B-Matching for Spectral Clustering

Tony Jebara and Vlad Shchogolev

Department of Computer Science, Columbia University, New York NY 10027, USA

Abstract. We propose preprocessing spectral clustering with b-matching to remove spurious edges in the adjacency graph prior to clustering. B-matching is a generalization of traditional maximum weight matching and is solvable in polynomial time. Instead of a permutation matrix, it produces a binary matrix with rows and columns summing to a positive integer b. The b-matching procedure prunes graph edges such that the in-degree and out-degree of each node is b, producing a more balanced variant of k-nearest-neighbor. The combinatorial algorithm optimally solves for the maximum weight subgraph and makes subsequent spectral clustering more stable and accurate. Experiments on standard datasets, visualizations, and video data support the use of b-matching to prune graphs prior to spectral clustering.

1 Introduction

Clustering is an important tool in the machine learning portfolio, particularly in unsupervised settings. Traditional approaches to clustering include iterative methods such as k-means and Expectation Maximization (EM) which make parametric assumptions about the data and can be easily confounded by local minima during their typically greedy optimizations. Recently, spectral clustering [9,6] methods have gained prominence as principled relaxations of the NP normalized cut clustering problem. These algorithms typically involve finding the top eigenvectors after processing an affinity matrix built from pairwise similarities between points in a dataset. This affinity matrix can be seen as a weighted graph. Essentially, spectral clustering makes an appeal to spectral graph theory [1] and approximates the NP-complete normalized cut procedure. Since a user need only specify a similarity function, spectral clustering methods are non-parametric and avoid explicit assumptions about the generative model of the data. Furthermore, spectral clusterings are not plagued by local minima and often outperform traditional greedy parametric clustering methods. Also, unlike many greedy methods, spectral methods enjoy polynomial run-time guarantees. Finally, spectral clustering produces the same result despite permutation and reordering of input points (unlike some greedy methods such as single-linkage clustering that process data-points in a sequential manner).

Currently, a gamut of spectral clustering algorithms are available and exhibit some variability in their performance on real-world datasets. In this article, we propose a pre-processing of the weighted graph of pairwise similarities. This can be done prior to *any* spectral clustering method. This pre-processing involves b-matching [5], a permutationally invariant (i.e. independent of the ordering of the points in the dataset) combinatorial procedure which eliminates edges in the weighted graph. Conveniently, b-matching on a weighted graph is solvable in polynomial time. The procedure finds the maximum weight subgraph in the original graph where each vertex has an in-degree and an out-degree of b. By preceding spectral clustering with b-matching, we reduce the mismatch the spectral clustering has to an exact normalized cut solution. We conjecture that removing edges optimally via b-matching makes the spectral clustering relaxation more closely approach the NP-complete normalized cut solution. Another argument which is often cited in the nonlinear manifold embedding literature is that the similarity metric being used is only locally reliable. It is unreliable if points are distant or produce low edge weight in the graph [8]. In fact, embedding methods typically resort to a k-nearest neighbor method for pruning the weighted similarity graph which can be seen as a greedy variant of b-matching. Therefore, a b-matching graph pruning procedure can compensate for a poor choice of the similarity metric used in spectral clustering.

2 Spectral Clustering

Assume we are given N samples, $\mathbf{x}_1, \ldots, \mathbf{x}_N$ where each datum is in a sample space $\mathbf{x}_i \in \mathcal{X}$. Also assume we can readily compute an affinity between pairs of samples via the function $k(\mathbf{x}_i, \mathbf{x}_j)$. Consider a matrix $A \in \mathbb{R}^{N \times N}$ of affinities between all pairs of points in the dataset such that $A_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ and $A_{ii} = 0$. This matrix describes a fully connected graph G with N vertices V and $N \times N$ edges E. The edge between node i and node j has weight A_{ij}. Without loss of generality, assume that the points $\mathbf{x}_i$ are in d-dimensional Euclidean space $\mathbb{R}^d$ and the similarity function is merely a radial basis function kernel $k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma^2)$. Therefore, $A_{ij} = A_{ji} \geq 0$. Although we focus on binary clustering, multi-category extensions are straightforward. Given a weighted graph G, a good clustering criterion is the minimum normalized cut [7]. Consider a subset $B \subset V$ of the full vertex set V. Normalized cut is the NP-complete minimization of the cost:

$$NCUT(B) = \frac{\sum_{i \in B, j \in V/B} A_{ij}}{\sum_{i \in B, j \in V} A_{ij}} + \frac{\sum_{i \in V/B, j \in B} A_{ij}}{\sum_{i \in V/B, j \in V} A_{ij}}.$$

To make the problem tractable, [9] provide a relaxation by solving for a real valued solution instead of a discrete vertex selection (or cut). This approach efficiently approximates normalized cut. First, we compute the $N \times N$ diagonal matrix $D_{ii} = \sum_j A_{ij}$. We represent the discrete set B via the indicator vector $\mathbf{y} \in \mathbb{R}^N$. This vector is defined as $\mathbf{y}(i) = \sqrt{d_{V/B}/d_B d}$ if node i is in B and otherwise $\mathbf{y}(i) = -\sqrt{d_B/d_{V/B} d}$. Here, we take $d = \sum_i D_{ii}$ and $d_B = \sum_{i \in B} D_{ii}$. Normalized cut finds a discrete $\mathbf{y}$ vector that minimizes $\mathbf{y}^T(D - A)\mathbf{y}$ subject to $\mathbf{y}^T D\mathbf{y} = 1$ and $\mathbf{y}^T D\mathbf{e} = 0$. Since this is an intractable, we solve for a real-valued $\mathbf{y}$ vector instead. This is done by via the generalized eigenvalue system $(D-A)\mathbf{y} = \lambda D\mathbf{y}$. We get $\mathbf{y}$ as the second smallest eigenvector of the eigensystem. The scalar values of $\mathbf{y}$ determine which nodes belong to the cut.

In practice, we will use a variant of the spectral clustering algorithm above [6]. This variant is as follows. First, compute A and D and the normalized Laplacian $D^{-1/2}AD^{-1/2}$. Second, find the k largest eigenvectors of L and form the matrix $X \in \mathbb{R}^{N \times k}$ by stacking them. Third, form the matrix Y from X by $Y_{ij} = X_{ij}/\sqrt{\sum_j X_{ij}^2}$. Fourth, treat each row of Y as a point in $\mathbb{R}^k$ and cluster them into k clusters using k-means. Fifth, assign the i'th point in the dataset the same cluster label that the i'th row of the Y matrix was assigned. There is evidence this variant of spectral clustering has better empirical performance as well as theoretical justification. This is because it exploits a larger eigengap in the eigensystem which prevents eigenvectors from rotating arbitrarily. Eigenvectors with similar eigenvalues can be rotated within the subspace and can then become unreliable for clustering. The most computationally demanding aspect of these spectral clustering methods is the $\mathcal{O}(N^3)$ eigensystem solution.

Clearly, the performance of spectral clustering algorithms hinges on the input weight matrix A. But, this matrix may be corrupted with poor pairwise affinity values. This intuition is also relevant for embedding methods [8] which also encourage pruning spurious edges from a weighted graph, typically using a k-nearest-neighbor method. One argument for pruning is that the similarity metric is only locally valid and becomes unreliable when points are distant from each other or produce low edge weight in the graph. Furthermore, a large eigengap is desirable for eigenvector stability and would emerge from an affinity matrix A that had a binary clustering structure. For instance, consider the binary clustering problem with an indicator vector $\mathbf{y} \in \mathbb{R}^N$ where $\mathbf{y}(i) = \pm 1$. Also, assume that the clustering is balanced such that $\sum_i \mathbf{y}(i) = 0$. The resulting *ideal affinity matrix* is $A_{ij} = \frac{1}{2}(\mathbf{y}(i)\mathbf{y}(j) + 1)$. In other words, use high affinity between points in the same class and none between points from different classes. The matrix A will be binary with rows and columns summing to $b = N/2$. Clearly, the L matrix in the spectral clustering procedure will exhibit a large and stable eigengap if fed such an *idealized affinity matrix.* Is it possible to achieve these goals by pre-processing the A matrix without straying too far from the original information it carries? We suggest using b-matching, a combinatorial optimization procedure that deletes some low-edge weight entries from the A matrix to produce an *idealized affinity matrix.* This is done while keeping strong edges to produce the maximum total weight sub-graph from the original A.

3 Weighted B-Matching

We will use b-matching to prune and binarize the affinity matrix A in the previous section. Assume we are given a weighted general graph G with nodes V and edges E. An edge connecting node i to j has weight A_{ij}. The maximum weight b-matching problem [7] is a maximum weight subgraph of G such that the degree of each vertex in the subgraph is b. This is a special case of the *degree-constrained subgraph problem.* We have the following combinatorial optimization.

Given a weight matrix $A \in \mathbb{R}^{|V|\times|V|}$, find a binary matrix $P \in \mathcal{B}$ where $\mathcal{B}$ is the space of binary matrices $\{0,1\}^{|V|\times|V|}$ that maximizes the following:

$$\max_{P\in\mathcal{B}} \sum_{ij} P_{ij} A_{ij} \;\; s.t. \sum_i P_{ij} \; = \; \sum_j P_{ij} \; = \; b. \tag{1}$$

The above is essentially a balanced variant of k-nearest neighbor. Nearest-neighbor methods only enforce the row constraint $\sum_i P_{ij} = b$ instead of enforcing both row and column summation. Meanwhile, b-matching ensures that each point has b neighbors and only b other points may choose it as a neighbor. This prevents, for example, a single centrally-located point from dominating the data and acting as a neighbor to too many other points. B-matching is also a generalization of the 1-matching problem or linear assignment problem (LAP) which finds a permutation matrix P (i.e. $b = 1$). LAP is solvable via the Kuhn-Munkres or Hungarian method in $\mathcal{O}(|V|^3)$ time. Tutte [10] shows that is possible to transform an instance of the b-matching problem into a 1-matching problem on general graphs. The later can be solved by the Blossom algorithm, a linear programming formulation developed by Edmonds [2]. A direct linear programming approach to general matching is not possible because one cannot guarantee that the algorithm will produce integral solutions. Edmonds solves this issue by adding an exponential number of constraints to the linear program. The added constraints enforce that no odd-length circuit (called a blossom) could have more matched edges than appropriate. These constraints are unnecessary for bipartite matching since bipartite graphs don't have odd circuits. The Blossom algorithm uses the primal-dual method to solve the linear program. A special search procedure avoids working explicitly with the exponential number of constraints. In particular, the algorithm works by shrinking blossoms into single pseudo-nodes, ensuring that the algorithm can detect a way to improve the current matching. The primal-dual method starts with a feasible dual solution and searches for a feasible primal solution that satisfies the complementary slackness conditions. The search is performed on a restricted primal (RP) problem. If unsuccessful, the dual solution is updated via an update rule, and the entire procedure is repeated. The algorithm moves from one basic feasible RP solution to another. The cost decreases monotonically and no basis is repeated. Hence the algorithm will terminate in polynomial time. The b-matching problem can also be solved directly by a variant of the blossom algorithm. One difference during b-matching is the blossom structure is more complicated than just odd circuits [5]. The current best exact algorithm for b-matching is due to Gabow [4] and runs in $\mathcal{O}(\min(|E|\log|V|, |V|^2)|V|b)$ time. Recently, this and other matching problems have been reformulated in terms of Balanced Network Flow problems [3]. We used the b-matching software package: www.math.uni-augsburg.de/opt/goblin.html.

4 B-Matching for Spectral Clustering

We will use the b-matching procedure to find an *idealized affinity matrix* that is closest to (in the Frobenius norm sense) the original affinity matrix. Suppose we have N objects and we wish to find a vector $\mathbf{y} \in \{-1, 1\}^N$ which gives a binary clustering of the objects. If $\mathbf{y}$ were the true labeling of the objects, then we would like our kernel matrix to be $P^* = \frac{1}{2}(\mathbf{y} \cdot \mathbf{y}^T + 1)$. However, the kernel or affinity matrix is normally computed from real world data, and hence does not have the simple structure of P^*. We address this by finding an approximation to A that has the structure of matrix P^*. In particular, we find a binary symmetric matrix with rows and columns summing to b which we will typically set to $b = \frac{N}{2}$. We thus have the following minimization:

$$\min_P \|P - A\|_F^2 \text{ subject to } \sum_i P_{ij} = \sum_j P_{ij} = b \quad P_{ij} \in \{0, 1\}.$$

The cost function is simplified as follows:

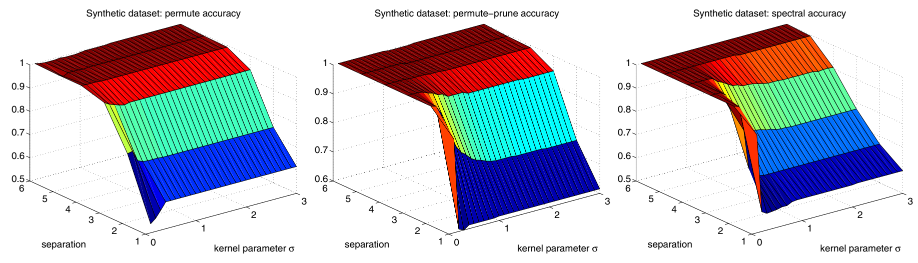
$$\|P - A\|_F^2 = tr[PP^T] + tr[AA^T] - 2\ tr[P^T A] = Nb + \|A\|_F^2 - 2\ tr[P^T A].$$

Only the last term depends on P. Note that $tr[P^T A] = \sum_{i,j} P_{ij} A_{ij}$ Therefore, the Frobenius norm minimization problem is exactly equivalent to b-matching maximization in Equation 1. In other words, we can equivalently solve for the b-matching that is most correlated with the input affinity weight matrix A. Thus, we precede spectral clustering algorithms with this b-matching procedure applied to the A affinity matrix and get the binary b-matching matrix P. For binary clustering, we set $b = N/2$ just as in the *idealized affinity matrix*. This essentially encourages a balanced clustering problem. We then use the P matrix in a spectral clustering procedure instead of the original A matrix. Since this matrix is binary, this may result in lost information so we also consider using the matrix $P \otimes A$ (where $\otimes$ is the element-wise product of the two matrices) as the affinity matrix for spectral clustering. This leads to two variations on spectral clustering. We find P via b-matching with weights in A and with $b = N/2$. One approach, **permute** performs spectral clustering with P as the affinity matrix. The other approach, **permute-prune** instead uses the element-wise product of $P \otimes A$ as the affinity matrix. We compare these two approaches to direct spectral clustering called **spectral** on the original affinity matrix. We also compare the performance where we replace the b-matching procedure with a k-nearest-neighbor procedure. Here, k is set to equal b to obtain a similar effect. This approach yields two more competitor methods, namely **knn** and **knn-prune**. Note, to go beyond binary clustering we find M clusters. This is done by setting $b = N/M$ in the b-matching procedure and using multi-class spectral clustering tools. We next show experiments with these various spectral clustering methods.

5 Experiments

To evaluate the clustering schemes, we applied them to classification problems where labels are hidden from the learning algorithm. The labels are used afterward to report a classification accuracy by seeing how well the clustering algorithms agree with the true labeling.

In a synthetic experiment, we generated data along two S-shaped curves with a different spread value c between them. The larger the value of c is, the further apart the two S-curves are. The desired classification is to separate each S-curve from the other. As c gets small, clustering algorithms will misplace a point from one curve onto another. We also vary the σ parameter in the RBF to see its effect on the algorithms. Figure 1 shows the accuracy of (a) the permute method, (b) the permute-prune method which does best for the range of settings for c and σ and (c) the traditional spectral clustering method which does worst.



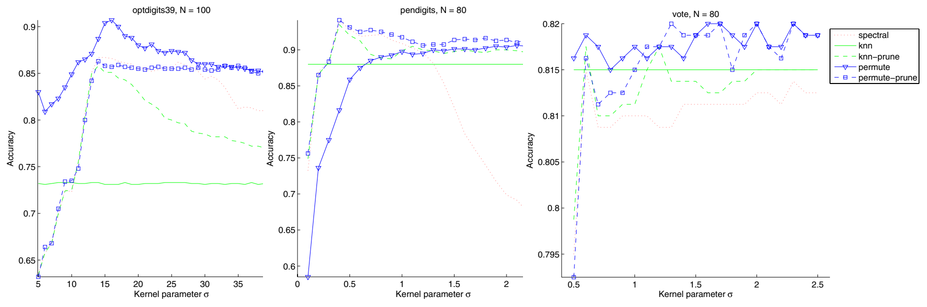
(a) Permute (b) Permute-Prune (c) Spectral

Fig. 1. Classification accuracy for c and σ settings

We evaluated the clustering methods on UCI binary classification problems across varying RBF σ settings. Figure 2(a) shows the accuracy on the UCI OptDigits dataset. The average accuracy over ten folds of size $N = 100$ training examples is shown. Permute is the top algorithm throughout and peaks at over 90% at the best choice of σ. In the PenDigits dataset in Figure 2(b), we down-sampled to $N = 80$ training examples and show the mean accuracy for all algorithms. Permute-prune is the top performer here. For the UCI Vote dataset with 80 training examples, Figure 2(c), shows both permute and permute-prune performed well and above the other methods.

We finally evaluated the clustering methods on video sequences composed of two scenes. In a scene, actors move and cameras pan smoothly. However, more abrupt changes occur during a scene transition (i.e. a sudden cut). If images are represented as vector coordinates in a Euclidean space (i.e. by rasterizing each image into a vector) a video sequence of two scenes looks like two nonlinear strands that are highly intertwined yet disconnected. Therefore, a good clustering algorithm should separate the two scenes. We obtained video sequence [1] and identified scene transitions manually to obtain a labeled classification problem. We evaluated the various clustering schemes as they discover the labeling just from RBF kernels between pairs of vectorized images. One class is the first scene and the other is the second scene in a video sequence. The order of the frames is randomized and we select $N = 48$ random samples. Interestingly, nearby frames

[1] The video, a real-life parody of the Simpsons television show, is available at: video.google.com/videoplay?docid=-2231271827736577327&q=simpsons+intro.



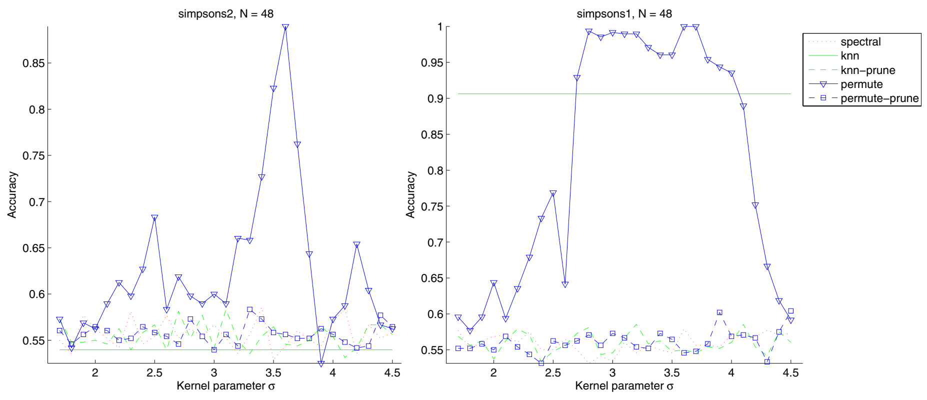
a) OptDigits dataset b) PenDigits dataset c) Vote dataset

Fig. 2. Accuracy of Spectral Clustering Procedures on UCI Data

have a strong RBF similarity values with each other yet only weak similarity values to other frames in the dataset. Thus, if the video sequence was sorted in time, we would expect the affinity matrix A to look like a thin banded matrix. We randomize the set of training examples to obtain an average classification accuracy for all five algorithms over 10 folds under various settings of the RBF σ. The goal is to split the video into the two scenes, one with the actress Maggie and one with the actress Marge. Figure 3(a) shows that permute is the only strong clustering method with almost 90% accuracy while the other methods perform close to random chance. A similar experiment was performed with another pair of scenes in the video sequence. One scene contains the actor Bart and one scene contains the actor Homer. Figure 3(b) shows the results of the various clustering approaches. Clearly permute does best and can achieve 100% accuracy (although knn does approach it somewhat). The b-matching solution is able to lock onto the two clusters or threads of video sequences. We conjecture that these video examples are actually reminiscent of our toy S-curves example since each scene forms a windy nonlinear curve from the nearby adjacent frames. These two video curves may be so close together that it is impossible to uncover the clustering unless an aggressive b-matching procedure finds the binary maximum weight b-matching and propagates information across the nearby neighbors on the nonlinear curves.

6 Discussion

We showed how b-matching, a polynomial time combinatorial algorithm, can prune and binarize the weighted affinity graph prior to spectral clustering. This procedure improves the accuracy of spectral clustering in applied problems and does better than a k-nearest-neighbor pre-processing. The preprocessing also reduces the variability of spectral clustering over different splits of a dataset. Furthermore, the performance advantage is maintained over a wider range of parameters of the similarity function (i.e. the RBF σ parameter). The b-matching algorithm takes cubic time which is close to the cost of the eigensystem solvers



(a) Maggie vs. Marge Scene (b) Homer vs. Bart Scene

Fig. 3. Clustering Accuracy on Simpsons Video Sequences

that underly spectral clustering. Nevertheless, we are investigating faster approximate b-matching algorithms to further reduce computational limitation. Finally, we are also looking into theoretical arguments for b-matching. [2]

References

1. Chung, F.: Spectral Graph Theory. American Mathematical Society's CBMS Regional Conference Series in Mathematics **92** (1997)
2. Edmonds, J.: Paths, trees and flowers. Canadian Journal of Mathematics **17** (1965)
3. Fremuth-Paeger, C., Jungnickel, D.: Balanced network flows. I. A unifying framework for design and analysis of matching algorithms. Networks **33** 1 (1999) 1-28
4. Gabow, H.: An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. STOC '83: Proceedings of the fifteenth annual ACM symposium on theory of computing (1983) 448-456
5. Muller-Hannemann, M., Schwartz, A.: Implementing weighted b-matching algorithms: insights from a computational study. J. Exp. Algorithmics **5** (2000) 1084-6654
6. Ng. A., Jordan, M., Weiss, Y.: On spectral clustering: Analysis and an algorithm. Neural Information Processing Systems **14** (2001)
7. Papadimitriou, C., Steiglitz, K.: Combinatorial Optimization: Algorithms and Complexity. Prentice-Hall (1982)
8. Roweis, S., Saul, L.: Nonlinear Dimensionality Reduction by Locally Linear Embedding. Science, **290** 5500 (2000)
9. Shi, F., Malik, J.: Normalized Cuts and Image Segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence, **22** 8 (2000) 888-905
10. Tutte, W.T.: A short proof of the factor theorem for finite graphs. Canadian Journal of Mathematics **6** (1954) 347-352

[2] This research was funded by NSF grants CCR-0312690, IIS-0347499 and IIS-0093302.