An Application of The Spectral Clustering Technique to Feature Selection in Maps

Ian Char
iach5500@colorado.edu

Denis Kazakov
deka6994@colorado.edu

Rhys Braginton Pettee Olsen
rhol9958@colorado.edu

December 11th, 2014

Abstract

For this project, we investigated and applied the technique of spectral clustering in the context of identifying complicated, non-concave but connected features of map images that include rivers and lakes.

We concluded that the techniques we applied could be effective at mechanising the selection of features of an image for which humans hugely outperform less sophisticated algorithms such as exploring the neighboring pixels with similar values or applying the k-means algorithm (described below) directly, noting however that our

choice of similarity function causes both performance and accuracy to suffer. We also contemplated ways in which the technique could be extended: in the tracking of objects as they change over time, for instance.

Introduction

Clustering refers to techniques that partition data points into a certain set number of *clusters* without making prior assumptions about which points should be clustered where. The technique of spectral clustering is highly general and applicable to a diverse array of problems and is particularly advantageous in those involving non-convex clustering that cannot be done by simply "lumping" together points in the original space [3, p. 28]. It is outlined in thorough detail by Von Luxburg [4] and described and quickly derived by Singh [3], both of whom outline derivations of it from a relaxation of a computationally difficult graph-cutting problem, which we describe in the Mathematical Formulation section of this paper.

Roughly familiar with the generality and usefulness of spectral clustering techniques, we outlined several ideas that involved clustering data. Ideas we considered included identifying separate speaking voices by clustering around overtones in speech and trying to identify separate users of Twitter by statistical patterns in their tweets. We ultimately chose to apply these techniques to identifying visual features of maps for several reasons.

First, conditioning either audio or textual data such that the patterns we are interested in can be efficiently represented is a challenge, especially when the size of

the graph that would likely result is considered. Second, defining notions of similarity between different sound intervals or different segments of text in a corpus is far from trivial and would likely involve some other sophisticated technique or reduction; spectral clustering would then be but a small step in a long pipeline of complicated reductions, drawing on tools that might need to be pulled from disparate corners of engineering or the bleeding edge of research—in fact, it's not even clear if these problems are tractable with clustering techniques. Third, even assuming spectral clustering could, at some point, be applied, interpreting the clusters found and identifying if we had succeeded in mapping the hidden relationships we seek would further extend the duration of the project. These concerns placed these interesting hypotheticals squarely out of the scope of this project.

On the other hand, applying spectral clustering to images of maps is attractive. Representing an image as a graph is straightforward, and several obviously good notions of similarity are easy to define—an indicator for color similarity of neighboring pixels or, in the case of our project, a Gaussian kernel based on differences in brightness, are good examples; thus the problem does not have to be significantly recast for spectral clustering to be applicable. Second, the inherently visual nature of pictures makes it easy to generate diagrammatic representations of our clusters and quickly say things about our hypothesis and the validity of what we're doing, bringing down the time it would take to execute this project further. Lastly, a great deal of interest surrounds map data, and the ability to automate tasks typically relegated to humans—such as picking out bodies of water, partitioning areas based on dividing structures like rivers, measuring the way features like ice sheets tend to change over

time—could help scale work volumes involving surveying and interpreting map data to levels not possible with human labor. With these advantages in mind, we settled on applying spectral clustering to map data.

Mathematical Formulation

Suppose you have a network or graph consisting of a collection of individual items or nodes with an arbitrary number of edges between them with a weight that represents how "similar" or close the two connected nodes are. This abstraction can be built over a variety of concrete and formal situations. for instance, the nodes could represent cities and the edges their geographical distance from one another, or—to extend this formulation to more conventional linear algebra problems—the nodes could represent points in some multidimensional space and the edges between them the inverse distances in some norm. Now suppose you wish to find a certain number of groups of points that are closely related (with high edge weights between them) by identifying "cuts" or divisions between the points such that the sum of the edge weights across the cuts are minimized. That is, you seek to separate the graph into a fixed number of groups such the groups are as dissimilar from one another as possible. The problem with this approach is that it tends to produce several very small groups. If the constraint of having the groups be of similar size is added, the problem becomes NP-hard [1]: in informal computational terms, intractable to solve for all but the smallest of cases. The technique of spectral clustering solves a slight relaxation of this problem; it produces results that are quite close to a true, optimal solution far more cheaply

than finding the true optimum.

To do spectral clustering, one starts by representing the n-node graph with a so-called adjacency matrix: an n-by-n matrix whose (i, j)th value is the edge weight from node i to node j and 0 if no such edge exists. If we assume the graph is non-negative and bidirectional (that is, all edges are non-negative and the edge from i to j is the same as the edge from j to i), then it immediately follows that its adjacency matrix will be symmetric with all non-negative values. Next, one creates what is known as a Laplacian matrix L of size $n \times n$ whose (i, i)th diagonal element is the number of edges from the ith vertex to its neighbors and whose other elements are simply the negatives of the corresponding elements of the adjacency matrix. The sum of each column and row of the L will be 0; for this reason, L has an n-lengthed eigenvector $(1, 1, .., 1)^{\mathrm{T}}$ whose corresponding eigenvalue is 0. In fact, L has exactly n eigenvectors, all of which are real because of L's symmetry.

To find the clusters, one builds a new matrix M consisting of the k smallest eigenvectors of L as columns for some chosen constant k representing the number of clusters sought, typically smaller than n. Now the rows of M represent points in a k-dimensional space. One runs the k-means clustering algorithm, which is described as follows: to make k clusters, randomly pick k centroids in the space being considered. Then group each item according to which centroid the item is closest to; move this centroid to be the new average of all points in the cluster and repeat this process until the amount by which each point changes is smaller than some chosen small value ϵ [2].

At a high level, this technique works by moving a set of data in n dimensions into a

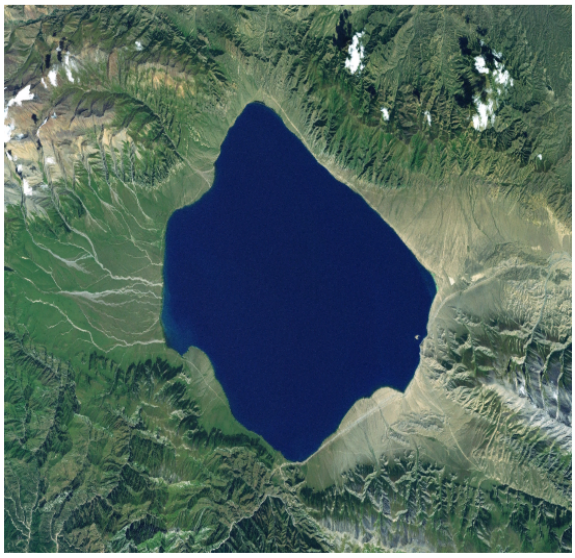
lower-, k-dimensional space described by a basis of eigenvectors that closely approximate the graph-cutting separation for the chosen similarity function defining the edge weights. Since this space represents the separability the graph into parts by construction, running k-means on it quickly finds the clusters that correspond to our notion of separability.

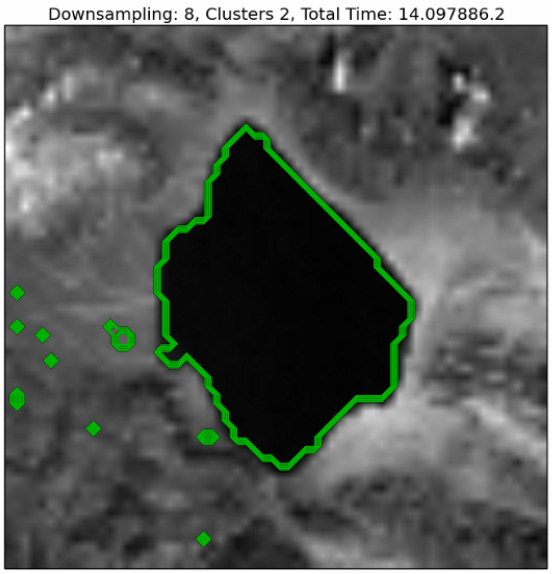
With this description established, we now outline how we applied spectral clustering to the problem of clustering map features. First, we downsampled to image to improve running time and represented each pixel with a node. Because features like bodies of water need not be continuous, we hypothesized that the overriding concern was the brightness-similarity of each pixel as opposed to spatial distance and defined our similarity function accordingly, with a standard Gaussian kernel function $\exp -(x[i] - x[j])^2/\sigma^2$ for (i, j)th and (j, i)th entries of the adjacency matrix, where $x_[i]$ is the brightness of the ith pixel. This causes similarity to fall off with the square of the exponential as two pixels being compared become less and less similar in brightness. The graph Laplacian is accordingly constructed and the algorithm is run. Once the clusters have been found, the divisions between them are drawn onto the map data.

Examples and Numerical Results

We ran the algorithm on several images under different settings to see how well the system performed and if it produced results consistent with our goals of identifying complicated and sometimes discontinuous bodies of water.

Example 1: Large Lake

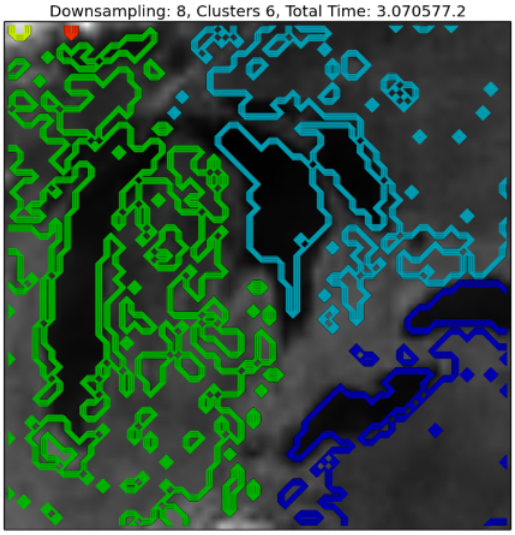




This first example is a satellite image of a mountainous region with a series of narrow, downhill tributaries feeding a large, central lake. We were hoping that the clustering method would identify both the lake and the narrow tributaries feeding it—ideally as part of the same body, hence the number of clustering being set to 2. The algorithm succeeded in isolating the lake from the surrounding landscape. Unfortunately, the downsampling that was necessary to make the code perform reasonably caused the detail needed to identify the tributary rivers to be lost. Additionally, since the similarity function is invariant with spacial distance, several features that are not water were grouped with the lake. To make the clustering technique suitable for this example, changes might include a similarity function that considers spacial locality (for instance, two pixels would only be similar if they were either one of the eight neighboring pixels or within some finite distance from one another) and a far less aggressive downsampling regimen, which might make a less costly similarity function

necessary. Note that if only pixels in a certain neighborhood were considered, calculating similarity values would become much cheaper, so these two changes could easily be complementary.

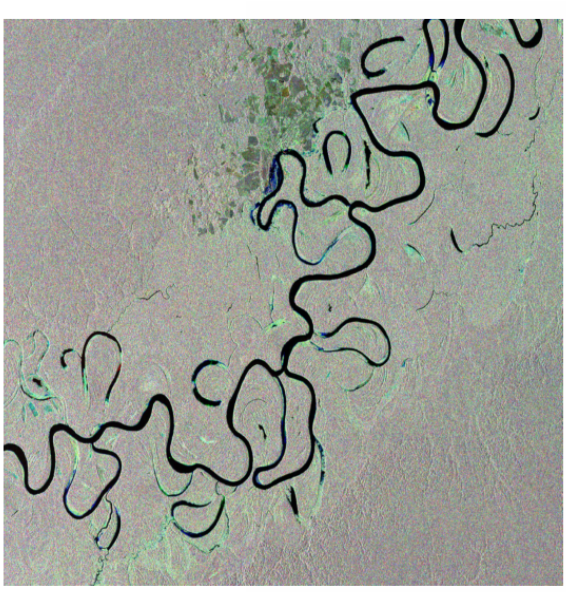
Example 2: Great Lakes

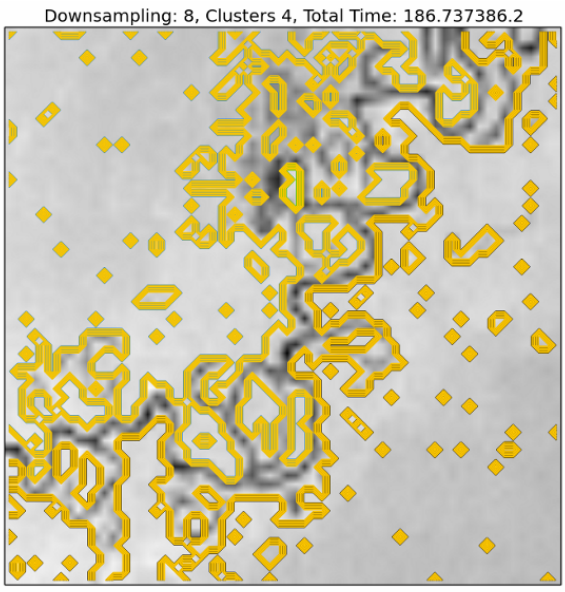


The second example is a satellite image of the famous Great Lakes of the northeastern United States. The number of clusters was set to a high 6 in hopes that the clustering system would identify each lake separately. The result, while certainly not perfect, is still impressive: despite the downsampling, the clustering algorithm correctly identified Lake Michigan (green) Lake Huron (cyan) as separate, single structures. Problems include the fact that the smaller Lake Erie and Lake Ontario were identified as the same object, the small part of Lake Superior that was included was grouped with Lake Michigan, and a lot of extraneous dark edges were grouped as parts of these lakes. The other clusters (yellow, orange) were essentially junk. As in the first example, weighting the algorithm to take spacial similarity into account and

decreasing the downsampling would probably lead to a better result.

Example 3: Winding River





This last example features a single, winding and splitting river with many small and convoluted pieces. The clustering algorithm succeeded in capturing both the main part of the river and the smaller tributaries that feed it and the splits in it as part of the same contiguous structure. For some reason, with the number of clusters set to 4, the river was actually separated out into two often neighboring parts. Nevertheless, this example may actually be the most successful; so far as we can tell, few spurious features were included in the lake, and despite its topological complexity, the algorithm described the shape of the river very well. Note, however, that of all the examples, this one took by far the most time to compute.

Discussion and Conclusions

The relatively simple tool we were able to build in two weeks was nevertheless robust enough to solve problems of identifying complicated structures and grouping objects that a naive approach might not immediately consider related (eg: the convex parts of Lake Michigan) together. This was satisfying to accomplish and see and is evidence to us that spectral clustering's reputation for flexibility and applicability is well-deserved. On the other hand, many of the features we were hoping to implement, such as a less costly and more nuanced notion of similarity and the ability to track changes in features over time, did not come to fruition due to time constraints. For the most part, our hypothesis that geographical distance is not particularly relevant to grouping map objects is undermined by these examples. Overall, we look upon this project as a very fruitful two weeks. Were we given the opportunity to continue, we feel confident our system could be made even more robust and useful.

References

[1] Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. The planar k-means problem is np-hard. In *WALCOM: Algorithms and Computation*, pages 274–285. Springer, 2009.

[2] Aarti Singh and Eric Xing. Clustering. In *Machine Learning 10-701/15-781*. Carnegie Mellon University, http://www.cs.cmu.edu/ aarti/-Class/10701/slides/Lecture13.pdf, 2010.

[3] Aarti Singh and Eric Xing. Spectral clustering. In *Machine Learning 10-701/15-781*. Carnegie Mellon University, http://www.cs.cmu.edu/ aarti/-Class/10701/slides/Lecture21_2.pdf, 2010.

[4] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.