ISYE-6740

Homework 1

1 Clustering [25 points]

Given m data points xi, i = 1,...,m, K-means clustering algorithm groups them into k clusters by minimizing the distortion function over {rij,µj}

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

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

1. Prove (using mathematical arguments that using the squared Euclidean distance as the dissimilarity function and minimizing the distortion function, we will have:

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

Proof
Step 1. Since we're trying to minimize the sum of errors with respect to the

$$\text{argmin} \left\| x^i - u^j \right\|^2 \text{ where } k = 1 \tag{1}$$

we know the centroid will only depend on the sum of the observations, therefore we can remove the sum related to 1 to K clusters. When converting the cluster objective function into quadratic form and take the derivative with respect to the cluster center u^j through chain rule, we can deduce:

$$J = \sum_{i=1}^{m} r^{ij} \left\| x^i - u^j \right\|^2 \tag{2}$$

$$-2 \sum_{i=1}^{m} r^{ij} \left(x^i - u^j \right) = 0 \tag{3}$$

Step 2. Solve for u^j by bringing over one of the terms to the left-side of the equation:

$$\sum_{i=1}^{m} r^{ij} x^i = \sum_{i=1}^{m} r^{ij} u^j \tag{4}$$

$$u^j = \frac{\sum_i r^{ij} x^i}{\sum_i r^{ij}} . \quad (5)$$

2. (5 points) Prove (using mathematical arguments) that K-means algorithm converges to a local optimum in finite steps.
Proof (Sentence form)
The k-means algorithm converges to a local minimum due to the problem being NP-hard. For each cluster, the algorithm will start by selecting a random permutation of observations to form the initial centroids. From this step, the objective function will assign points to each cluster based on the closest proximity to the nearest centroid.

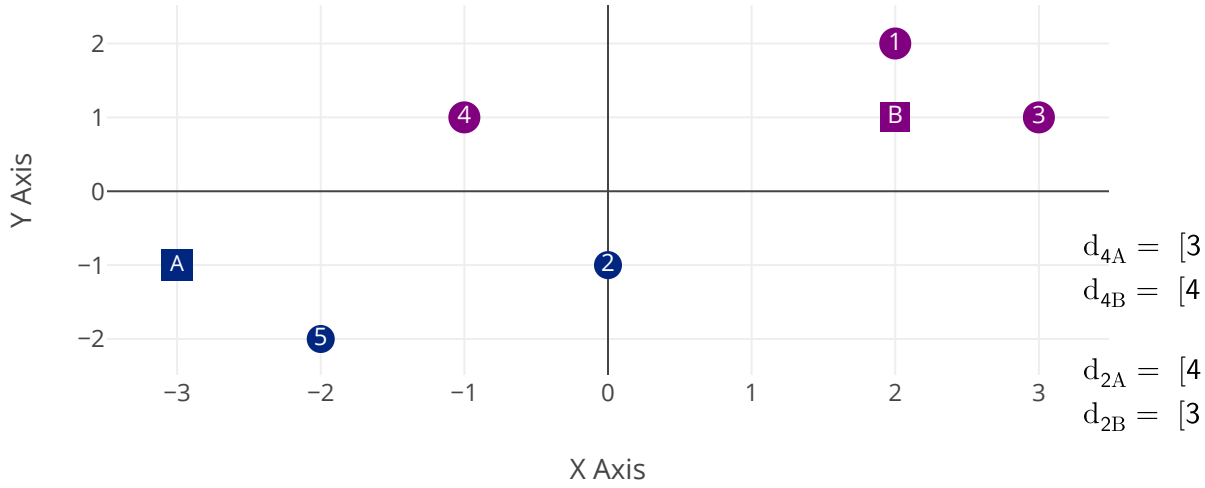
Once done with the assignment on the first iteration, it will re-asses the centroids by calculating the mean-value of the cluster in order to minimize the sum of error across all observations in the cluster.

The process will repeat these steps until it's met a local convergence, meanings that the existing mean value used for the centroid has minimized the error as much as it can for that particular cluster. The reason it converges to a local minimum rather than a global minimum is that there could possibly be an initial selection of centroids, where after several iterations could potentially reduce the total sum of errors compared to the previous run. Therefore, we're never truly finding the global minimum.

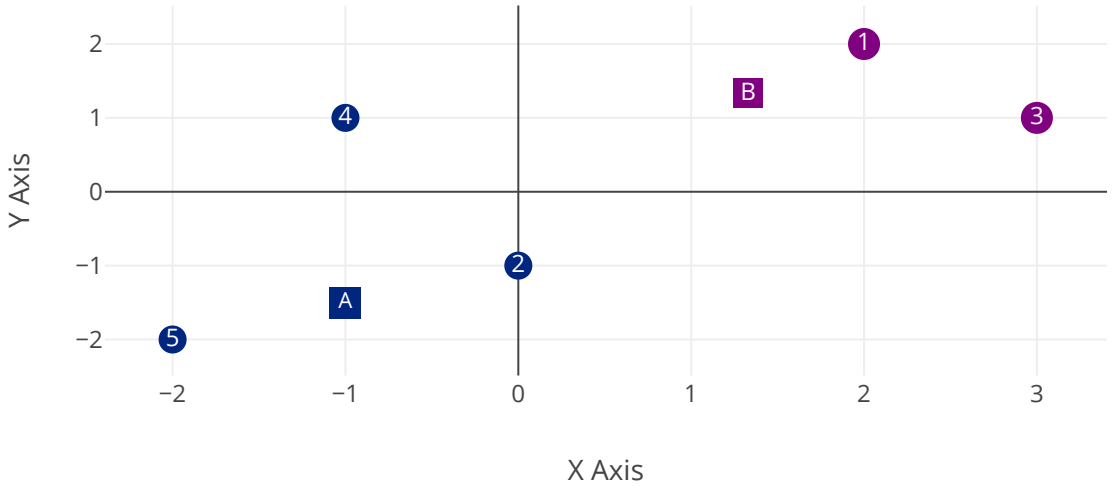
(1)

3. (10 points) 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 (x1,y1) and (x2,y2), their distance is |x1 −x2|+|y1 −y2|). Assuming the initialization of centroid as shown, after one iteration of k-means algorithm, answer the following questions.

(a) Show the cluster assignment;
Based on the below cluster assignment, we can determine that the Manhattan distance is smaller for the distance from point A to point 4 and B to point 2, respectively.



(b) Show the location of the new center;



Once the new centroids have been calculated, centroid A and B are now located at (-1,-1.5) and (1.33,1.33), respectively. We now see point 4 has been reassigned to cluster A due to it's closer Manhattan distance.

(c) Will it terminate in one step?
No, there's an additional step after this as centroid A needs to move into it's final place at point (-1,-1.33), since the new point for observation 4 will move it up. Once complete the k means algorithm will have converged.

2.

1. (5 points) 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.

For my k-medoids algorithm I used the following steps:

Step 1) Select initial medoids by randomly selecting certain observations from the matrix. I later tried to select observations which were at different quartiles of the graph in order to spread out the initial centroids so they weren't too close together.
Step 2) Using euclidean distance as the metric, I calculated the distance between each cluster observation to the medoids in order to determine which cluster the observation belonged to.
Step 3) Once the cluster assignments are complete, we needed to determine out of the observations in each cluster, which one minimized the total distance across all observations in the cluster. Since this was too intensive for performance, I ended up using a sample instead, then calculating the new medoids that way. Using this approach I saw much better performance with a minimal impact on convergence for lower values of k.
Step 4) Then I would continue to iterate through steps 2 and 3 until I either reached my convergence or reached my max number of iterations.

2-4 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.

Algorithm Name	Number of Clusters (K)	Number of Iterations Ran	Total Iterations	Image Name
kmedoids	3	9	100	compressed_kmedoids_3_iter100_football.png
kmeans	3	19	100	compressed_kmeans_3_iter100_football.png
kmedoids	3	10	100	compressed_kmedoids_3_iter100_beach.png
kmeans	3	10	100	compressed_kmeans_3_iter100_beach.png
kmedoids	16	27	100	compressed_kmedoids_16_iter100_football.png
kmeans	16	100	100	compressed_kmeans_16_iter100_football.png
kmedoids	16	100	100	compressed_kmedoids_16_iter100_beach.png
kmeans	16	46	100	compressed_kmeans_16_iter100_beach.png
kmedoids	32	100	100	compressed_kmedoids_32_iter100_football.png
kmeans	32	100	100	compressed_kmeans_32_iter100_football.png
kmedoids	32	100	100	compressed_kmedoids_32_iter100_beach.png
kmeans	32	100	100	compressed_kmeans_32_iter100_beach.png

You notice that the convergence time is significantly lower with smaller amount for k, which would make sense because by having such few clusters would mean that the observations have

less option to be reassigned once the algorithm get's closer to convergence, and the choice of medoid is a much clearer choice once the algorithm reaches the local minimum. However, for larger values of k it seems to not be able to converge at all, or just takes much longer when compared to runs with lower values of k.

Figure 1: k-means k=3

Figure 2: k-medoids k=3

Figure 3: k-means k=3

Figure 4: k-medoids k=3

Figure 5: k-means k=3

Figure 6: k-medoids k=3

Figure 7: k-means k=16

Figure 8: k-medoids k = 16

Figure 9: k-means k=16

Figure 10: k-medoids k=16

Figure 11: k-means k=32

Figure 12: k-medoids k = 32

Figure 13: k-means k=32

Figure 14: k-medoids k = 32

3. (5 points) 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.

I was starting to notice that selecting specific observations closer together, when compared to doing random assignment, we would see much different results. The rate of convergence when using random assignment was much faster, biggest I believe starting with a wider dispersion will allow the clusters to cover more observations in much fewer iterations, vs choosing initial

observations closer together would cause the clusters to fight over observations much faster then expected.

This is why it takes so long to converge for larger values of k, because the clusters are constantly fighting for observations since the medoids are much closer to each other with higher values of k.

4. (5 points) 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.

You can see noticeable differences between k-means and k-medoids when you look at observations with fewer values for k, you'll notice that k-medoids isn't able to pick up the cloud shapes in the image. However, when you look at the image quality for the larger values for k, you can see that it almost converts back into the original image, if not very close.

The beach photos are a good example to take note of. For higher values of k like 16 or 32, you can see that the k-medoids representation looks a little less blotchy compared to k-means, however k-means is about to account for saturation much better whenever you look at the color of the roofs between the two images.

For run-time, the k-medoids algorithm was significantly more intensive initially due to the re-calculation of the medoids. However, this problem was mitigated once I starting selecting a sample of observations in the cluster to determine the point with the minimum total distance.

3 Spectral clustering [25 points]

1. (10 points) For the following data (two moons), give one method that will successfully separate the two moons? Explain your rationale.

The Spectral clustering algorithm would be a good choice in this case as the data in the figure for this question is highly connected data. Based on the adjacency calculation inherit in the algorithm, then utilize the k-means algorithm on the eigenvector and eigenvalues to find the clusters of connected data in the image.

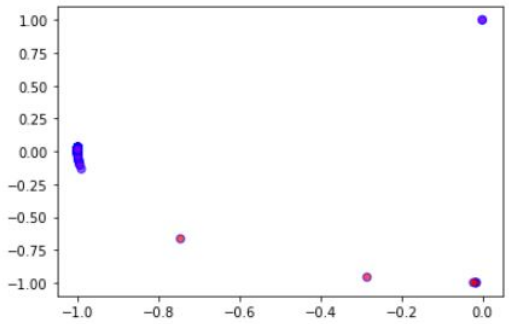
2. (15 points) Political blogs dataset. We will study a political blogs dataset first compiled for the paper Lada A. Adamic and Natalie Glance, “The political blogosphere and the 2004 US Election”, in Proceedings of the WWW-2005 Workshop on the Weblogging Ecosystem (2005). The dataset nodes.txt contains 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).

(a) (10 points) Assume the number of clusters to be estimated is k = 2. Using spectral clustering to find the 2 clusters. Compare the clustering results with the true labels. What is the false classification rate (the percentage of nodes that are classified incorrectly). It is required you implementing the algorithms yourself rather than calling from a package.

Provided is the results of our k-means where k=2:

```
Out[150]: <matplotlib.collections.PathCollection at 0x1de8f979dc8>
```



When you look at the results for the classification, you can see that the algorithm performs extremely poorly when compared to true results. There seems to be a relatively even split with the true labels, whereas our results are significantly skewed to a 0 label.

```
In [126]: unique, counts = np.unique(c_idx, return_counts=True)

          results = np.asarray((unique, counts)).T
          results

Out[126]: array([[   0, 1218],
                 [   1,    6]], dtype=int64)

In [149]: import pandas as pd
          nodes = pd.read_csv('./data/nodes.txt', sep='\t',header=None,names= ['Idx','Blog_Url','Label','Blog_Name']).reset_index(drop=True
          nodes.groupby('Label')['Label'].count()

Out[149]: Label
          0.0    756
          1.0    732
          Name: Label, dtype: int64
```

(b) (5 points) You might observe the performance is not as good as you expected (given that there is no coding bugs). What do you think might be the reason for the not-so-good performance, due to the discrepancy from “theory” and “application”? Please write in your

report.

Based on the use of spectral clustering here, this is the making the assumption that there is underlying communities determined by the clustering approach of kmeans have a strong correlation to their political interests, which is not likely the case and is an unsuitable application of spectral clusering for this situation.

4 PCA: Food consumption in European area [25 points]

The data food-consumption.csv contains 16 countries in the European area and their consumption for 20 food items, such as tea, jam, coffee, yoghurt, and others. There are some missing data entries: you may remove the rows “Sweden”, “Finland”, and “Spain”. The goal is to perform PCA analysis on the data, i.e., find a way to perform linear combinations of features across all 20 food-item consumptions, for each country. If we extract two principal components, that means we use two singular vectors that correspond to the largest singular values of the data matrix, in combining features.

1. (5 points) Write down the set-up of PCA for this setting. Explain how the data matrix is set-up in this case (e.g., each dimension of the matrix correspond to what.) Explain in words how PCA is performed in this setting.
In our case we're looking to calculate the covariance matrix for our observations, which is then used to determine our eigenvectors and eigenvalues.

Step 1) Find the covariance matrix in the form:

$$C = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu)(x_i - \mu)^T$$

Step 2) Find the eigenvectors such corresponding to the largest eigenvalues

$$w_i : \lambda_i C = Cw_i, i = 1,... d$$

Where λ_i are the corresponding eigenvalues of C.

Step 3) Project those back onto the original data to calculate the z matrix representation. When choosing the top k principal components, we're selecting the 1 to k eigenvectors with the largest eigenvalues.

$$z_i = \begin{bmatrix} w_1^T(x_i - \mu)\sqrt{\lambda_1} \\ w_2^T(x_i - \mu)\sqrt{\lambda_2} \\ \\ w_k^T(x_i - \mu)\sqrt{\lambda_k} \end{bmatrix}$$

2. (5 points) Suppose we aim to find top k principal components. Write down the mathematical optimization problem involved for solving this problem. Explain the procedure to find the top k principal components in performing PCA.
To find the first principal component the formula as mentioned in our lecture videos is the following:

$$w_1 = \begin{array}{l} \text{argmax } w^T C w \\ ||w||_2^2 = 1 \end{array}$$

$$w_k = \begin{array}{l} \text{argmax } w^T C - \sum_{i=1}^{k-1} w_i w_i^T) w \\ ||w||_2^2 = 1 \end{array}$$

Where k is the number of principal components chosen. You will iterate this algorithm until you've selected k eigenvectors corresponding to the first k principal components with the largest eigenvalues.

3. (7 points) Find the top two principal direction vectors (i.e., the eigenvectors of C) for the dataset and plot them (plot a value of the vector as a one-dimensional function). Describe do you see any pattern. You may either use a package or write your own code.

```
In [272]: plt.plot(pd_pca.columns,pca.components_[1])
          plt.xticks(rotation=90)

Out[272]: ([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20],
           <a list of 21 Text major ticklabel objects>)
```

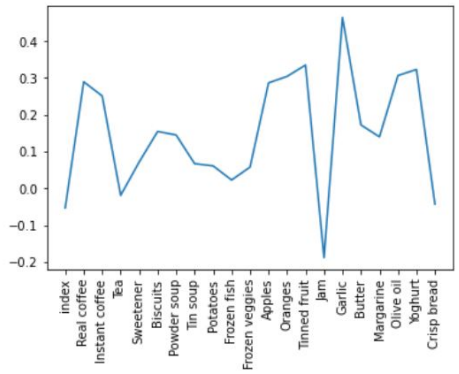


Figure 15: First Principal Component

```
In [271]: plt.plot(pd_pca.columns,pca.components_[0])
          plt.xticks(rotation=90)

Out[271]: ([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20],
           <a list of 21 Text major ticklabel objects>)
```

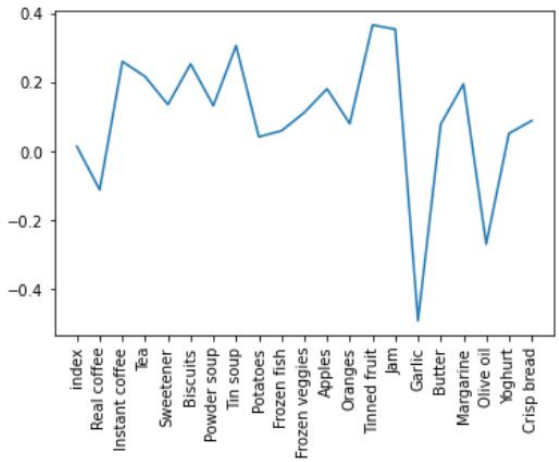


Figure 16: Second Principal Component

When we take a look at the plots of the first two principal components, we can see that the first principal component has largely positive values and seems to fall closely to the average of the data, whereas whenever we look at the 2nd principal component we can see that it has an extensive amount of negative values and it's capturing mostly the variance of the observed values.

4. (8 points) Now project each data point using the top two principal component vectors

(thus now each data point will be represented using a two-dimensional vector). Draw a scatter plot of two-dimensional reduced representation for each country. What pattern can you observe? You may use use a package or write your own code.

By utilizing sklearn's inverse_transform function, we can project our two principal vectors back onto the original data in order to retrieve the data in it's original form. We can see that the new values predominantly fit the data well, but the defining features that seem to separate certain countries from the main group is the consumption of Garlic and Olive Oil.

In [267]:
```
germany_transformed = np.array(X_concat_df.loc[X_concat_df['Country'] == 'Germany'].iloc[:, 1:])
plt.plot(X_concat_df.columns[1:],germany_transformed[0])
plt.plot(X_concat_df.columns[1:],germany_transformed[1])
plt.xticks(rotation=90)
```

Out[267]:
```
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
 <a list of 20 Text major ticklabel objects>)
```