Spectral Clustering Implementation and Appliacation

Qi Wang Hanqiu Xia

April 28, 2016

Abstract

Spectral clustering is widely used in image segmentation. The main idea of spectral clustering is to use the spectrum (eigenvalues) of the similarity matrix of the data to perform dimensionality reduction and then perform clustering in fewer dimensions. In this project, we investigated the normalized spectral decomposition algorithm from the paper "On Spectral Clustering: Analysis and an algorithm" by A. Ng, M. Jordan, and Y. Weiss [1]. Their algorithm improved upon the existing spectral clustering algorithm by resolving the issues of inconsistent algorithms of using eigenvectors as well as providing evidence of it resulting in a reasonable clustering. We implemented the algorithm using Python with considering both common and edge cases. We also optimized the Python codes by applying vectorization, Cython, and Just-In-Time compiling. Finally, we tested the algorithm on simulated and real datasets and compare the experimental results with those of k-means clustering to see if spectral clustering dramatically improves the results.

1 Background

The research paper we have chosen is "On Spectral Clustering: Analysis and an algorithm" by A. Ng, M. Jordan, and Y. Weiss [1]. Spectral clustering is inspired by the the idea of spectral graph partitioning, in which we use the first two eigenvectors to partition the graph into exactly two parts. It is widely used in image segmentation. According to Ng et al [1]., the basic idea of spectral clustering is utilising the k eigenvectors simultaneously to cluster points into k subsets. The detailed algorithm is as follows:

Suppose we have a set of n points $S = \{s_1, \dots, s_n\}$ in $\mathbb{R}^m$, and we want to cluster them into k groups.

Step 1: Construct the affinity matrix $A \in \mathbb{R}^{n\times n}$, each element in A is defined as $A_{ij} = exp(-||s_i - s_j||^2/2\sigma^2)$, for $i, j = 1, \dots, n$. We will introduce a method of choosing σ in later pages.

Step 2: Define D to be the diagonal matrix with $D_{ii} = \sum_{j=1}^{n} A_{ij}$, and form the normalized Laplacian matrix $L = D^{-1/2}AD^{-1/2}$.

Step 3: Capture the fist k largest eigenvectors of $L, e_1, e_2, \dots, e_k, e_i \in \mathbb{R}^n$ and form the matrix $E = [e_1\ e_2\ \cdots\ e_k] \in \mathbb{R}^{n\times k}$.

Step 4: Create the new normalized matrix $U \in \mathbb{R}^{n\times k}$ from E, defined as $U_{ij} = E_{ij}/(\sum_{j=1}^{n} E_{ij}^2)^{1/2}$.

Step 5: Consider U to be a set of n points that need to be clustered now, apply K-means or any other algorithm that can minimize distortion to cluster U.

Step 6: Assign the original point s_i in to cluster j if and only if the ith-row of U was distributed to cluster j in previous step.

It improved upon the spectral clustering algorithm by suggesting the normalized spectral decomposition in forming the affinity matrix. Their algorithm resolved the problem of inconsistent algorithms of using eigenvectors in spectral clustering. When applying the algithm to data that does not have a clear segmentation, it performs much better than the other clustering algorithms. We will implement the spectral clustering algorithm and optimize the coding, and then we will utilize it on real datasets and compre the results with other clustering mechanisms such as k-means clustering.

2 Implementation

We have written three functions to implement the algorithm. The first function GenerateData generates data for the ideal case. The second function CalculateAffinity calculates the affinity matrix which is the first step in the above described algorithm. The third function Spectral implements Step 2 to 4 in the algorithm and creates a normalized matrix.

2.1 Ideal Case

The ideal case denotes the situation when the clusters are clearly partitioned and are far apart from each other. The proposed mechanism works very well in this case and will result in an exact match to the true clustering of the original data.

2.2 General Case

3 Testing

We used pytest-ipynb to conduct unit testing. The testing file is named test*.ipynb, and by running through the file in terminal by the command py.test, the unit testing is run. All tests are passed.

3.1 Unit Test for Common Case

For each function, there are three tests to assert that each function creates data in the correct format and correct dimension. And by setting the seed, we can confirm that each function creates the correct matrix.

3.2 Unit Test for Edge Case

For each function, there are two tests for edge cases. For the edge cases, all functions should generate an error for the incorrect inputs.

4 Optimization

We used vectorization, cython, and Just-In-Time compiling to optimize our code.

4.1 Vectorization

To eliminate the use of for loops, we vectorized to code to improve the performance. If we have a large dataset to perform clustering, the runtime will improve consistently by at least a factor of two.

4.2 JIT (Just-In-Time compiling)

We also used JIT to improve the performance, and as a result, JIT improves the runtime the most. It improved by around 20 fold.

4.3 Cython

Cython does not perform quite as well compared to the above two methods. It only improves the runtime by around a factor of 1.5.

4.4 Improvement result

We can see that JIT produces the best results in optimization. Thus, we will use JIT in the following applications.

5 Application and Comparison

5.1 Application on Simulated Data

The first set of simulated data we used fall under the ideal case. The data points are well segmented and from the graph we can see they form three clusters perfectly.

5.2 Application on Real Data

We will now apply the algorithm to the iris dataset.

5.3 Comparison of Sepctral clustering and K-means

6 Conclusion

References

[1] A. Ng, M. Jordan, and Y. Weiss. On Spectral Clustering: Analysis and an algorithm. *Advances in Neural Information Processing*. Vol. 14, No. 2. (2001), pp. 849-856