AMATH 582: ASSIGNMENT 4

JON STAGGS

Applied Mathematics M.S. Student, University of Washington, Seattle, WA
jrstaggs@uw.edu

Abstract. Given the political party of individuals and their voting records we seek to build two models which can classify an individual's political party based on their voting records. We use the Graph Laplacian to build a spectral clustering model. We also use the Laplacian Embedding to build a Semi-Supervised Learning regression model.

1. Introduction and Overview

In order to build our models we rely on the Graph Laplacian. For our spectral clustering we will utilize the fact the the sign of the Fiedler vector contains the indicators of the clusters. For our semi-supervised learning model we use a linear regression on our Laplacian embedding as our predictor for all features. We then discuss the implementation and relevant packages used to build and test the models. In the computational results we discuss finding an optimal variance for the weight function of our graph and present the accuracy of the models ability to classify the labels based on voting records.

2. Theoretical Background

2.1. **Similarity Graphs and Weight Functions.** We use the notation and follow the flow of information that comes from [3]. We denote our data, $X = \{\underline{x}_0, \underline{x}_1, ..., \underline{x}_{N-1}\}$. The underline indicates a column vector of features. We define a weighted, undirected graph,

$$G = \{X, W\} \tag{1}$$

where $\underline{x}_j$ are vertices of G and W is a weight matrix whose elements, $w_{i,j}$, are associated to the weights of edges that connect x_i to x_j. Since the graph is undirected $w_{i,j} = w_{j,i}$ and so W is a symmetric matrix. To define the elements of W we consider a weight function, $\eta : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$. In our use case we define the weight function to be the gaussian,

$$\eta(t) = e^{-\frac{t^2}{2\sigma^2}}. \tag{2}$$

To define the elements we also need to define a norm to compute the distance between two points. We choose the typical Euclidean norm. We then have,

$$w_{i,j} = e^{\frac{||x_i - x_j||_2^2}{2\sigma^2}}. \tag{3}$$

2.2. **Graph Laplacian and Embedding.** We now define the Graph Laplacian. We define the degree vector, $\vec{d} \in \mathbb{R}^n$, and associated diagonal degree matrix, $D \in \mathbb{R}^{n \times n}$,

$$d_j = \sum_{i=0}^{n-1} w_{i,j}, \;\; j = 0, ..., n-1, \quad D = \begin{bmatrix} d_1 & & & \\ & d_2 & & \\ & & \ddots & \\ & & & d_{n-1} \end{bmatrix}. \tag{4}$$

Date: December 29, 2023.

The diagonal degree vector entries are the row sums of corresponding of the weight matrix. We now have the Graph Laplacian, $\tilde{L}$, as [3]

$$\tilde{L} = D - W. \tag{5}$$

We wish to define a mapping F so clustering is more effective than on the original data X. To find this F, we perform an eigendecomposition on $\tilde{L}$ - this is guaranteed since $\tilde{L}$ is negative definite and symmetric. Then $\tilde{L} = \tilde{Q}\tilde{\Lambda}\tilde{Q}^T$. We write,

$$\tilde{Q} = [\tilde{q}_0 \ \tilde{q}_1 \ \dots \ \tilde{q}_{n-1}] \tag{6}$$

where q_i are column vectors in $\mathbb{R}^n$ which correspond to the eigenvectors of $\tilde{L}$. We define our feature map, F, which we call the Laplacian Embedding [3]. This is an embedding into a lower dimensional space say $\mathbb{R}^M$ so we have that,

$$F(\underline{x}_j) = \begin{bmatrix} \tilde{q}_{1,j} \\ \vdots \\ \tilde{q}_{M,j} \end{bmatrix}. \tag{7}$$

2.2.1. *Properties of Graph Laplacian.* To illustrate the properties of the Graph Laplacian and what the eigenvalues/vectors can tell us we consider data that has two distinct clusters. Our graph, G, can then be written as $G = G_0 \cup G_1$ each with Graph Laplacians L_0, L_1 respectively. We can form the block diagonal matrix

$$L = \begin{bmatrix} L_0 & 0 \\ 0 & L_1 \end{bmatrix}. \tag{8}$$

L_0 and L_1 have zero eigenvalues corresponding to constant eigenvectors. Let I_{N_0} and I_{N_1} be constant egienvectos corresponding to these zero eigenvectors for the respective L_i. We then have,

$$L = \begin{bmatrix} L_0 & 0 \\ 0 & L_1 \end{bmatrix} \begin{bmatrix} I_{N_0} \\ 0 \end{bmatrix} = \begin{bmatrix} L_0 & 0 \\ 0 & L_1 \end{bmatrix} \begin{bmatrix} 0 \\ I_{N_1} \end{bmatrix} = 0. \tag{9}$$

Notice that the vector $[I_{N_0} \ 0] = \tilde{q}_0$ and $[0 \ I_{N_1}] = \tilde{q}_1$. Hence, these are precisely the indicators of the of the sets X_0 and X_1. In practice we get a linear combination of these two eigenvectors and $\tilde{q}_0 = I$ so the important information ends up being stored in $\tilde{q}_1$ and this is called the Fiedler vector [3].

2.2.2. *Spectral Clustering.* In our case we will assign our labels, Republican or Democrat, as ∓ 1. Once we form the Graph Laplacian and obtain our Fiedler vector then we are precisely in the scenario in the section immediately above. Hence we can classify a point in a cluster based off of the sign of the Fiedler vector.

2.3. **Semi-Supervised Regression.** The goal of semi-supervised learning is the classify all labeled data while only being trained on a limited number of labeled data. The relevance of these problems is that it is easy to generate data, but hard/expensive to label data. We combine the techniques from supervised and unsupervised learning by combining regression with the Laplacian Embedding/clustering. We begin in the same way as Kernel regression: we want to find a function $F(x)$ such that

$$f(\underline{x}) \approx \sum_{j=0}^{J} c_j F_j(x) \tag{10}$$

so that $f(\underline{x}_i) \approx y(\underline{x}_i)$ for $M \leq j \leq n-1$. This amounts to saying that we want to approximate our output only on unlabelled data. The question we have now is how we pick F_j. Of course, we take this to be our Laplacian Embedding. We then have the regression problem [2],

$$\hat{\beta} = \text{argmin}_{\beta \in \mathbb{R}^M} ||F_{M,J}\beta - b|| \tag{11}$$

where $F_{M,J} \in \mathbb{R}^{M \times J}$ is our Laplacian embedding and b is our labeled data both corresponding to the lower dimension M. Then our classifier is $\hat{y} = \text{sign}(F(x)\hat{\beta})$ [3].

3. Algorithm Implementation and Development

We use Python version 3.8 [9] with SciKit-learn [5], NumPy [1], SciPy [10], Pandas [8] and MatplotLib [4] libraries and Sypder as our IDE [6]. We will briefly discuss the preprocessing and then describe the implementation.

3.1. **Preprocessing and Set Up.**

For preprocessing we take the guidance from [2], we encode Democrats as $+1$, Republicans as -1, and the missing information which is marked as "?" is replaced with a 0. We also encode the "yea" or "nay" indicating whether an individual voting for or against a bill with a 1 and -1 respectively. To do this encoding we convert our textfile into a Pandas dataframe and use Pandas *replace* method. We also note that no centering or scaling of the data takes place as this can adversely affect the spectral clustering coming from the eigenvector/Fiedler vector. We can then use Pandas *to_numpy* method to convert our dataframe into a numpy array.

3.2. **Functions from Relevant Packages.**

We use SciPy's *distance_matrix* method from the *spatial* class which computes the pairwise distance between all points of our features X and forms a matrix that contains this distance information. This will help us to build our Graph Laplacian. We also use NumPy's *eigh* function which leverages the fact that our Graph Laplacian is Hermitian (Symmetric) to more efficiently compute the eigenvalues and eigenvectors of our Graph Laplacian. We utilize SciKit Learn's *LinearRegression* and *fit* method from the *linear _model* class to compute the the coefficients of our linear regression. We have covered regression in more detail in [7]

3.3. **Implementation.**

3.3.1. *Spectral Clustering.*

We first compute the distance matrix using *distance_matrix* and define the weight function eqn. (2). Our weight matrix is then eq.. applied to our distance with a variance σ. We then compute the diagonal degree matrix eqn. (4). Then using *eigh* we compute the eigenvectors. We then grab the second eigenvector, our Fiedler vector, and identify the sign ($\pm$) of each entry. This corresponds to the encoding of the parties. To calculate the accuracy we use the equation from [2] and compute the proportion of incorrect classifications.

To find an optimal σ, σ^*, we simply loop through the above process and change σ. We determine the optimal sigma to be the sigma that corresponds to the highest accuracy.

3.3.2. *Semi-Supervised Learning.*

The Semi-Supervised Learning algorithm begins much the same way as the spectral clustering. In this algorithm we use the optimal sigma value. We compute the weight matrix and diagonal degree matrix to form the Graph Laplacian. We now must compute the embedding. We simply take the first M eigenvectors so that the dimensions are $435 \times M$ then we take a limited number of samples, J, so that the dimensions of our submatrix are $J \times M$. We similarly adapt our label vector Y so that it has dimensions $M \times 1$. We then perform a linear regression which is fitting our submatrix and shortened labeled vector. Note that we do not fit the intercept in the regression. We then loop through multiple M and J values and compute the ssl accuracy by computing the proportion of misclassified individuals.

4. Computational Results

We remark that when we plot our data we sort the data by party so that visualization is easier, however, we do not sort any data while computing the Graph Laplacian or Fiedler vector.

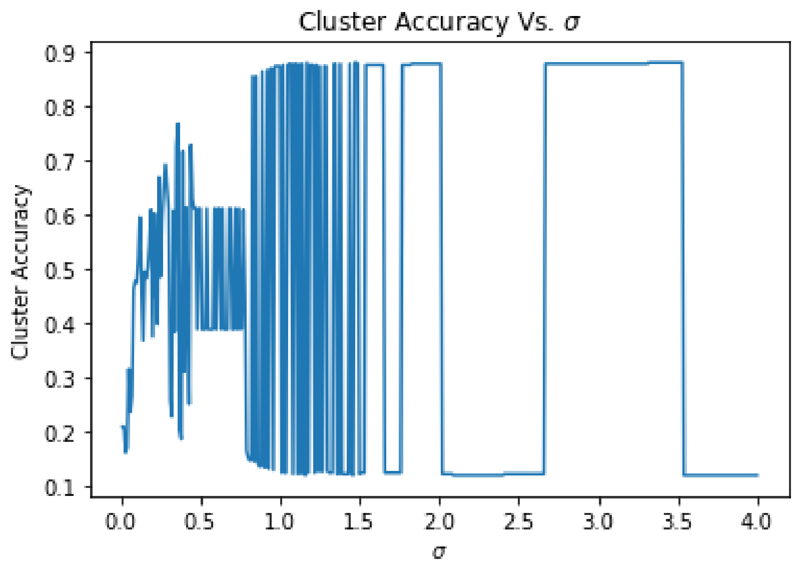

Figure 1. Clustering Accuracy vs. Weight Function Variance, σ.

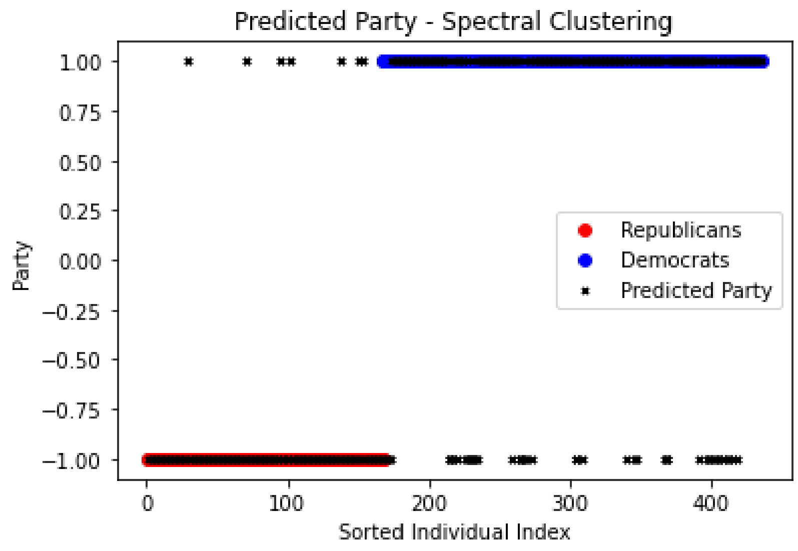

Figure 2. True party and Spectral Clustering Predictions.

4.1. **Optimal Variance.** In Fig. 1 we plot various values of the variance σ compared to the clustering accuracy. We plot values of $\sigma \in (0, 4]$. We computed with a grid spacing of $h = .001$. We find that the optimal sigma, $\sigma^* = 1.159$. For sigmas less than approximately 1.7 we see that there is high frequency oscillations. We also note that "clumps" seem to exist in this region - between $\approx .5$ and $\approx .75$ the clustering accuracy is oscillating between .4 and .6 while between .7 and 1.6 the accuracy bounces between $\approx .11$ and $\approx .88$. This suggests that the weight function is very sensitive to σ values. However, we begin to see more stable regions of σ as it gets larger and we observe regions of σ where the clustering accuracy plateaus and the length of the plateaus increases. This is reminiscent of $\sin(1/x)$ where we have bounded (in magnitude) oscillations, but the period increases.

4.2. **Classification with Spectral Clustering.** The clustering accuracy that corresponds to σ^* is 88.05% . The spectral clustering misclassified 52 individuals. We see from Fig. 2

M	J	# misclassified	SSL Accuracy
2	5	48	88.97 %
2	10	49	88.74 %
2	20	51	88.29 %
2	40	52	88.05 %
3	5	49	88.74 %
3	10	80	81.61 %
3	20	78	82.07%
3	40	70	83.91%
4	5	70	83.91 %
4	10	65	85.06 %
4	20	59	86.43 %
4	40	54	87.59 %
5	5	57	86.90 %
5	10	127	70.80 %
5	20	70	83.91 %
5	40	52	88.05 %
6	5	57	86.90 %
6	10	136	68.74 %
6	20	55	87.36 %
6	40	59	86.44 %

Table 1. Number of Misclassifications and Accuracy for different sized Laplacian Embeddings with Linear Regression.

that the spectral clustering tended to identify more Democrats as Republicans. The spectral clustering identified 45 true Democrats as Republicans while only 7 true Republicans were predicted to be Democrats.

4.3. **Semi-Supervised Learning.** In Table 1, we compare the accuracy with different sized embeddings with different amounts of samples. The most interesting note is that the smallest dimension, 5×2 yielded the highest accuracy for our choice of σ^*. We also note when the dimension of the embedding is $M = 2$ that regardless of the amount of samples that we use. This is powerful in the use case of only having very limited labeled samples. Another loose trend follows "similar likes similar" meaning that when the embedding dimension M is smaller the accuracy tends to be higher when J is smaller and vice versa for larger M and J. For example, when $M = 2, 3$ the smaller J is the better accuracy we see. When $M = 5, 6$ then the accuracy tends to increase as J gets larger. This also points that when $J = 10$ we get particularly bad accuracies when M is larger.

In Fig. 3, we show the predicted values of the parties vs the true party. We can see that the semi-supervised learning seemed to more evenly distribute it's misclassifications in comparision to the spectral clustering. In the semi-supervised learning case the number of misidentified Republicans was 20 (republican predicted to be democrat), and 28 for the misidentified Democrats (democrat predicted to be republican). If we contrast this to the spectral clustering we see that the spectral clustering seemed to believe there was more symmetry in the parties. This might be due to the fact that the mean of the $\pm$ labels is $\approx .2276$ and the optimal variance is just over 1 so we have nearly a standard normal. However, the regression seems to have more symmetry in it's miscalculations.

5. Summary and Conclusions

Using spectral techniques we were able to build two models 1) clustering, 2) semi-supervised learning to predict the political parties of individuals based on their voting records. The SSL was able to slightly outperform the clustering with only 48 misclassified

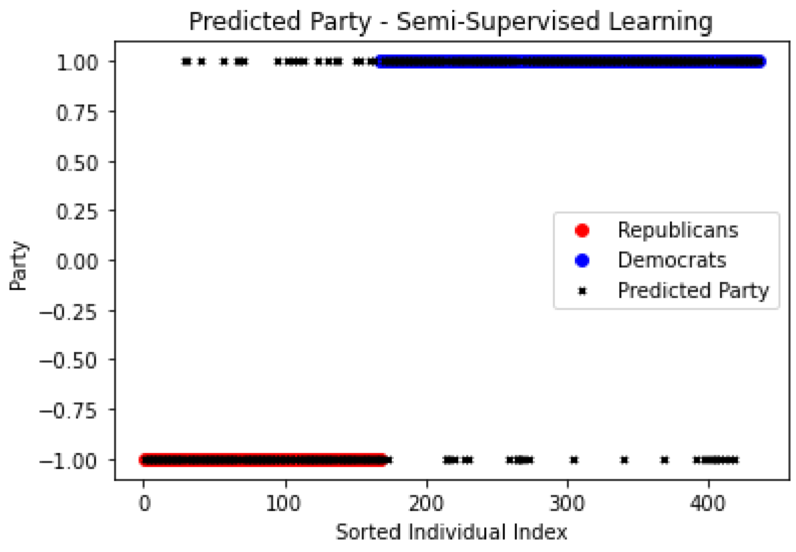

FIGURE 3. True Party and Semi-Supervised Learning Predictions with best performing Laplacian Embedding.

parties where clustering misclassified 52 individual's parties. The true power of the SSL came from only doing these predictions with regressing on 2 dimensions.

Future avenues to explore include understanding the symmetry of the misclassifications in the SSL model vs. the lop sided misclassifications in the clustering model. We also would like to explore why $J = 10$ seems to give particularly nasty results for the SSL when $M = 5, 6$. We would also like to experiment with more values of M and J in the embedding.

Acknowledgements

The author thanks fellow classmate Daniel Leon for helpful discussions about the theoretical aspects of the Graph Laplacian's clustering ability as well as general discussions about the report.

References

[1] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020.
[2] B. Hosseini. Amath 582 assignment 4 sheet. Sheet that contains relevant information for Assignment 4.
[3] B. Hosseini. Amath 582 course notes. Notes for Computational Methods for Data Analysis.
[4] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.
[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
[6] P. Raybaut. Spyder-documentation. *Available online at: spyder-ide.org*, 2009.
[7] J. Staggs. Amath 582 assignment 2. Homework assignment 2 for amath 582.
[8] T. P. D. Team. *pandas-dev/pandas: Pandas*, 2020.
[9] G. Van Rossum and F. L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.
[10] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.