has a row of 0s, it is singular, so it has an
    eigenvalue 0, which is the (n+1) th eigenvalue. Its
    corresponding eigenvector($\mathbf{x_{pad_{n+1}}}$) must be
    orthogonal to each of $\mathbf{x_{pad_i}}$ since $L_{pad}$ is a real
    symmetric matrix. This is found by solving
    $[\mathbf{x_{pad_1}}|\mathbf{x_{pad_2}}|...|\mathbf{x_{pad_n}}]^T\mathbf{x_{pad_{n+1}}} = 0$.
5 $L_{pad}$ and $L'$ are both of dimension $(n+1)\times(n+1)$.
    We know the $n+1$ eigenvalues and eigenvectors of
    $L_{pad}$. Using Eigenvalue Perturbation Theory, we
    approximate the eigenvalues $\{\lambda_1, \lambda_2, ..., \lambda_n, \lambda_{n+1}\}$ and
    eigenvectors $\{\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_n}, \mathbf{x_{n+1}}\}$ from the
    eigenvalues $\{\lambda_{01}, \lambda_{02}, ..., \lambda_{0n}, 0\}$ and eigenvectors
    $\{\mathbf{x_{pad_1}}, \mathbf{x_{pad_2}}, ..., \mathbf{x_{pad_n}}, \mathbf{x_{pad_{n+1}}}\}$ of $L_{pad}$ using
    equations 1 and 2
    $\forall i \in \{1, 2, ..., n+1\}(\Delta L = L' - L_{pad}, \lambda_{0,n+1} = 0)$.
6 Once we get these eigenvectors $\{\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_n}, \mathbf{x_{n+1}}\}$,
    we normalize and sort them to get the top $k$
    eigenvectors and then the algorithm proceeds as the
    static version, with k-means.
```

4.2 Description of Incremental algorithm

$$\lambda_i = \lambda_{0i} + \mathbf{x_{pad_i}}^T \Delta L \mathbf{x_{pad_i}} \tag{1}$$

$$\mathbf{x_i} = \mathbf{x_{pad_i}} + \sum_{\substack{j=1 \\ j\neq i}}^{n+1} \frac{\mathbf{x_{pad_j}}^T \Delta L \mathbf{x_{pad_i}}}{\lambda_{0i} - \lambda_{0j}} \mathbf{x_{pad_j}} \tag{2}$$

Given n points $\{p_1, p_2, ..., p_n\}$, matrices A, D, L (each of dimension $n\times n$) calculated as in the static algorithm. Eigenvalues $\{\lambda_{01}, \lambda_{02}, ..., \lambda_{0n}\}$ and eigenvectors $\{\mathbf{x_{01}}, \mathbf{x_{02}}, ..., \mathbf{x_{0n}}\}$ of L calculated as in static algorithm. k-means applied to these and clusters identified. All eigenvectors mentioned are unit normalized. The algorithm is described in Algo 1.

This proposed algorithm is for handling **insertion** of new data point. For just **change** in distances between points, we can directly apply the eigenvalue perturbation theory. For **deletion**, say removing the rth data point, we construct L'_{pad} with $0s$ in the rth row and column. Its eigenvalues and eigenvectors are calculated from those of L using eigenvalue perturbation theory. Then from each eigenvector, we remove the rth element as the dimension is reduced from n to $n-1$.

For k-means step also, we can use a standard incremental version, instead of re-calculation. But since k-means is not the bottleneck here, we are using the static k-means in our algorithm.

Fig. 3 summarizes the proposed incremental algorithm.

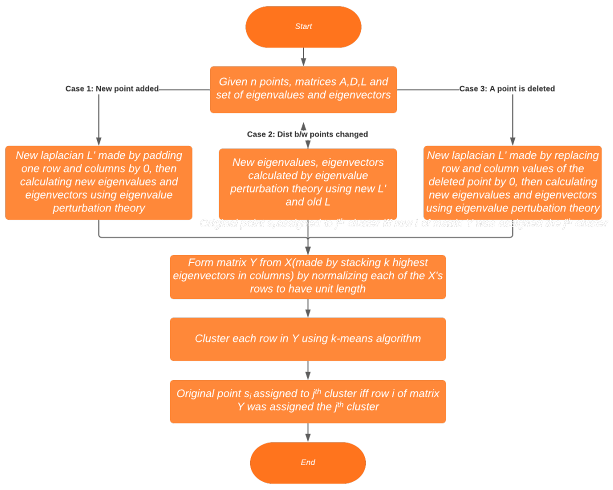

Start
Given n points, matrices A,D,L and set of eigenvalues and eigenvectors
Case 1: New point added
Case 2: Dist b/w points changed
Case 3: A point is deleted
New laplacian L' made by padding one row and columns by 0, then calculating new eigenvalues and eigenvectors using eigenvalue perturbation theory
New eigenvalues, eigenvectors calculated by eigenvalue perturbation theory using new L' and old L
New laplacian L' made by replacing row and column values of the deleted point by 0, then calculating new eigenvalues and eigenvectors using eigenvalue pertubation theory
Form matrix Y from X(made by stacking k highest eigenvectors in columns) by normalizing each of the X's rows to have unit length
Cluster each row in Y using k-means algorithm
Original point s_i assigned to j^{th} cluster iff row i of matrix Y was assigned the j^{th} cluster
End

Figure 3. Proposed Incremental NJW Algorithm

4.3 Code Architecture

We have two classes, a basic class **NJW** - defining the static version, and a class **Incremental** defining the incremental version. The attributes and methods of these classes are described in the class diagram(Fig. 4). The Incremental class contains a pointer to object of class NJW. It has methods for insertion, deletion and updation of data points. It makes the updates in the attributes of the NJW class' object using setter and getter methods.

5 Datasets & Experimental Results

We test our algorithm on several datasets(Fig. 5) available here. We performed insertion, deletion and updation operations on each of the datasets, the results are summarized in tables 2, 3 and 4. The summary of our work is provided in table 1.

6 Alternate Explorations and Lessons learnt

Handling the various issues we faced across different explorations was a big learning in itself. During the course of the project, we tried different approaches, many of which performed poorer than expected. Some of them are described below.

Base Paper	On Spectral Clustering: Analysis and an algorithm
Code of the Base Paper	Spectral Clustering Algorithms
Datasets	Clustering Datasets
Our Code	Github
Is the code working ?	Yes
Maximum speed up	≈ 1.5 times of Static Algorithm
Extra Amount of Memory	≈ 1.4 times of Static Algorithm
Accuracy compared to Static version	≈ 60 %
Measure of Accuracy	Accuracy is measured by calculating fraction of points assigned to right cluster using this method
Interested in further work	Interested, but depends on company joining dates

Table 1. Summary of our work(Statistics averaged across the datasets)

Dataset	Points	Total Time(Incremental)(sec)	Total Time(Static on New Set)(sec)	% Accuracy (Incremental)
jain	373	44.102	108.325	100
flame	240	6.77017	18.5257	99.17
spiral	312	19.9922	52.3158	100
pathbased	300	17.6889	45.3113	100
R15	600	314.52	824.473	65.34

Table 2. Running time and accuracy comparison for various datasets(Updation)

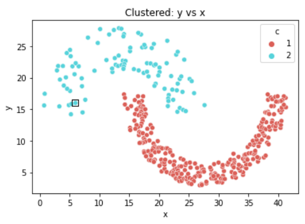
Dataset	Points	Total Time(Incremental)(sec)	Total Time(Static on New Set)(sec)	% Accuracy (Incremental)
jain	373	44.9543	109.624	65.77
flame	240	7.05119	18.692	100
spiral	312	20.7202	53.3201	53.99
pathbased	300	18.1427	45.9236	100
R15	600	325.581	833.852	51.91

Table 3. Running time and accuracy comparison for various datasets(Insertion)

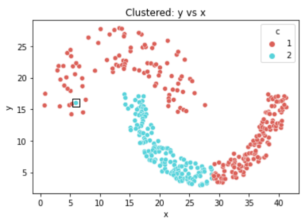
Dataset	Points	Total Time(Incremental)(sec)	Total Time(Static on New Set)(sec)	% Accuracy (Incremental)
jain	373	44.8069	106.614	65.86
flame	240	6.77902	18.1361	99.58
spiral	312	19.8795	51.9497	67.52
pathbased	300	17.7658	44.8977	100
R15	600	337.242	839.619	62.44

Table 4. Running time and accuracy comparison for various datasets(Deletion)

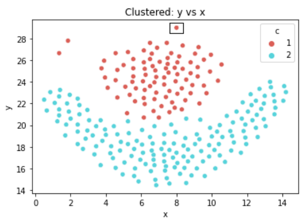
- The static algorithm we coded in C++ practically does not work for datasets with number of points greater than around 1000, whereas the same code in MATLAB works much faster. This highlights the presence of inbuilt optimizations in some programming languages which are suitable for a particular domain. We were not able to optimize our code to such an extent despite using optimization flags like -O3 which improved the speedup to some extent.
- The results indicate the mediocre accuracy we achieve, and the speedup is also not as significant as expected. For the speedup part, we tried using CUDA toolkit to use GPU parallelization. But owing to the small number of points that our static algorithm works on, the asymptotic speedup does not materialize for such small datasets. Similarly on using multi-threading, the running time actually increased as the memory operations required much more time using multiple threads as compared to single thread implementation.
- To keep the matrix dimensions same, we tried a way in which we initially kept some extra n' points at infinity - for implementation they were kept heuristically at a large distance. Then insertion was just updating these points. This method yielded almost identical results as our original implementation.



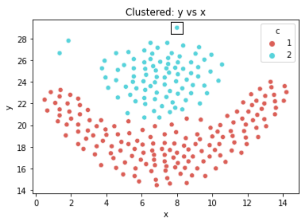
(a) Jain static, k=2



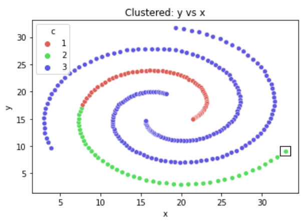
(b) Jain incremental, k=2



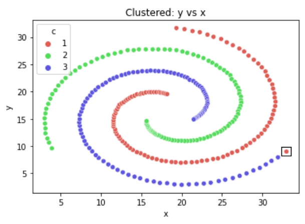
(c) Flame static, k=2



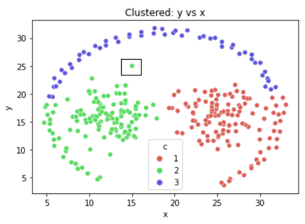
(d) Flame incremental, k=2



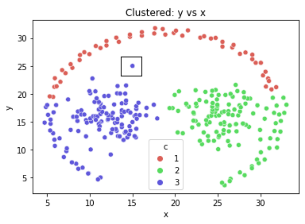
(e) Spiral static, k=3



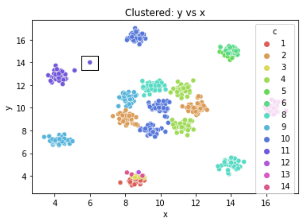
(f) Spiral incremental, k=3



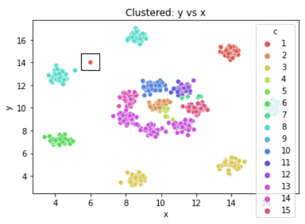
(g) Pathbased static, k=3



(h) Pathbased incremental, k=3



(i) R15 static, k=15



(j) R15 incremental, k=15

Figure 5. Static and Incremental results on datasets used(Insertion), inserted point is marked by a box

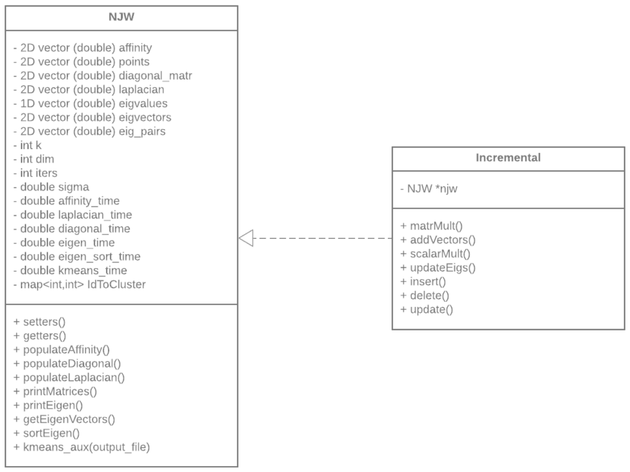


Figure 4. Class Diagram

- The pivot of our incremental algorithm is the eigen perturbation theory. As the results demonstrate, this theory is only applicable for small changes to the dataset. Large deviation leads to fall in accuracy. So the approximations of eigen perturbation theory become a bottleneck for our algorithm's accuracy.

7 Future Work

As we saw the different explorations and their drawbacks, further work shall be aimed at overcoming those hurdles one by one. A better approximation than eigen perturbation theory can help with the accuracy. Also if we are able to get the running time of the static algorithm in a feasible range for the static algorithm, the asymptotic speed up with CUDA and/or multi-threading can be fruitful. Further optimization of the code, if possible using some inbuilt libraries can be a good step in this direction.

References

[1] Marina Meila and Jianbo Shi. 2001. A Random Walks View of Spectral Segmentation.

[2] Huazhong Ning, Wei Xu, Yun Chi, Yihong Gong, and Thomas Huang. 2007. Incremental spectral clustering with application to monitoring of evolving blog communities. In *Proceedings of the 2007 SIAM International Conference on Data Mining*. SIAM, 261–272.

[3] Huazhong Ning, Wei Xu, Yun Chi, Yihong Gong, and Thomas S Huang. 2010. Incremental spectral clustering by efficiently updating the eigen-system. *Pattern Recognition* 43, 1 (2010), 113–127.

[4] Daniel A. Spielman and Shang-Hua Teng. 2007. Spectral partitioning works: Planar graphs and finite element meshes. *Linear Algebra Appl.* 421, 2 (2007), 284 – 305. https://doi.org/10.1016/j.laa.2006.07.020 Special Issue in honor of Miroslav Fiedler.

[5] Yair Weiss. 1999. Segmentation using eigenvectors: a unifying view. In *Proceedings of the seventh IEEE international conference on computer vision*, Vol. 2. IEEE, 975–982.