Sims_Kelly_HW1_report

August 27, 2020

Q1 Clustering

Given m data points x^i , i = 1, . . . , m, K-means clustering algorithm groups them into k clusters by mini- mizing the distortion function over $\{r^{ij}, ^j\}$

$$J = \sum_{i=1}^{m} \sum_{j=1}^{k} r^{ij} ||x^i - \mu^j||^2$$

where $r^{ij} = 1$ if x^i belongs to the j-th cluster and $r^{ij} = 0$ otherwise.

1.)

Prove that using the squared Euclidean distance $||x^i - \mu^j||^2$ as the dissimilarity function and minimizing the distortion function, we will have

$$\mu^j = \frac{\sum_i r^{ij} x^i}{\sum_i r^{ij}}$$

Answer

with

$$J = \sum_{i=1}^{m} \sum_{j=1}^{k} r^{ij} ||x^i - \mu^j||^2 \tag{1}$$

taking the derivative of (1) with respect to μ^j

$$\frac{\partial J}{\partial \mu^j} = -2 \sum_{i=1}^{m} \sum_{j=1}^{k} r^{ij} (x^i - \mu^j)$$

$$= \sum_{i=1}^{m} \sum_{j=1}^{k} (-2 r^{ij} x^i + 2 r^{ij} \mu^j) \tag{2}$$

Setting (2) equal to 0 and solving for μ^j yields

$$\sum_{i=1}^{m}\sum_{j=1}^{k}(-2r^{ij}x^i + 2r^{ij}\mu^j) = 0$$

$$2r^{ij}\mu^j = 2r^{ij}x^i$$

$$\mu^j = \frac{\sum_i 2r^{ij}x^i}{\sum_i 2r^{ij}} \tag{3}$$

reducing (3) yields

$$\boxed{\mu^j = \frac{\sum_i r^{ij}x^i}{\sum_i r^{ij}}}$$

2.)

Now suppose we replace the similarity function (the squared $_2$ distance here: $||x^i - \mu^j||^2$ by another distance measure, the quadratic distance (also known as the Mahalanobis distance) $d(x, y) = (x-y)^T(x-y)$, where the given weight matrix $\mathbf{\Sigma}$ is symmetrical and positive definite (meaning that the corresponding d(x, y) > 0 when $x \neq y$). (i) Show (prove) that the centroid in this case will be the same

$$\mu^j = \frac{\sum_i r^{ij}x^i}{\sum_i r^{ij}}$$

Answer

with

$$J = \sum_{i=1}^{m}\sum_{j=1}^{k} r^{ij}(x^i - \mu^j)^T\Sigma(x^i - \mu^j) \tag{1}$$

Noting that $X^TX = X^2$, (1) becomes

$$J = \sum_{i=1}^{m}\sum_{j=1}^{k} r^{ij}\Sigma(x^i - \mu^j)^2 \tag{2}$$

Taking the derivative of (2) with respect to μ yields

$$\frac{\partial J}{\partial \mu} = -2\Sigma\sum_{i=1}^{m}\sum_{j=1}^{k} r^{ij}(x^i - \mu^j)$$

$$= \sum_{i=1}^{m} \sum_{j=1}^{k} -2\Sigma x^i + 2\Sigma \mu^j \tag{3}$$

Setting (3) equal to 0 and solving for μ

$$\sum_{i=1}^{m} \sum_{j=1}^{k} -2\Sigma r^{ij} x^i + 2\Sigma r^{ij} \mu^j = 0$$

$$2\Sigma rij\mu^j = 2\Sigma r^{ij} x^i$$

$$\mu^j = \frac{2\Sigma r^{ij} x^i}{2\Sigma r^{ij}} \tag{4}$$

Reducing (4) yields

$$\boxed{\mu^j = \frac{\sum_i r^{ij} x^i}{\sum_i r^{ij}}}$$

(ii) However, the assignment function will be different – comment how the assignment function r^{ij} should be in this case. Thus, the point is here that, depending on the similarity function, the results of k-means may be different.

Answer

Note that the assignment will be different between the two similarity functions because $d(x, y) = (x-y)^T(x-y) \neq ||x - y||^2$. Although similar, the former (Mahalanobis distance) multiplies the squared differences by a corresponding weight matrix which in turn potentially adjusts what would have been cluster assignment before weight multiplication

3.)

Prove (using mathematical arguments) that K-means algorithm converges to a local optimum in finite steps.

Answer

The first thing to note is that the algorithm iteraties over a training set that is finite. No new points are added, nor are points removed. Centroids are randomly initialized within the latent space. At time τ we iterate over each point and assign them to a centroid (cluster) that minimizes:

$$J_\tau = \sum_{i=1}^{m} \sum_{j=1}^{k} r^{ij} ||x^i - \mu^j||^2$$

The centroid locations are then updated with respect to all points that make up its cluster. This is done by taking the positional mean of the "within cluster points" positions:

$$\mu_\tau^j = \frac{\sum_i r^{ij} x^i}{\sum_i r^{ij}}$$

Upon next iteration $\tau + 1$ we iterate and calculate again

$$J_{\tau+1}$$

Since we moved the centroids closer to their "within cluster points" that minimized J, there's only two possiblities:

1. $J_\tau > J_{\tau+1}$
2. $J_\tau = J_{\tau+1}$

For case (1), if we find that $J_\tau > J_{\tau+1}$, this means we have found cluster groupings with a lower cost at iteration $\tau+1$. The centroids $\mu_{\tau+1}$are then updated accordingly with the new classifications

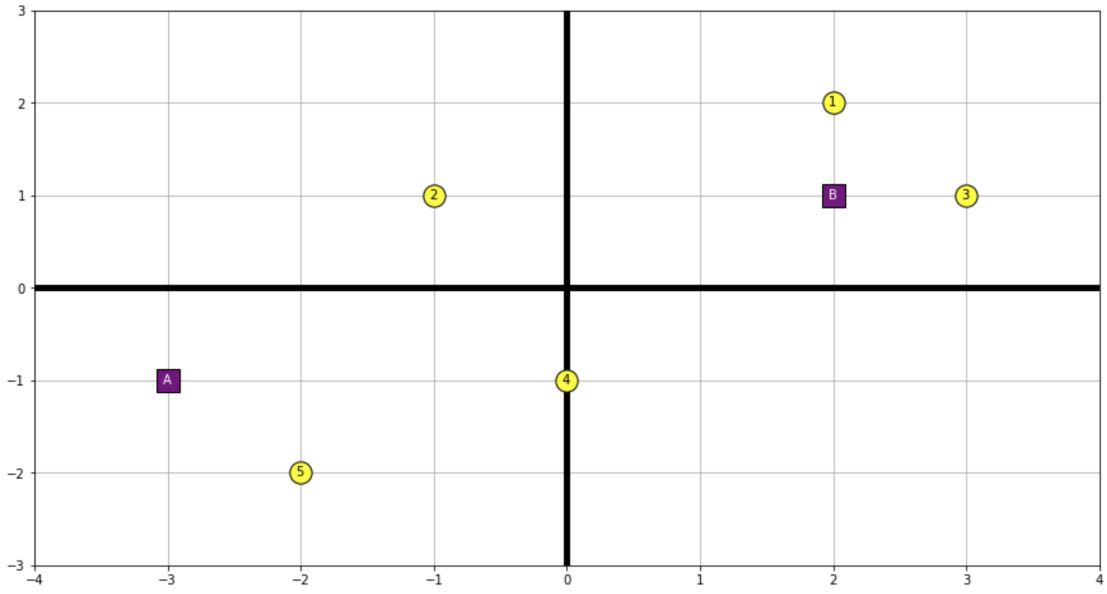
For case (2), if $J_\tau = J_{\tau+1}$ then no points have changed clusters. Therefore, the centroid update equation $\mu_\tau = \mu_{\tau+1}$. Since the centroids didn't update, on iteration $\tau + 2$, it will be true that $J_{\tau+2} = J_{\tau+1} \quad \therefore J_{\tau+1} = J_{\tau+2} = J_{\tau+3}..... = J_{\tau+inf}$. The algorithm has converged. However it is important to note that convergence isn't necessarily at the global minimum due to the random initialization of the centroids.

The key thing to remember is that we are iterating over a finite set of data points that are stationary. Since only (1) & (2) are possible. Eventually the centroids will converge to a position s.t. only (2) is possible.

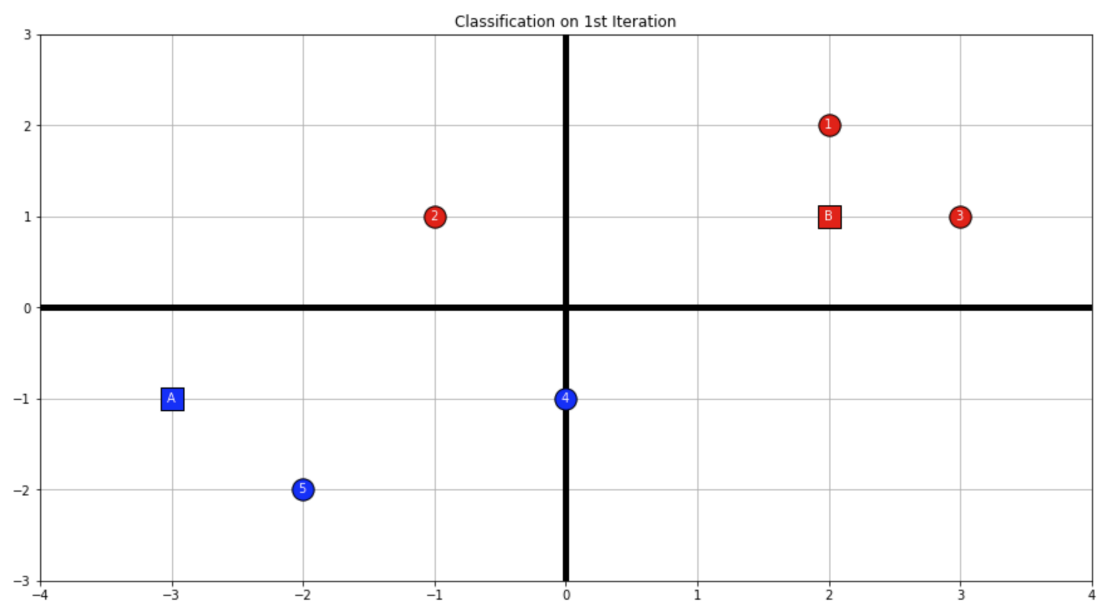
4.)

Calculate k-means by hands. Given 5 data points configuration in Figure 1. Assume k = 2 and use Manhattan distance (a.k.a. the $_1$ distance: given two 2-dimensional points (x 1 , y 1) and (x 2 , y 2), their distance is |x 1 − x 2 | + |y 1 − y 2 |). Assuming the initialization of centroid as shown, after one iteration of k-means algorithm, answer the following questions.

(a.) Show the cluster assignment With initial configuration of the data points as the following:

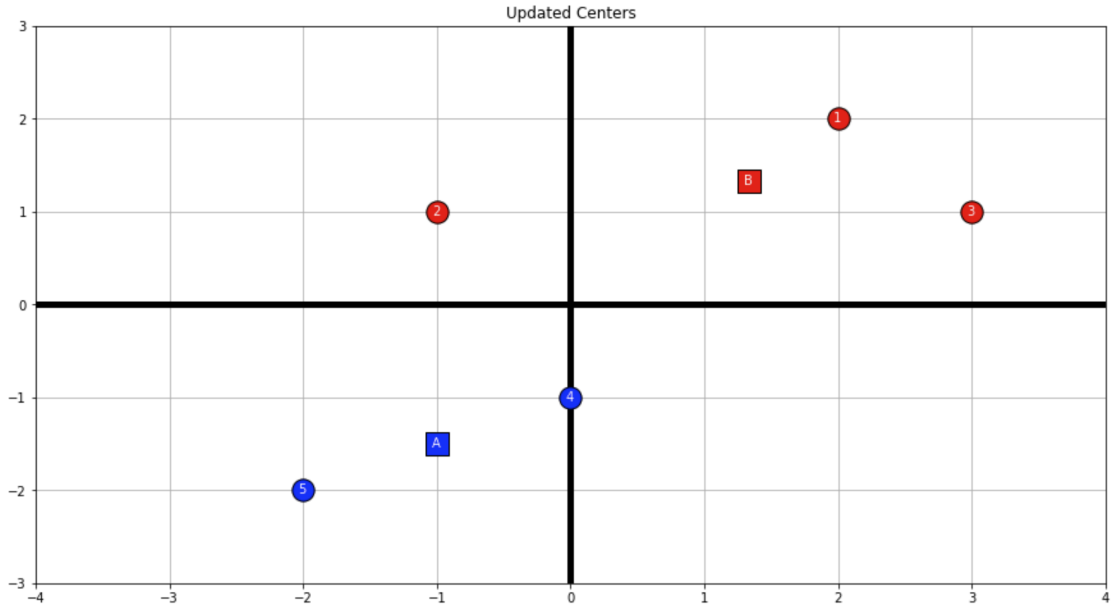


The cluster assignments after the first iteration have the points [(2,2),(-1,1),(3,1)] aka points [1,2,3] assigned to centroid **B** while the points [(0,-1),(-2,-2)] aka points [4,5] are assigned to cluster **A**. The graph belows shows proper cluster assignments

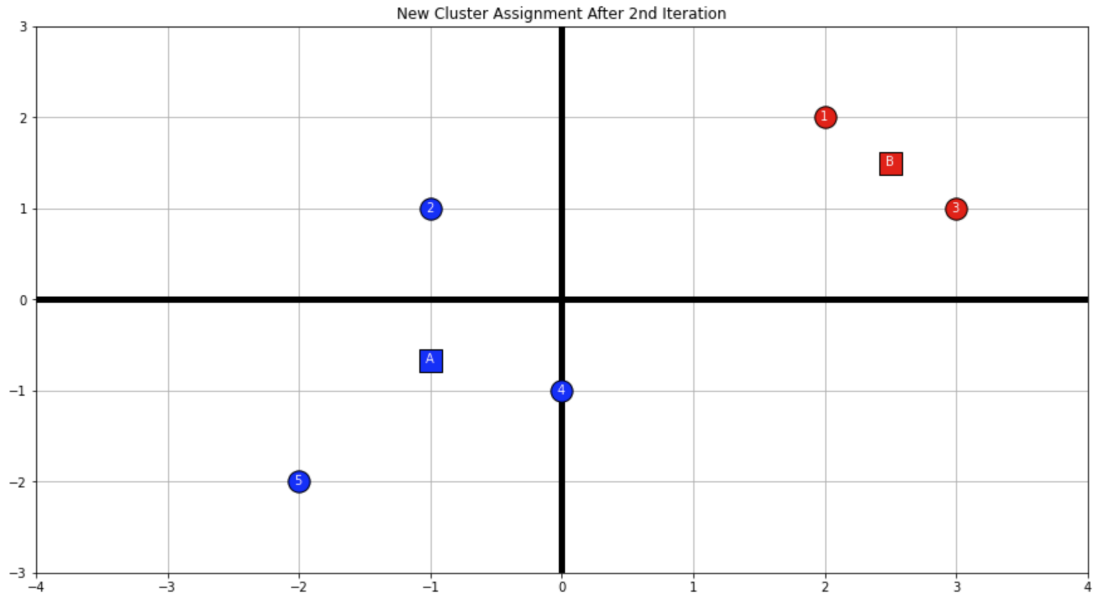


(b.) Show the location of the new center A New Center –> (-1.0, -1.5)

B New Center –> (1.33, 1.33)



(c.) Will it terminate in one step? Visually we can see that no, it won't terminate in one step. With the new updated centroid locations, point 2's manhattan distance is closer to centroid A than centroid B. However, let's not leave it to visual assumptions. Let's run another update. Before running the update, I manually checked the manhattan distance specifically for point 2 with respect to the two centroids. Point 2 to Centroid A is 2.5. Conversely point 2 to Centroid B is 2.66666



As we can see, point 2 was closer to Centroid A. Therefore it was reclassified and the position for both centroids was updated

Q2 Image Compression

In this programming assignment, you are going to apply clustering algorithms for image compression. Your task is implementing the clustering parts with two algorithms: K-means and K-medoids. It is required you implementing the algorithms yourself rather than calling from a package.

K-medoids

In class, we learned that the basic K-means works in Euclidean space for computing distance between data points as well as for updating centroids by arithmetic mean. Sometimes, however, the dataset may work better with other distance measures. It is sometimes even impossible to compute arithmetic mean if a feature is categorical, e.g, gender or nationality of a person. With K-medoids, you choose a representative data point for each cluster instead of computing their average. Please note that K-medoid is different from generalized K-means: Generalized K-means still computes centre of a cluster is not necessarily one of the input data points (it is a point that minimizes the overall distance to all points in a cluster in a chosen distance metric). Given m data points $x^i(i = 1, ..., m)$, K-medoids clustering algorithm groups them into K clusters by minimizing teh distortion function $J = \sum_{i=1}^{m} \sum_{j=1}^{k} r^{ij} D(x^i, \mu^j)$ where D(x,y) is a distance measure between two vectors x and y in same size (in case of K-means, D(x,y) $= ||x - y||^2$), μ^j is the center of j-th cluster and $r^{ij} = 1$ if x^n belongs to the k-th cluster and $r^{ij} = 0$ otherwise. In this exercise, we will use the following iterative procedure.

- Initialize the cluster center μ^j, j=1, …, k
- Iterate until convergence:
- Update the cluster assignments for every data point $x^i : r^{ij} = 1$ if j = $argmin_j D(x^i, ^j)$, and $r^{ij} = 0$ otherwise.
- Update the center for each cluster j: choosing another representative if necessary.

There can be many options to implement the procedure; for example, you can try many distance measures in addition to Euclidean distance, and also you can be creative for deciding a better representative of each cluster. We will not restrict these choices in this assignment. You are encouraged to try many distance measures as well as way of choosing representatives (e.g., $_1$ norm).

1.)

Within the k-medoids framework, you have several choices for detailed implementation. Explain how you designed and implemented details of your K-medoids algorithm, including (but not limited to) how you chose representatives of each cluster, what distance measures you tried and chose one, or when you stopped iteration. I created a separate classs for each of the 3 main features of the kmedoids algorithm

- **Distance** - This class houses the different implementations for the following distance metrics **{euclidean | euclidean squared | manhattan | minkowski}**. Since Euclidean and manhattan are just the minkowski distance where p = 2 and 1 respectively, the methods for these metrics internally call the minkowski method with the proper exponent
- **Initialization** - There's three different classes for intialization **{Random | Uniform | Bad}**. The random intialization class takes in k (the amount of clusters) and x (the points to be classified). It then randomly selects k points out of x to serve as the cluster centroids for k

clusters. The uniform intialization class takes in k (the amount of clusters) and x (the points to be classified). It then sorts the input x. After sorting x, it selects $\sum_i^k (\frac{i}{k}) * m$ index positions where m is the number of data points in x. e.g. If m = 20 and k = 3, uniform initialization will select the points of x at index locations (1/3)*20 , (2/3)*20 , (3/3)*20 = x[6], x[13], x[20]. The bad initializer first sorts the input data points. It then finds the midpoint index and returns data points at indices midpoint + i for i in range(k). E.g if x has 20 data points and k=3, x is first sorted, then datapoints x[10], x[11], x[12] are returned as the cluster centers

- **Kmedoids** - This class makes use of the distance and intialization classes to perform the kmedoids algorithm. First the algorithm initializes the clusters per the user defined initialization **(random | uniform | bad)**. It then assigns each point in X to a cluster. Next the algorithm updates the cluster centers by selecting 35% of the points in each cluster and takes turns swapping the subset of points as the cluster center. It then calculates the distance of that point from all the other subselected points by using the Distance class and using the user specified distance metric **{euclidean | euclidean squared | manhattan | minkowski}**. The point with the smallest distance is selected to be the new cluster center. The algorithm then repeats this process until convergence (new clusters = old clusters) or total iterations is achieved. In traditional Kmedoids, the subselected points in a cluster has its distance calculated from all other points in the cluster. In my implementation, the subselected points only have their distances calculated from other subselected points instead of every point in a cluster. This method greatly speeds up runtime and still produces great results. E.g. If cluster 1 has 1000 points, 350 points are randomly selected to be tried out as new cluster center. Each point of the 350 has its distance calculated only from the points of the subselected 350 points, not the full 1000 points in the cluster

2.)

Attach a picture of your own. We recommend size of 320 × 240 or smaller. Run your k-medoids implementation with the picture you chose, as well as two pictures provided (beach.bmp and football.bmp), with several different K. (e.g, small values like 2 or 3, large values like 16 or 32) What did you observe with different K? How long does it take to converge for each K? Please write in your report. For the 3 images utilized, the first thing to note is their overall size. After reshaping the pixels for each image to a dimension of (m,3), the football image had 255,440 data points. The beach image had 68,480 data points. And Finally, my cusomt image had 786,432 data points, making it signifcantly bigger than the other 3. Because of this, runtime for each varied.

- Football Image Runtime

K	Runtime (s)
2	271
32	44
256	36

- Beach Image Runtime

K	Runtime (s)
2	66
32	13
256	13

- Custom Image Runtime

K	Runtime (s)
2	2764
32	303
256	117

For each image, I tried out k=2, 32, 256 with manhattan distance. I utilized the uniform initialization to avoid the initialization of clusters very close to each other in order to facilitate faster convergence. By using this intialization and my implementation of cluster updates, I only needed 2 iterations for the football and custom image, and 5 iterations for the beach image. In fact, for all

instances of model execution, 2 iterations was sufficient to terminate execution. While letting the algorithms run longer, no significant difference was seen between an instance that ran for 2 iterations vs an instance that ran for 20 iterations. when k=2 for all 3 images, the pixels are grouped correctly into one of two clusters such that the image is still visually relevant. as k increased » 2, kmedoids does a very good job of replicating the original image. Runtime was the highest for all 3 images when k=2 due to the amount of points in each cluster and having to iterate through each one. When k » 2, there was far fewer points to iterate through during the update step since only 35% of the points were being sample. Overall the fastest k=22 was 66 seconds with the beach image since it had the fewest number of data points, 68,480. The longest k=2 runtime was my custom image at 40 minutes. This is because this is a full size 4k image that I didn't downsample. Because of this, there were 786,432 data points. This ordering of runtime (as seen above) holds true for both k=32 and k=256.

3.)

Run your k-medoids implementation with different initial centroids/representatives. Does it affect final result? Do you see same or different result for each trial with different initial assignments? (We usually randomize initial location of centroids in general. To answer this question, an intentional poor assignment may be useful.) Please write in your report. For this section, I ran the analysis on k=10. This is because k is large enough that runtime won't take a substantial amount of time. Also k is small enough that any variance in the intializations of clusters should reveal a difference in the final image. For the initializations, I used all 3 initialization methods, random, uniform, and bad. Remember that random initialization randomly chooses k points from the dataset, uniform first sorts the dataset then selects k, equally distant points, bad initialization first sorts the data, then finds the midpoint. It then returns all points from midpoint to midpoint + k as the intialized clusters

As we can see, cluster intialization does play a significant role in overall performance. All images using the bad intialization faired far worse in compression than the other two methods of random and uniform. It is clear that Uniform initialization is the best as it ensures the intial clusters are evenly spread out and not bunched into one region in the latent space. Random intialization didn't perform that bad either. However it is random, so results would vary with more runs of the algorithm.

4.)

Repeat question 2 and 3 with k-means. Do you see significant difference between K-medoids and k-means, in terms of output quality, robustness, or running time? Please write in your report.

- Football Image Runtime (k-means)

K	Runtime (s)
2	2764
32	303

K	Runtime (s)
256	117

- Beach Image Runtime (k-means)

K	Runtime (s)
2	3
32	2
256	4

- Custom Image Runtime (k-means)

K	Runtime (s)
2	27
32	30
256	49

KMeans clusterings produces images just as good as kmedoids, if not better. Not only this, but kmeans runs so much faster. After only 2 iterations, kmeans, at its fastest ran in 2.81 seconds for the beach image with 32 clusters. This is almost 7x faster than kmedoids for the same model parameters. Also, the longest run time was my custom image for k=256. This took approximately 50 seconds. This 55x faster than kmedoids slowest run time!! Kmeans is clearly the better choice for this type of process. Up next, we will do the intialization test for the kmeans algorithm

Again, we can see that intialization does play a role even with the kmeans algorithm. Uniform intialization produces the better image representation, with random as a close second. Bad intialization, as expected, produces bad results

---

Q3 Politcal Blogs Dataset

We will study a political blogs dataset first compiled for the paper Lada A. Adamic and Natalie Glance, "The political blogosphere and the 2004 US Election", in Proceedings of the WWW-2005 Workshop on the Weblogging Ecosystem (2005). The dataset nodes.txt contains a graph with n = 1490 vertices ("nodes") corresponding to political blogs. Each vertex has a 0-1 label (in the 3rd column) corresponding to the political orientation of that blog. We will consider this as the true label and try to reconstruct the true label from the graph using the spectral clustering on the graph. The dataset edges.txt contains edges between the vertices. You may remove isolated nodes (nodes that are not connected any other nodes).

1.)

Assume the number of clusters in the graph is k. Explain the meaning of k here intuitively

- k (with respect to eigenvectors/eigenvalues) - The laplacian of the adjacency matrix is generated and from this, eigenvalues and eigenvectors are calculated. A laplacian of k columns will have k eigenvectors and eigenvalues. If a laplacian has k eigenvalues such that $\lambda_1 = \lambda_2 = ... = \lambda_k = 0$, it is said that the network has **k - connected components**. Eigenvectors with $\lambda > 0$ express the level of "connectivity" of the connected nodes. The larger the eigenvalue, the stronger the connectivity. However, for better cluster separation, we want an eigenvalue that is as low as possible (weaker connectivity for easier discrimination between clusters).
- k (with respect to clusters) - If the graph is grouped into k clusters, it is interpreted (hopefully) that each cluster would be indicative of nodes that have high connection strength, thus share the same political orientation, or possibly share the same topic with regards to the blog itself

2.)

Use spectral clustering to find the k = 2, k = 3, and k = 4 clusters in the network of political blogs (each node is a blog, and their edges are defined in the file edges.txt). Then report the majority labels in each cluster, when k = 2, 3, 4, respectively. For example, if there are k = 2 clusters, and their labels are {0, 1, 1, 1} and {0, 0, 1} then the majority label for the first cluster is 1 and for the second cluster is 0. It is required you implementing the algorithms yourself rather than calling from a package. There are a total of 1490 nodes in the file. Of those nodes, only 1224 are connected. Spectral clustering was used with uniform intialization of nodes in the k-means algorithm. From this, the major label for each cluster with k=2,3,4 is as follows:

K	Cluster 1	Cluster 2	Cluster 3	Cluster 4
2	1	0	x	x
3	1	1	0	x
4	0	1	N/A	1

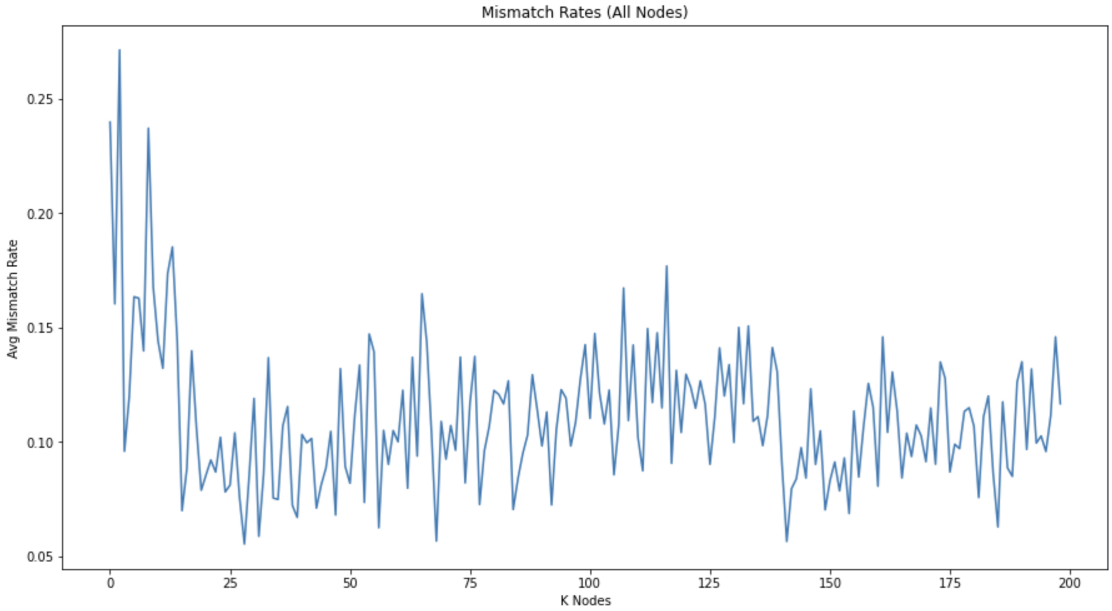
3.)

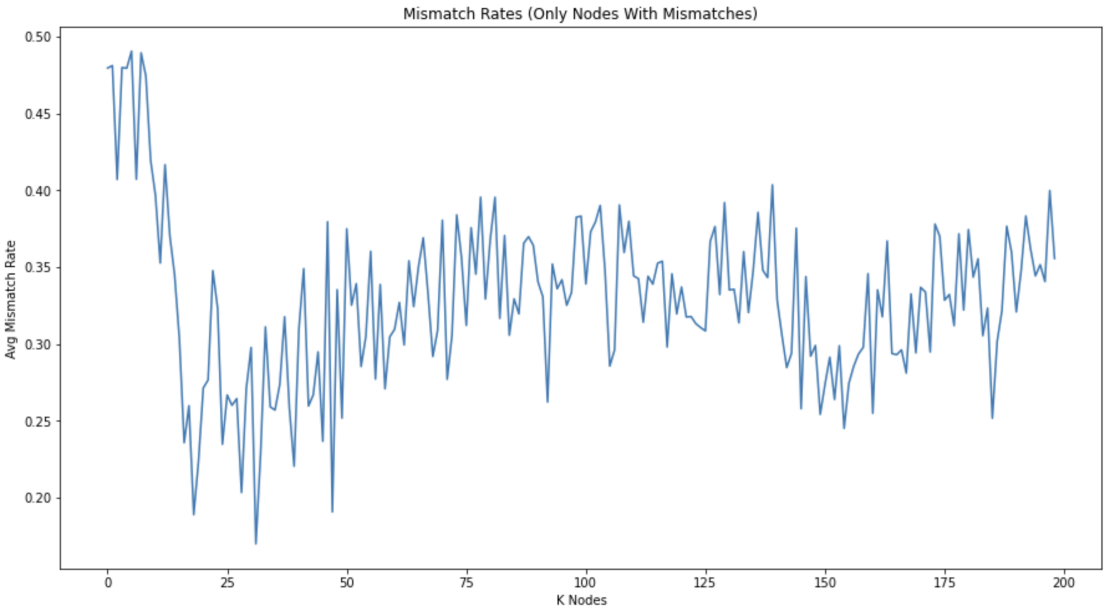
Now compare the majority label with the individual labels in each cluster, and report the mismatch rate for each cluster, when k = 2, 3, 4. For instance, in the example above, the mismatch rate for the first cluster is 1/4 (only the first node differs from the majority) and the the second cluster is 1/3. The mismatch rate can be seen below

K	Mismatch Rate
2	47.9 %
3	48.1 %
4	40.7 %

4.)

Now tune your k and find the number of clusters to achieve a reasonably small mismatch rate. Please explain how you tune k and what is the achieved mismatch rate. In analyzing the mismatch rate, there's two things we can take into account. We know there will be several clusters that will only have one or two points in them with no mismatch at all. These stringers can tend to bring down the average mismatch rate if there is a lot of them. But that doesn't adequately describe the state of things. So we will calculate 2 different mismatch rates. One that accounts for the mismatch rates of all nodes, and then mismatch rates that accounts only for nodes that actually have mismatches. We will do the latter by ignoring the nodes that have a mismatch rate of 0.





The model was tuned by simply iterating through various values of k. Also, it was seen that when a uniform initialization was utilized, the overall mismatch rates were lower than random initialization. We can also see that by accounting only nodes with mismatch rates in the averages, we get a little better picture of the magnitude of mismatches rates witnessed for each value of k where mismatches existed. When accounting for all nodes in the averages, the mismatch rate is drastically lower. This leads me to believe that with uniform initializaiton, there were several clusters of single values only which would produce 0 mismatch rate and bring down the overall mismatch averages. However, an average mismatch rate as low as 6% was achieved

5.)

Please explain the finding and what can you learn from this data analysis (e.g., node within same community tend to share the same political view, or now? Did you find blogs that share the same political view more tightly connected than otherwise?) To be completely honest, any analysis of this graph without further information isn't quite possible. A key piece of information is missing….How are these blogs connected? Sure we have edges.txt that physically says this node is connected to that one, but there's nothing explaining what the connection link is. Are two blogs connected because one cites the other? Is it only a one way citation where one cites the other but not vice versa? Are they connected because there is actually a hyperlink from one blog to the other? Without this information, true analysis of this graph would fall short. However, one could infer a few things from this type of analysis. Since there is a relatively high mismatch rate for almost any value of k, the connections between blogs could be the actual topics the blogs are writing about. Blog A with political affiliation "a" could be writing their blog with their views on gerrymandering. Blog B with affiliation "b" could also be writing their blog with their views on gerrymandering. Even though they're opposite views with a different take on the topic, the relationship between the two could be the topic itself.