THE UNIVERSITY OF TEXAS AT AUSTIN
Department of Statistics and Data Sciences
College of Natural Sciences

Stat models for big data

Clustering

Purnamrita Sarkar
Department of Statistics and Data Science
The University of Texas at Austin

https://psarkar.github.io/teaching

Clustering

- Given n data points, we want to divide them into K groups such that
  - datapoints in a group are very similar to each other
  - datapoints in two different groups are less similar

Clustering

- Given n data points, we want to divide them into K groups such that
    - datapoints in a group are very similar to each other
    - datapoints in two different groups are less similar
- But how to define similarity?

Clustering

- Given n data points, we want to divide them into K groups such that
  - datapoints in a group are very similar to each other
  - datapoints in two different groups are less similar
- But how to define similarity?
- Let us start with something very simple. We will use the Euclidean distance as dis-similarity) between two points.

- The k-means loss is given by:

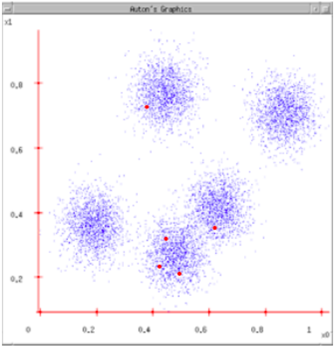
$$\sum_k \sum_{i \in C_k} \|x_i - \mu_k\|^2$$

The k^{th} cluster

k^{th} cluster center

The algorithm

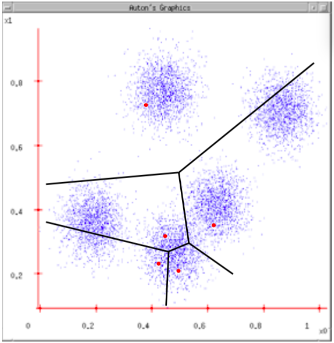
- Initialization: randomly guess k cluster center.



Courtesy: Andrew W. Moore's k-means slides at CMU

The algorithm

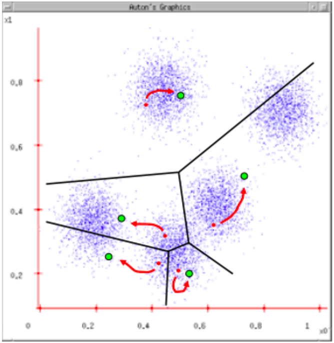
- Initialization: randomly guess k cluster center.
- In each iteration, each point finds out which cluster it is closest to.



Courtesy: Andrew W. Moore's k-means slides at CMU

The algorithm

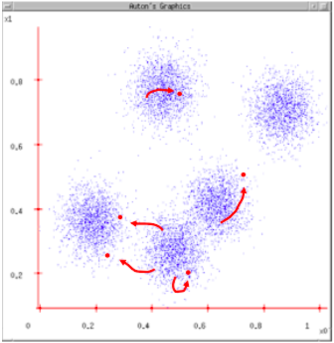
- Initialization: randomly guess k cluster center.
- In each iteration, each point finds out which cluster it is closest to.
- Each cluster computes the centroid of points belonging to it



Courtesy: Andrew W. Moore's k-means slides at CMU

The algorithm

- Initialization: randomly guess k cluster center.
- In each iteration, each point finds out which cluster it is closest to.
- Each cluster computes the centroid of points belonging to it
- Make those the new centroids and continue.



Courtesy: Andrew W. Moore's k-means slides at CMU

A video

`https://miro.medium.com/max/335/1*JUm9BrH21dEiGpHg76AImw.gif`

Courtest: Sunny K. Tuladhar

So all is well?

- The k-means algorithm will converge to a local optima, not necessarily a global one.
- A bad initialization may lead to a poor local optima.

So all is well?

- The k-means algorithm will converge to a local optima, not necessarily a global one.
- A bad initialization may lead to a poor local optima.

Figure 1: Courtesy: https://www.geeksforgeeks.org/ml-k-means-algorithm/

What if I give you funny looking clusters

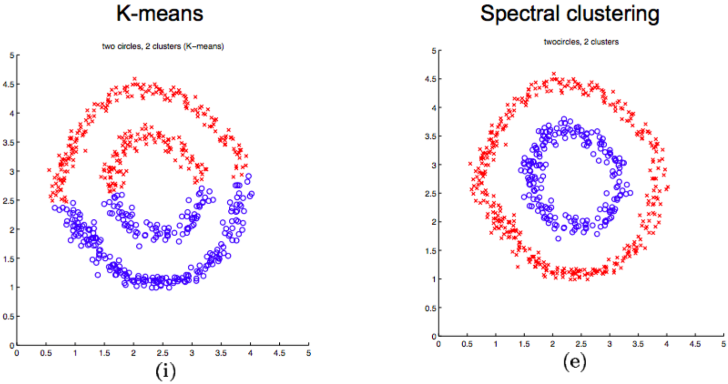


Figure 2: Shi and Malik ‘00; Ng, Jordan, and Weiss NIPS ‘01]

Spectral Clustering

The graph partitioning view

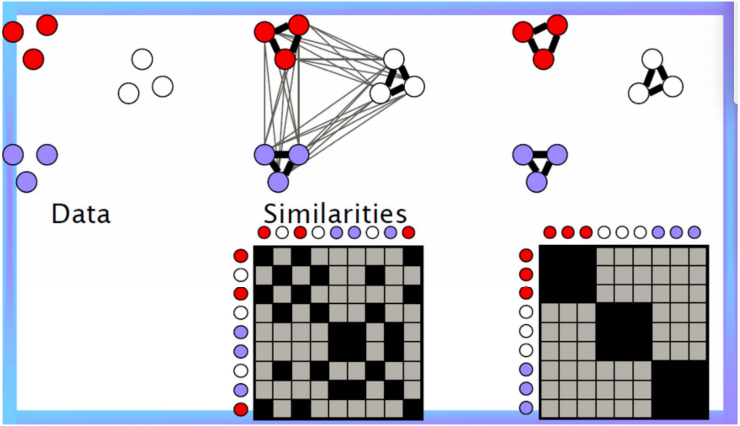


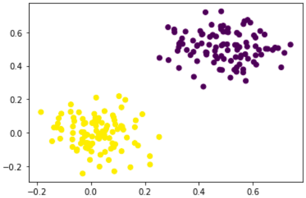
Figure 3: David Sontag's class notes

The Spectral Clustering algorithm in Ng, Jordan and Weiss

- Given dataset $x_1, \ldots x_n$ first form the affinity or similarity matrix $A \in \mathbb{R}^{n \times n}$ such that $A_{ij} = \exp(-\|x_i - x_j\|^2 / 2\sigma^2)$, and set the diagonal to zero.
- Let D be a diagonal matrix whose $D_{ii} = \sum_j A_{ij}$. Construct $L = D^{-1/2} A D^{-1/2}$
- Find the top K eigenvectors of L $v_1, \ldots v_K$ and build the matrix Y with these along the columns.
- Normalize the rows of Y to have length 1.
- Treating rows of Y as data points in K dimensions, run k-means with K clusters.

Try it out

- There are K Gaussians
- For each datapoint, you first decide which Gaussian it comes from by drawing from a multinomial with parameters $(\pi_1, \ldots, \pi_K)$
- If datapoint i comes from center i, then generate it from $N(\mu_i, \Sigma_i)$
- Goal: Given the data, figure out which gaussians/clusters/component it came from, and their parameters.



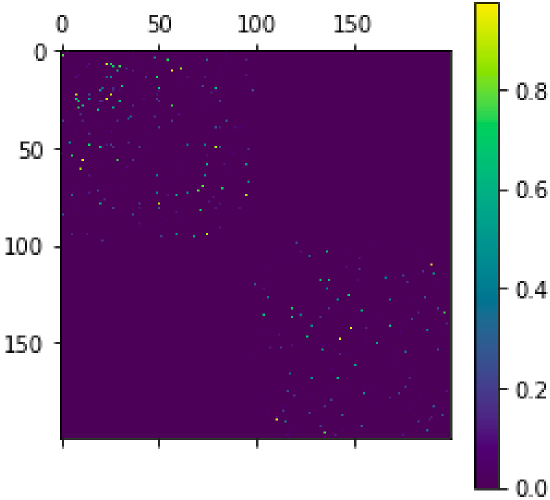
- $K = 2$
- $\pi = (.5, .5)$
- Spherical Gaussians

Try it out

```
from scipy.spatial.distance import pdist, squareform
from sklearn import metrics
D=metrics.pairwise_distances(X,X)
s=.01
K=np.exp(-D**2/(2*s**2))
K=K-np.diag(np.diag(K))
D1=np.diag(sum(K,axis=0)**(-.5))
K1=D1.dot(K).dot(D1)
```

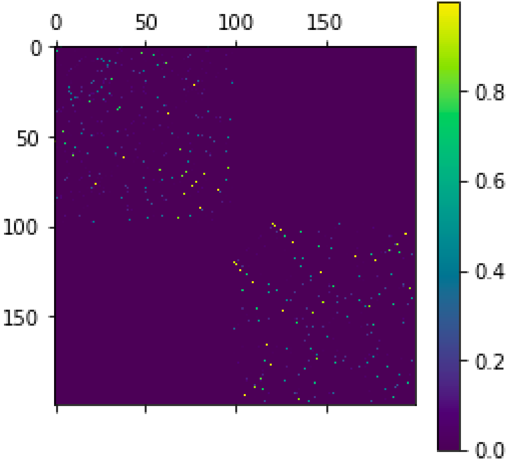
- Build the Kernel matrix and normalize it.

What does it look like with $s = .01$



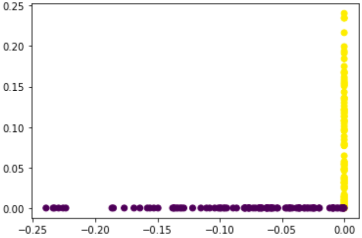
- What do you see?

What does it look like with $s = .05$



- What do you see?

Look at the eigenvectors

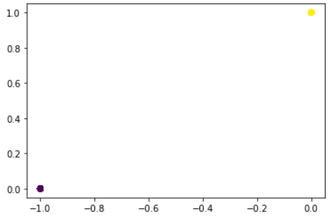


- This is the representation using the eigenvectors without normalization

Look at the eigenvectors-post normalization

```
plt.scatter(u1[:,0], u1[:,1],c=y)
```

```
<matplotlib.collections.PathCollection at 0x18ab63bb0>
```



- This is the representation using the eigenvectors with normalization
- Perfect accuracy

Wait, how do I measure accuracy or error

- Say you are given y_1 and y_2, two cluster assignments.
- You could do $\sum_i 1(y_1(i) \neq y_2(i))$
- Can you think for two minutes what is wrong with this?

Wait, how do I measure accuracy or error

- Say you are given y_1 and y_2, two cluster assignments.
- You could do $\sum_i 1(y_1(i) \neq y_2(i))$
- Can you think for two minutes what is wrong with this?
- Well, clusterings are not identifiable up-to permutation.

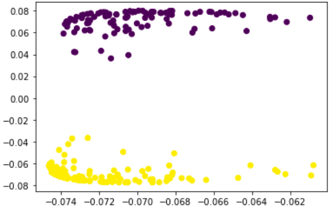
Wait, how do I measure accuracy or error

- Say you are given y_1 and y_2, two cluster assignments.
- You could do $\sum_i 1(y_1(i) \neq y_2(i))$
- Can you think for two minutes what is wrong with this?
- Well, clusterings are not identifiable up-to permutation.
- So the correct thing to do will be

$$\min_{\pi \in \Pi_K} 1(\pi(y_1(i)) \neq y_2(i))$$

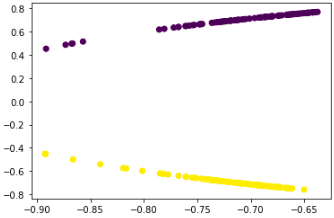
- Really costly when K is large, since we need to evaluate $K!$ permutations.
- You can use the Hungarian algorithm for mapping labels.

Look at the eigenvectors



- This is the representation using the eigenvectors without normalization

Look at the eigenvectors-post normalization

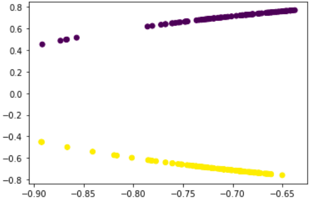


- This is the representation using the eigenvectors with normalization

Do k-means

```
kmeans = cluster.KMeans(2).fit(ul)
plt.scatter(ul[:,0], ul[:,1],c=kmeans.labels_)
```

```
<matplotlib.collections.PathCollection at 0x19134f640>
```



- This is the result labeled with k-means with 2 clusters
- Perfect accuracy, so pretty robust with selection of s

A tiny bit of theoretical insight

$$\hat{A} = \begin{bmatrix} A^{(11)} & 0 & 0 \\ 0 & A^{(22)} & 0 \\ 0 & 0 & A^{(33)} \end{bmatrix}; \quad \hat{L} = \begin{bmatrix} \hat{L}^{(11)} & 0 & 0 \\ 0 & \hat{L}^{(22)} & 0 \\ 0 & 0 & \hat{L}^{(33)} \end{bmatrix}$$

Figure 4: From Ng, Jordan, and Weiss 2001

- Consider an idealized setting, where there are no connections between the three clusters, and all elements of diagonal blocks are nonzero.
- Here after you normalize, for each of the smaller blocks there is exactly one eigenvalue 1, and all eigenvalues are strictly smaller.
- Call this $\hat{x}_1^{(i)}$

A tiny bit of theoretical insight

$$\hat{X} = \begin{bmatrix} x_1^{(1)} & \vec{0} & \vec{0} \\ \vec{0} & x_1^{(2)} & \vec{0} \\ \vec{0} & \vec{0} & x_1^{(3)} \end{bmatrix} \in \mathbb{R}^{n \times 3}.$$

Figure 5: From Ng, Jordan, and Weiss 2001

- Here padding the eigenvectors with zeros and then stacking them gives us eigenvectors of the bigger matrix.
- We need to be a bit careful, because all these eigenvalues are 1, so any 3 orthogonal vectors spanning this same subspace as the columns of the above will be eigenvectors too.

A tiny bit of theoretical insight

$$\begin{bmatrix} \vec{1} & \vec{0} & \vec{0} \\ \vec{0} & \vec{1} & \vec{0} \\ \vec{0} & \vec{0} & \vec{1} \end{bmatrix} R$$

Figure 6: From Ng, Jordan, and Weiss 2001

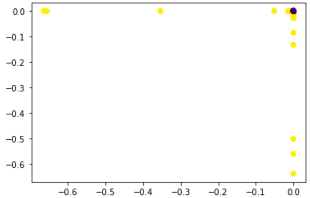
- R is a 3×3 orthogonal matrix, $R^T R = RR^T = I$
- When you normalize, all the points in cluster i get mapped to the i^{th} row of the orthogonal matrix R
- Typically, there will be some noise in the diagonal elements, and hence, *suitable conditions* we will have "tight" clusters around k well-separated points on the surface of the k sphere.

Why do the first normalization

```
u,s,vt=svd(K)
```

```
plt.scatter(u[:,0], u[:,1],c=y)
```

```
<matplotlib.collections.PathCollection at 0x18f055fd0>
```



- The very large entries in K take over and the top eigenvectors do not have any information about the cluster structure.