Machine Learning and Data Mining

Lecture 9.2: Unsupervised Learning II – Clustering (cont.) and Dimensionality Reduction Techniques

CS 434

RECAP

From Last Lecture

Moving to Unsupervised Learning

Supervised Learning

So far, our data has looked like:

$$\left[\begin{array}{c|c} x_{11}, x_{12}, \dots, x_{1d} & y_1 \\ x_{21}, x_{22}, \dots, x_{2d} & y_2 \\ \vdots & \vdots \\ x_{n1}, x_{n2}, \dots, x_{nd} & y_d \end{array}\right]$$

Features | Label

Goal: Prediction

Unsupervised Learning

Now, we'll consider **unlabeled data**

$$\left[\begin{array}{c} x_{11}, x_{12}, \dots, x_{1d} \\ x_{21}, x_{22}, \dots, x_{2d} \\ \vdots \\ x_{n1}, x_{n2}, \dots, x_{nd} \end{array}\right]$$

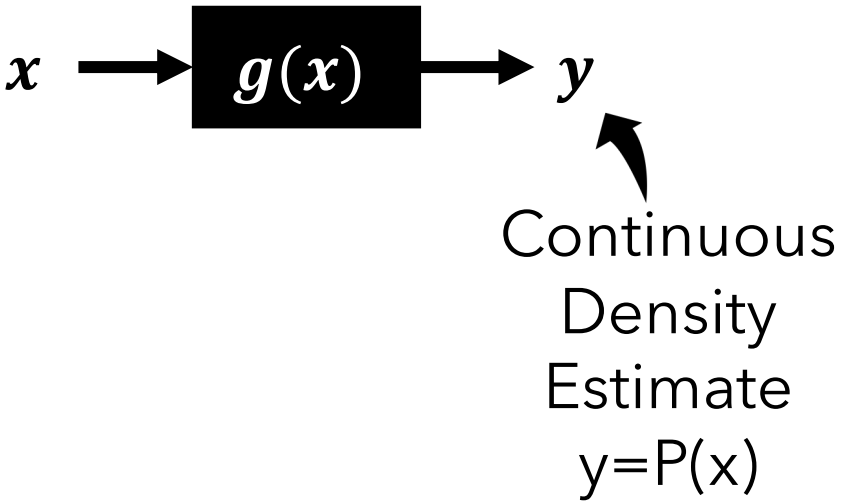
Features

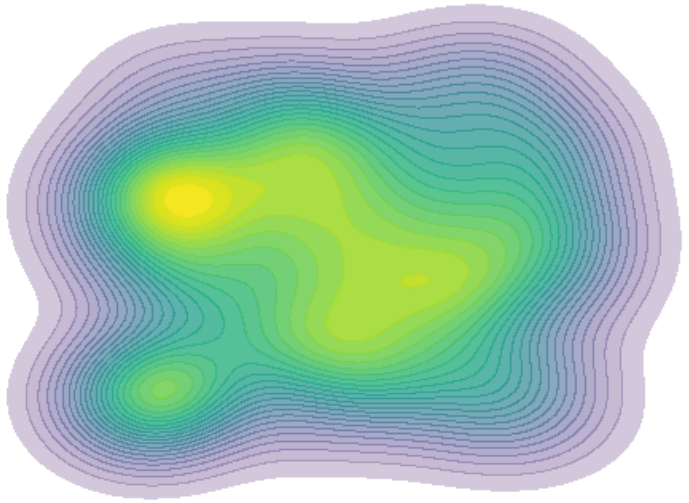
Goal: Discovery

Now, We Are Turning Towards Unsupervised Learning

Unsupervised Learning
Only given a set of input instances (x)

Density Estimation: Estimate the underlying distribution generating our data



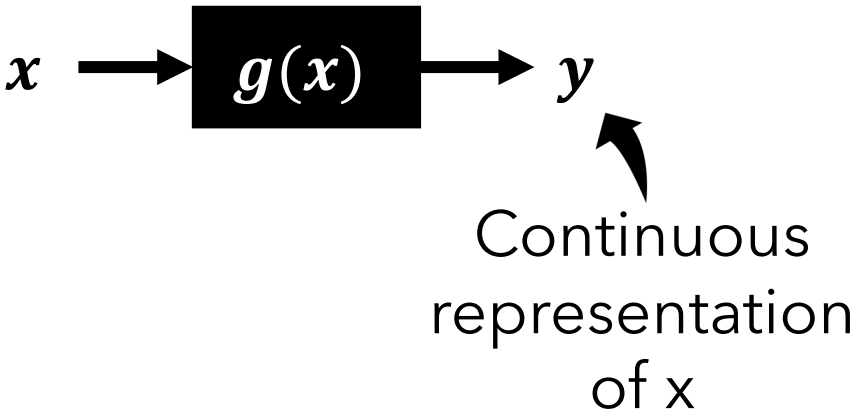


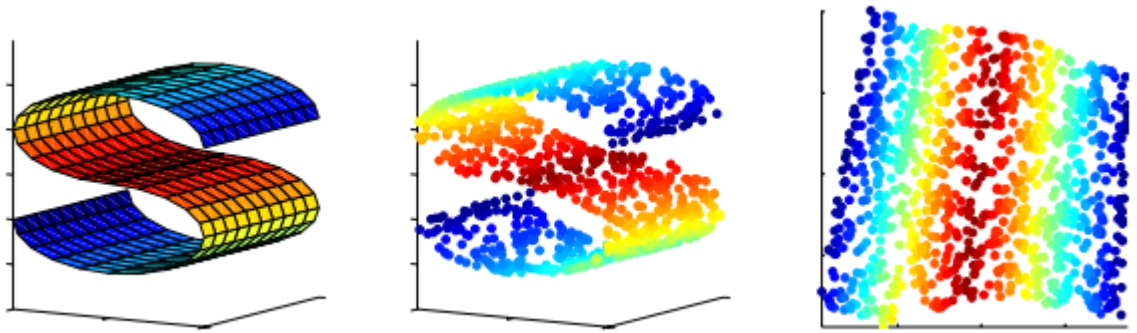
Example: Estimate how likely is a given house with these properties

Now, We Are Turning Towards Unsupervised Learning

Unsupervised Learning
Only given a set of input instances (x)

Dimensionality Reduction: Represent high-dim data as low-dim data



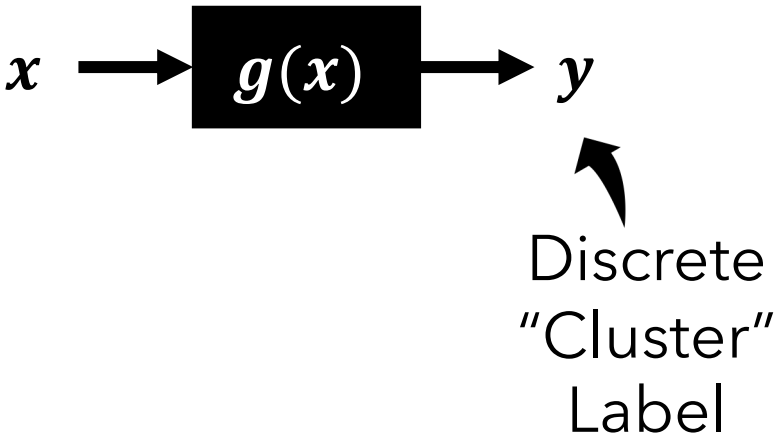


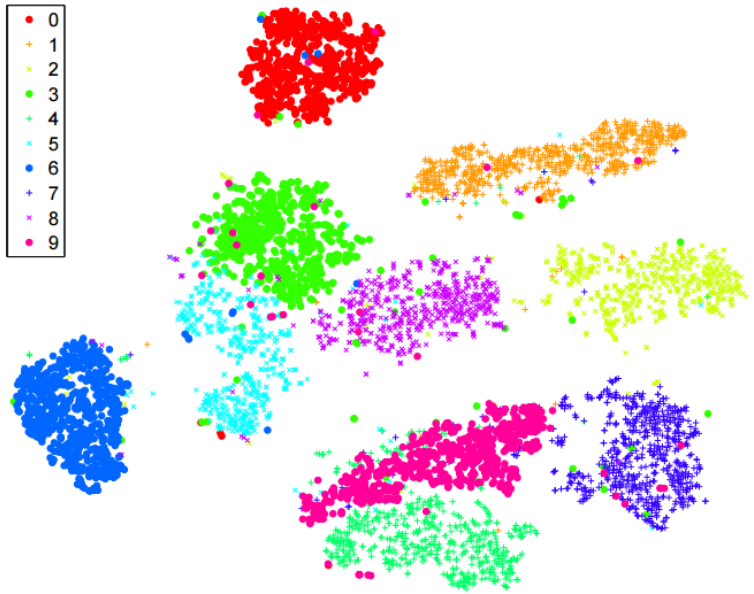
Example: Finding a 2D subspace in a 3D feature space

Now, We Are Turning Towards Unsupervised Learning

Unsupervised Learning
Only given a set of input instances (x)

Clustering: Discover self-similar groups within the data



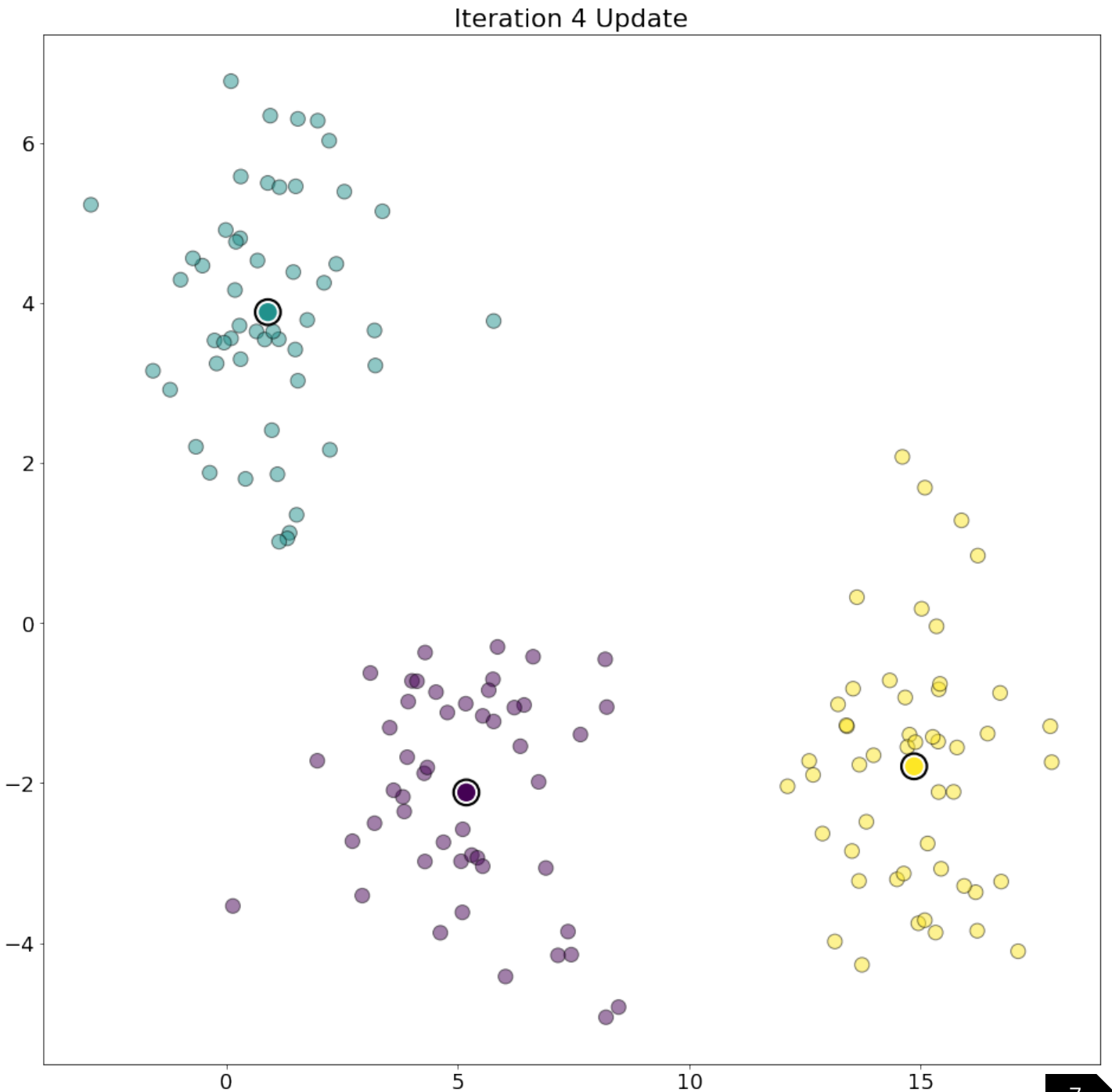


Example: Find groups of houses with similar properties

K-Mean Clustering

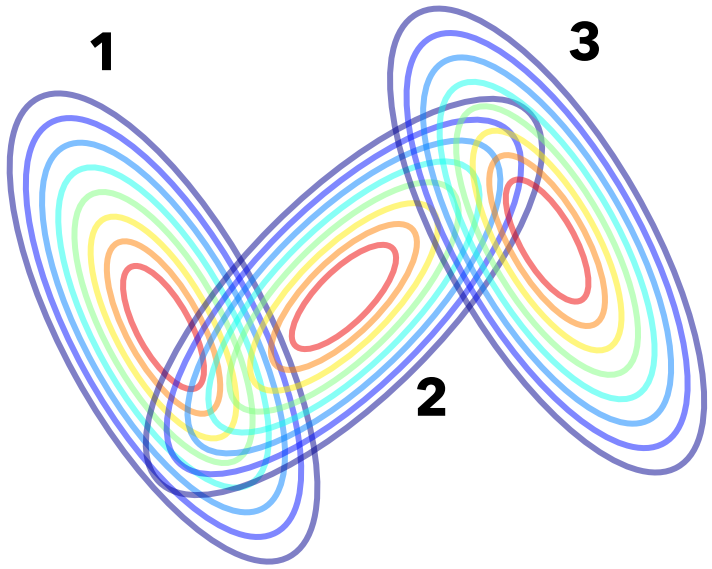
Algorithm:

1. Initialize *k* centroids randomly
2. While not converged:
    a) Associate each point with its nearest centroid
    b) Update each centroid as the average of associated points
3. Return centroids and associations

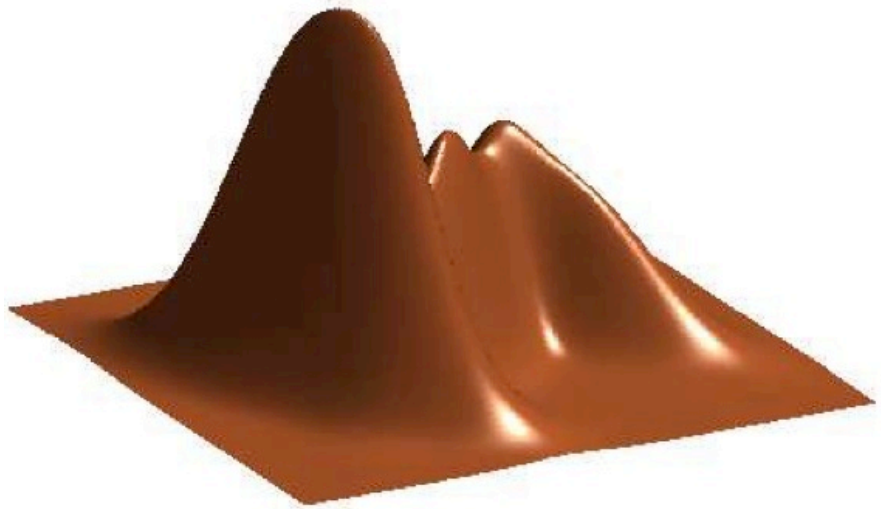


An Example Gaussian Mixture Model

$$P(\boldsymbol{x}) = 0.5 * \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu_1}, \Sigma_1) + 0.25 * \mathcal{N}(\mathbf{x}; \boldsymbol{\mu_2}, \Sigma_2) + 0.25 * \mathcal{N}(\mathbf{x}; \boldsymbol{\mu_3}, \Sigma_3)$$



$\boldsymbol{P(x)}$



$\boldsymbol{P(x)}$

Rightmost figure adapted from Kevin Murphy

The General Case – EM for GMM

Note: We didn't derive this. It comes from maximizing a tight lower-bound of the log-likelihood. See the textbook if you are curious.

The Expectation Maximization (EM) Algorithm for GMM

Initialize: probabilities of being in each Gaussian $\theta_1, \dots, \theta_k$ all to 1/k
means of the Gaussians $\mu_1, \dots, \mu_k$ to random points
covariances of the Gaussians $\Sigma_1, \dots, \Sigma_k$ to identity matrices

E-Step: Compute fractional assignment of point $\boldsymbol{i}$ coming from class $\boldsymbol{c}$

$$P(z_i = c \mid \boldsymbol{x}_i) \propto P(z_i = c)\mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_c, \Sigma_c) \quad \text{denote this} \quad p_{c|x_i}$$

Normalize these probabilities such that $\sum_c p_{c|x_i} = 1$

M-Step: Update the parameters based on the current fractional assignments

$$\theta_c^* = \frac{\sum_i p_{c|x_i}}{n}$$

Fraction of mass assigned to c

$$\boldsymbol{\mu}_c^* = \frac{1}{\sum_i p_{c|x_i}} \sum_i p_{c|x_i} \boldsymbol{x}_i$$

Weighted mean of fractional points assigned to c

$$\Sigma_c^* = \frac{1}{\sum_i p_{c|x_i}} \sum_i p_{c|x_i} (\boldsymbol{x}_i - \boldsymbol{\mu}_c)(\boldsymbol{x}_i - \boldsymbol{\mu}_c)^T$$

Weighted covariance of fractional points assigned to c

An Example GMM

Iteration 20:

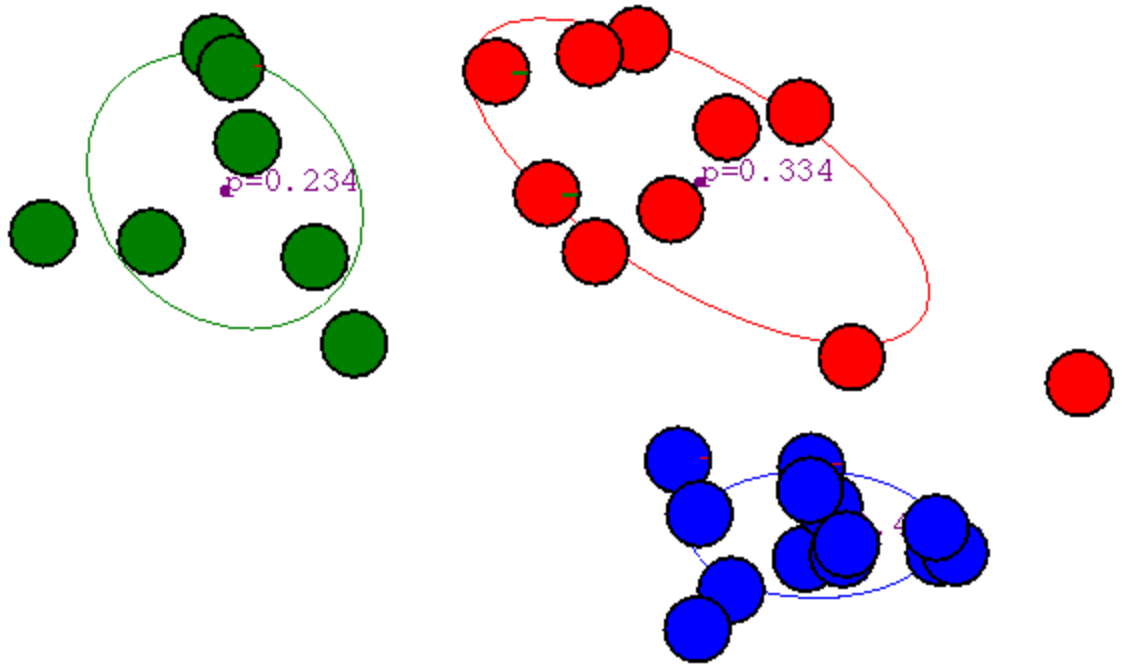
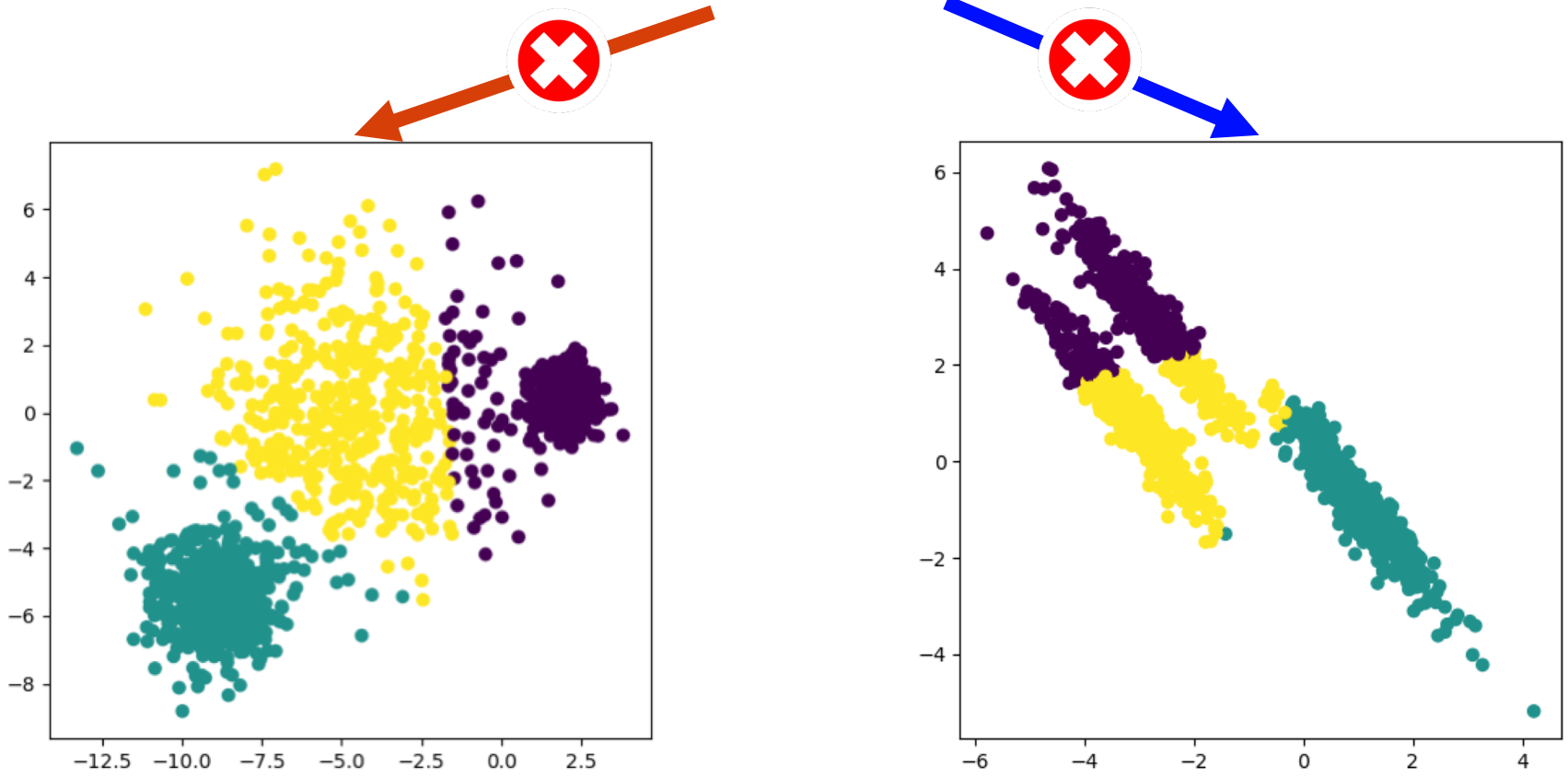


Figure credit Carlos Guestrin

What can Gaussian Mixture Models Tell Us About k-Means?

How to get k-Means back from GMMs:

- Assume a uniform distribution for P(z)
- Assume hard-assignment rather than fractional.
- Assume all Gaussians have the **same** **isotropic** covariance.



Figures adapted from Phil Roth

Today's Learning Objectives

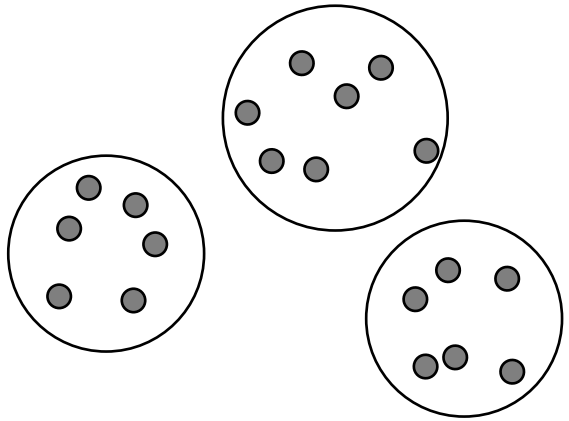
Be able to answer:

- What is Hierarchical Agglomerative Clustering (HAC)?
  - What are different linking strategies?
- How can we evaluate clustering?
- What is dimensionality reduction?
  - What is principal component analysis (PCA) and how does it work?
  - What are stochastic neighbor embeddings (SNE) and how do they work?
- What is kernel density estimation (KDE)?
  - How does a Gaussian KDE work?

Two Categories of Clustering Algorithms

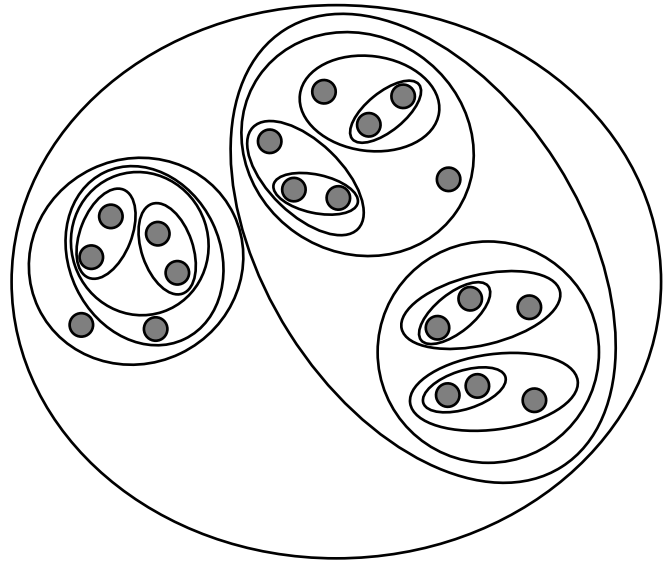
Partition Algorithms ("Flat" Clusterings)

- k-Means / k-Medians
- Gaussian Mixture Models



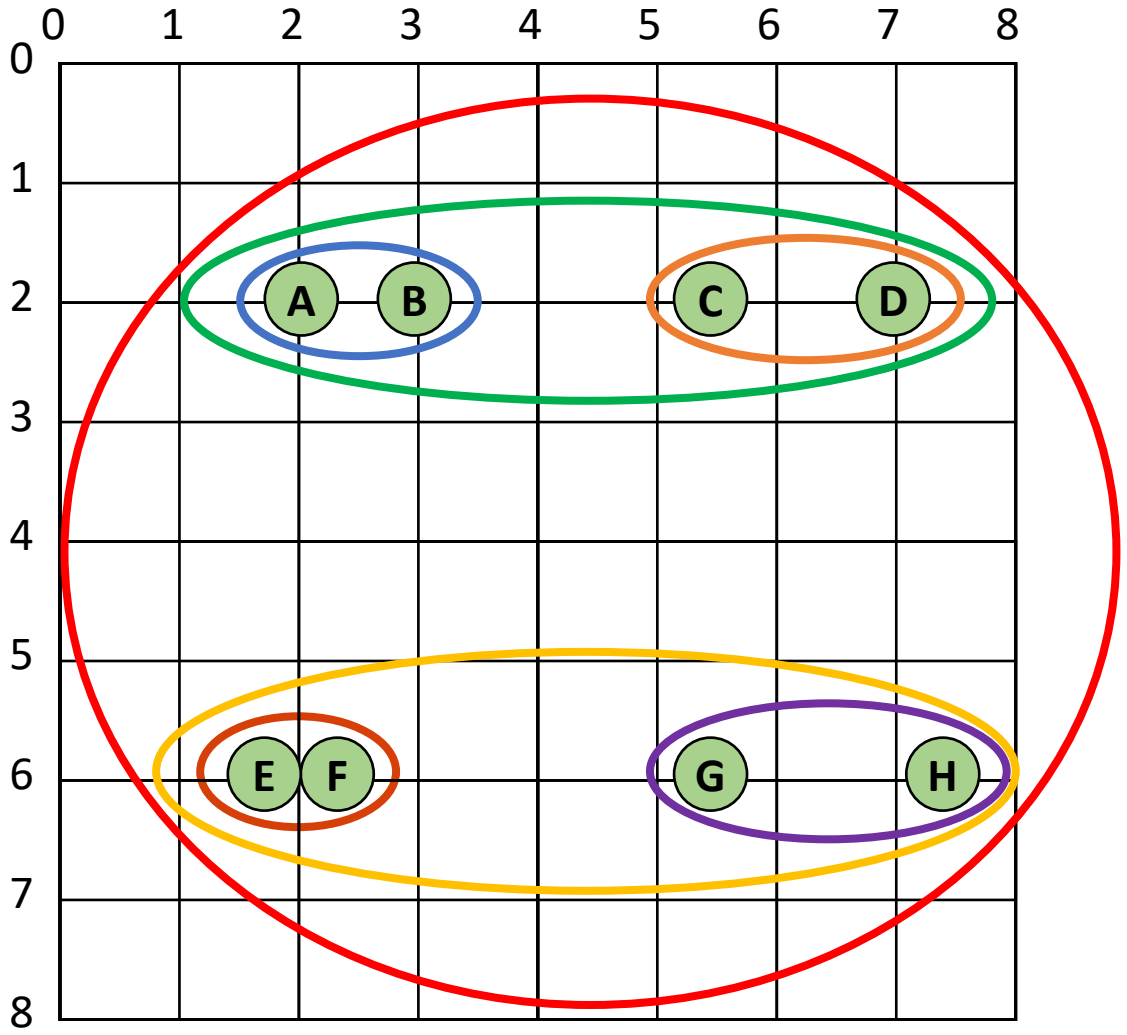
Hierarchical Algorithms

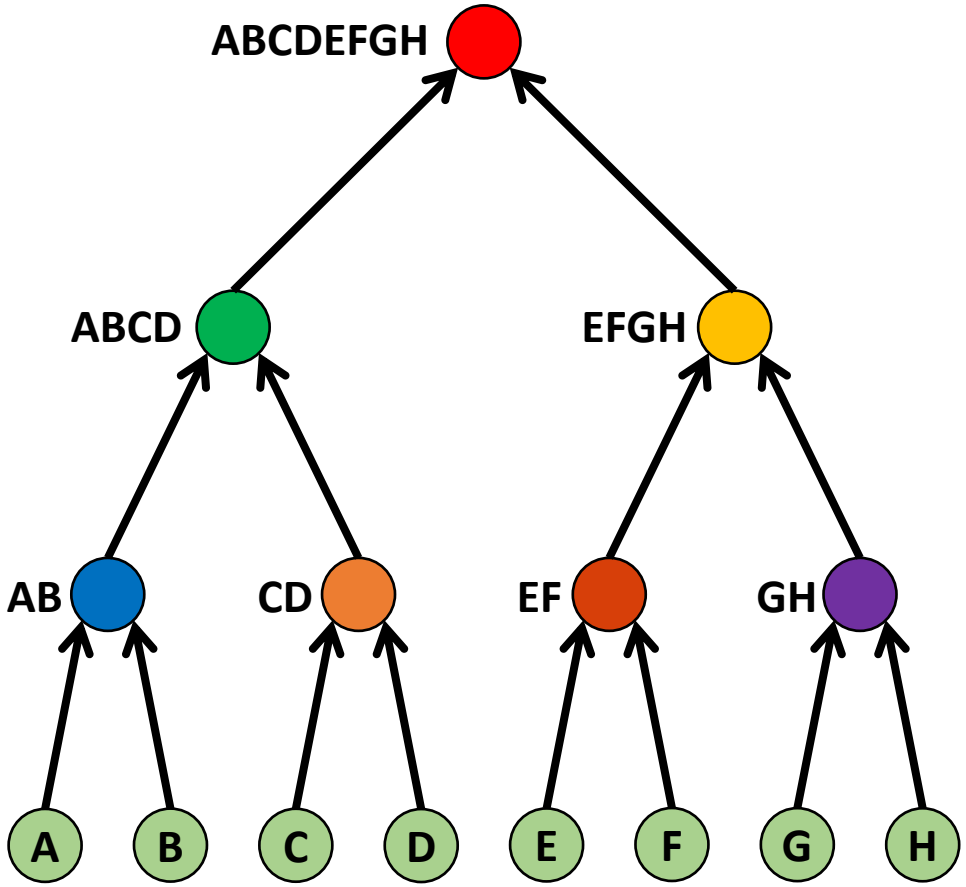
- Bottom-up – Agglomerative
- Top-down – Divisive



Slide adapted from Xiaoli Fern

Hierarchical Agglomerative Clustering (HAC)

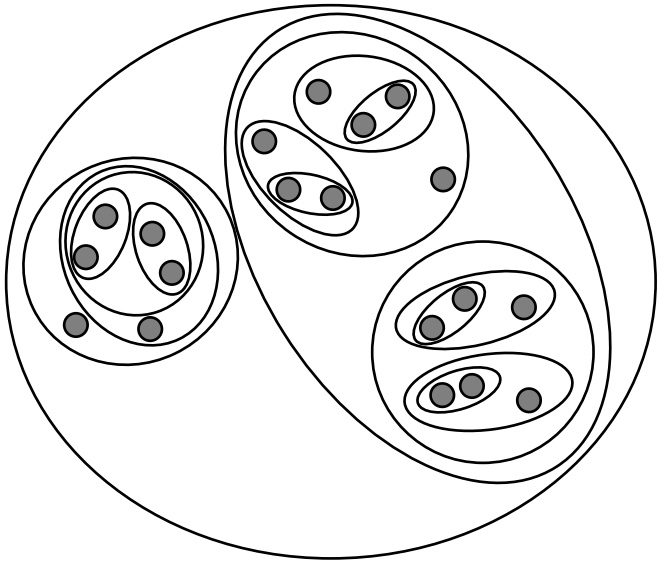




Hierarchical Agglomerative Clustering (HAC)

Given a dataset $X = \{\boldsymbol{x}_i\}_{i=1}^{n}$ where $\boldsymbol{x}_i \in \mathbb{R}^d$ and some distance function $d(x_i, x_j)$ which measures the distance between two points:

1. Initialize every datapoint as its own cluster
2. Until there is only one cluster remaining:
    a) Merge the two closest clusters



Notice it doesn't output a specific number of clusters. Can save intermediate clusterings from k=n to k=1.

Question: We have a measure of distance between points, how do we use it to measure "closeness" of clusters?

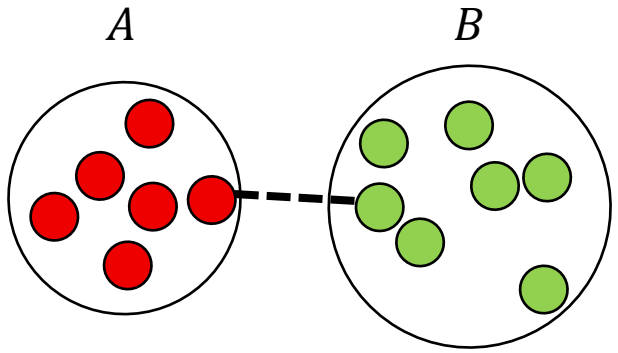
Hierarchical Agglomerative Clustering (HAC)

Consider two clusters A and B, consider the following distance measures $l(A, B)$ defined based on a point-wise distance function $d(x, y)$:

Single-link

- The distance between the nearest points

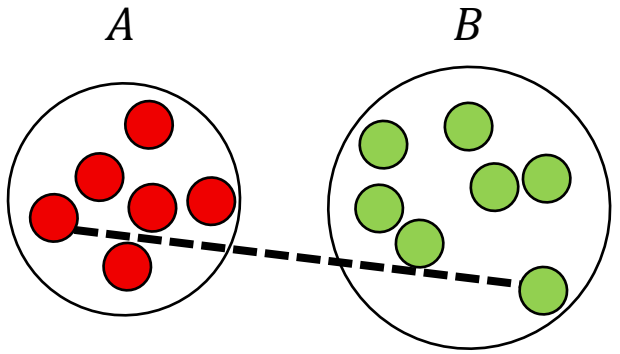
$$d(A, B) = \min_{x_a \in A, x_b \in B} d(x_a, x_b)$$



Complete-link

- The distance between the furthest points

$$d(A, B) = \max_{x_a \in A, x_b \in B} d(x_a, x_b)$$



Slide adapted from Xiaoli Fern

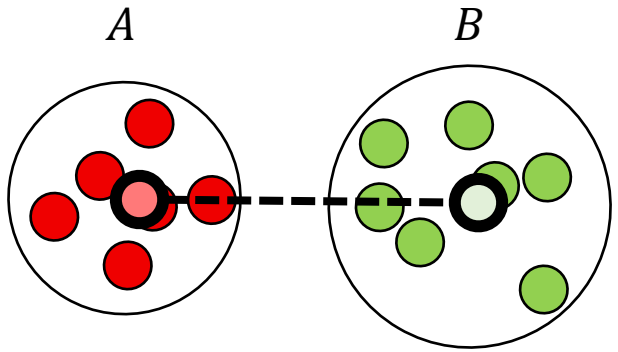
Hierarchical Agglomerative Clustering (HAC)

Consider two clusters A and B, consider the following distance measures $l(A, B)$ defined based on a point-wise distance function $d(x, y)$:

Centroid

- The distance between the cluster means

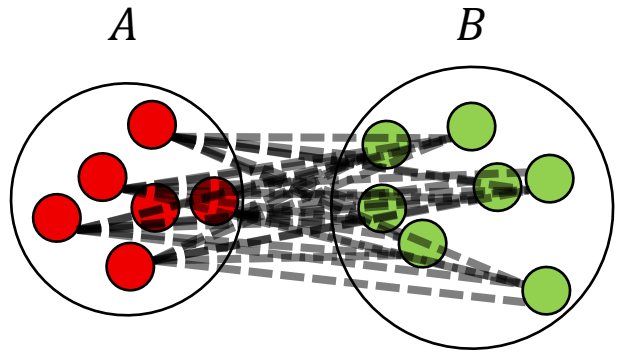
$$d(A, B) = d\left(\overline{x_a}, \overline{x_b}\right)$$



Average-link

- Average distance between all cross-cluster pairs

$$d(A, B) = \frac{1}{|A||B|} \sum_{x_a \in A} \sum_{x_b \in B} d(x_a, x_b)$$



- **Most robust and most commonly used**

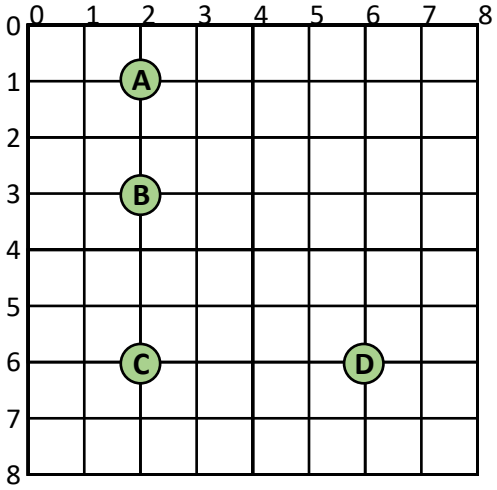
Slide adapted from Xiaoli Fern

Single-Link Method Example (L1 distance for simplicity)

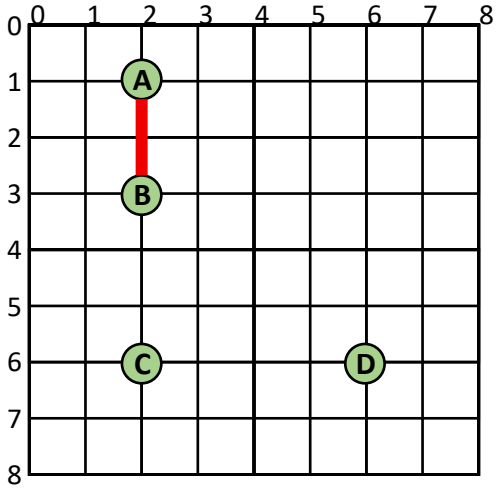
Single-link

- The distance between the nearest points

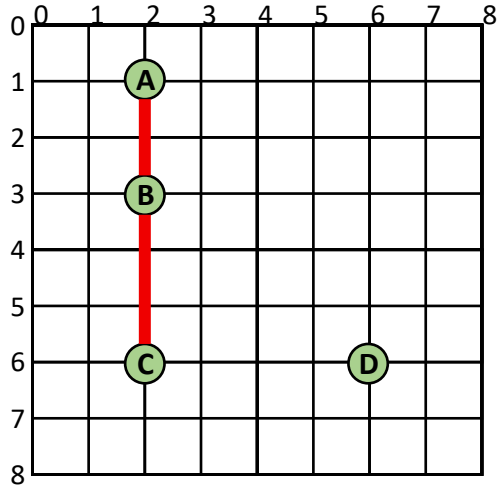
$$d(A,B) = \min_{x_a \in A, x_b \in B} d(x_a, x_b)$$



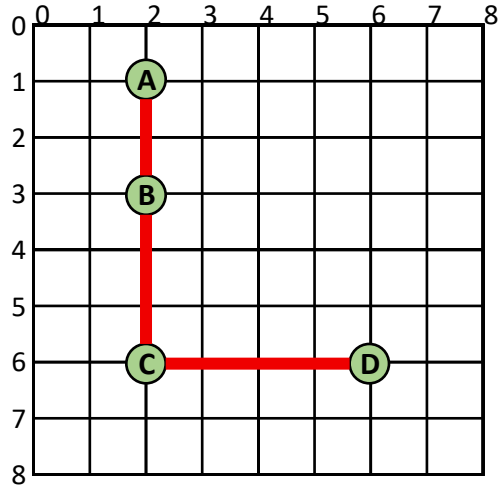
	A	B	C	D
A	-	2	5	9
B		-	3	7
C			-	4
D				-



	AB	C	D
AB	-	3	7
C		-	4
D			-



	ABC	D
ABC	-	4
D		-



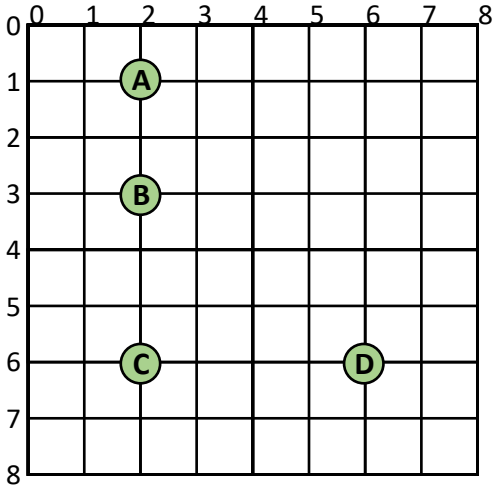
Slide adapted from Xiaoli Fern

Complete-Link Method Example (L1 distance)

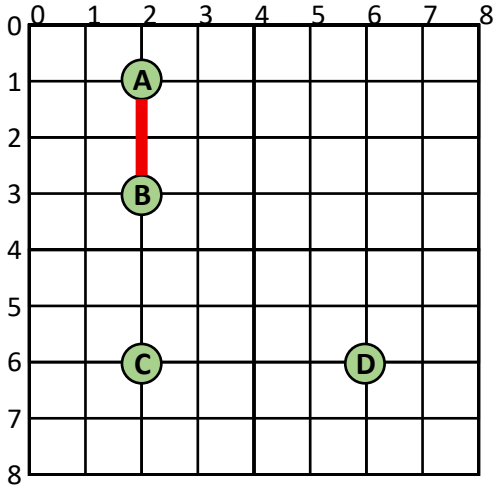
Complete-link

- The distance between the furthest points

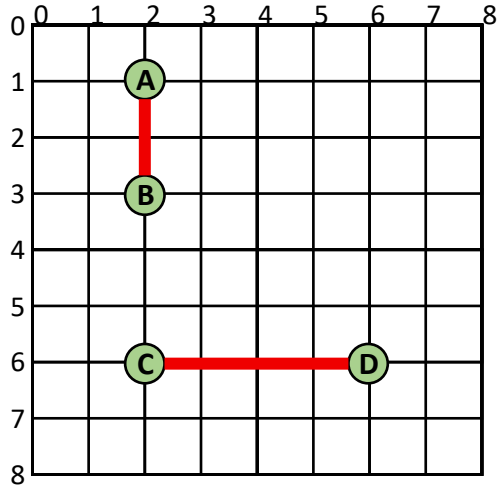
$$d(A,B) = \max_{x_a \in A, x_b \in B} d(x_a, x_b)$$



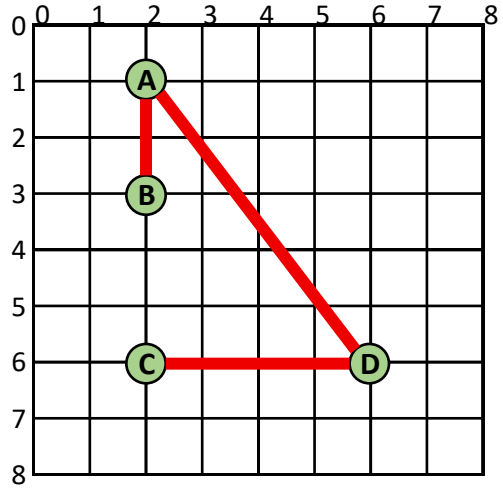
	A	B	C	D
A	-	2	5	9
B		-	3	7
C			-	4
D				-



	AB	C	D
AB	-	5	9
C		-	4
D			-



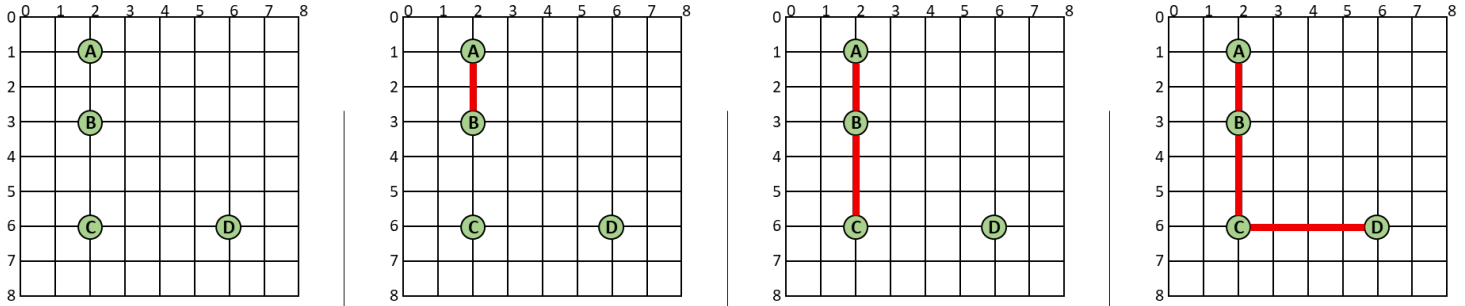
	AB	CD
AB	-	9
CD		-



Slide adapted from Xiaoli Fern

Visualizing Hierarchical Clusterings: Dendrogram

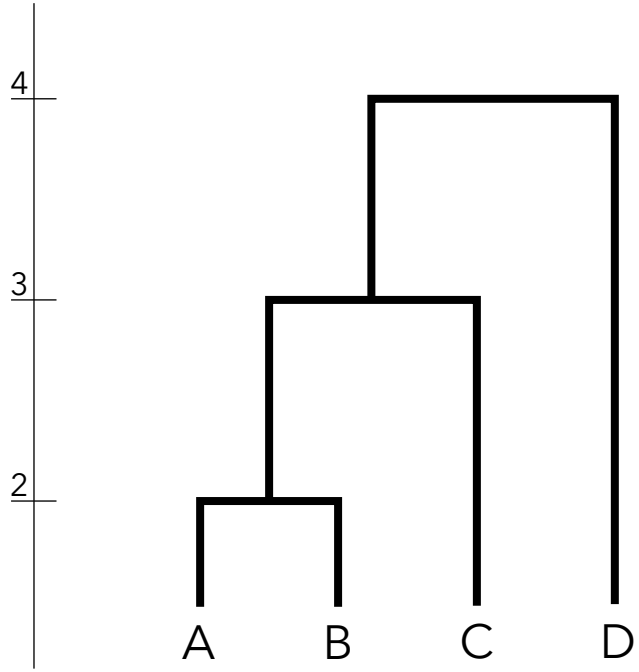
- Height of joint is the distance between the two merged clusters.
- Merge distance monotonically increased as we merge more for single / complete / average linking (not for centroid)



	A	B	C	D
A	-	2	5	9
B		-	3	7
C			-	4
D				-

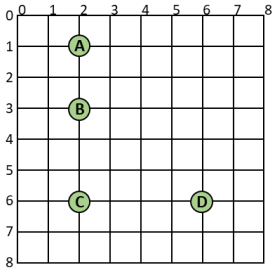
	AB	C	D
AB	-	3	7
C		-	4
D			-

	ABC	D
ABC	-	4
D		-

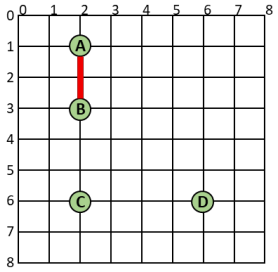


Slide adapted from Xiaoli Fern

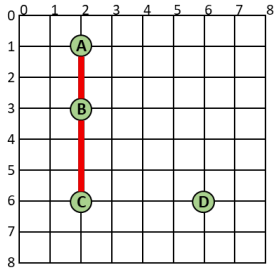
Visualizing Hierarchical Clusterings: Dendrogram



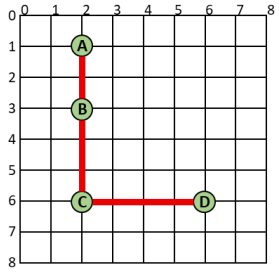
	A	B	C	D
A	-	2	5	9
B		-	3	7
C			-	4
D				-

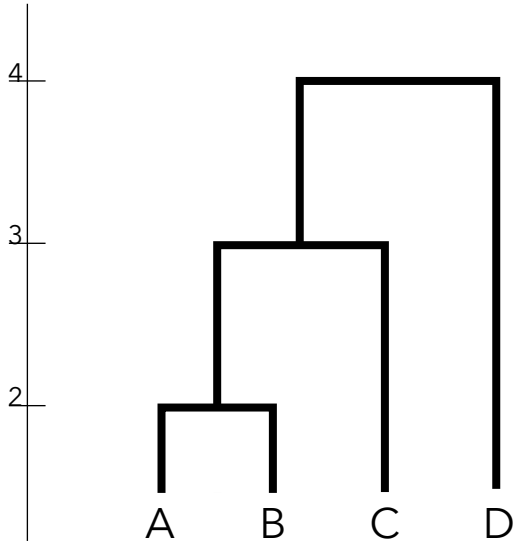


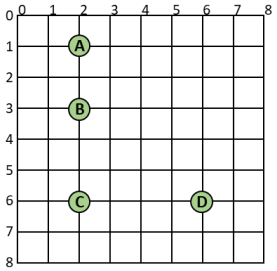
	AB	C	D
AB	-	3	7
C		-	4
D			-



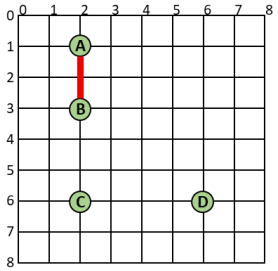
	ABC	D
ABC	-	4
D		-



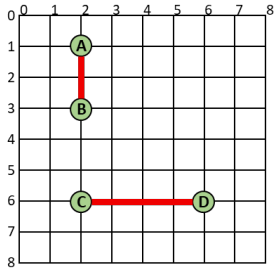




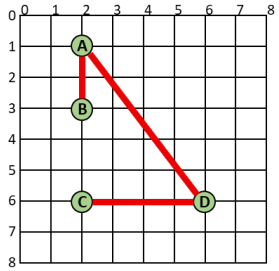
	A	B	C	D
A	-	2	5	9
B		-	3	7
C			-	4
D				-

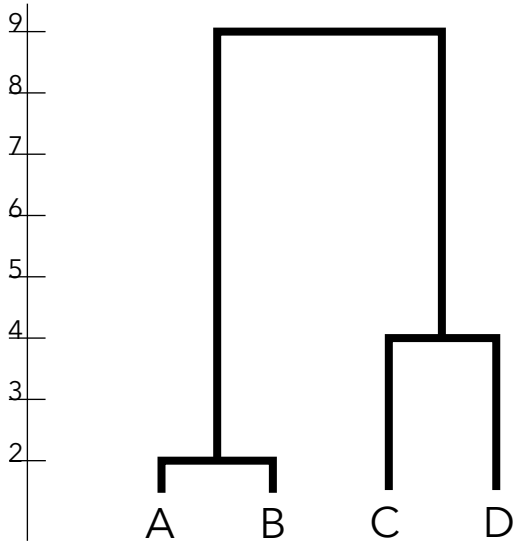


	AB	C	D
AB	-	5	9
C		-	4
D			-



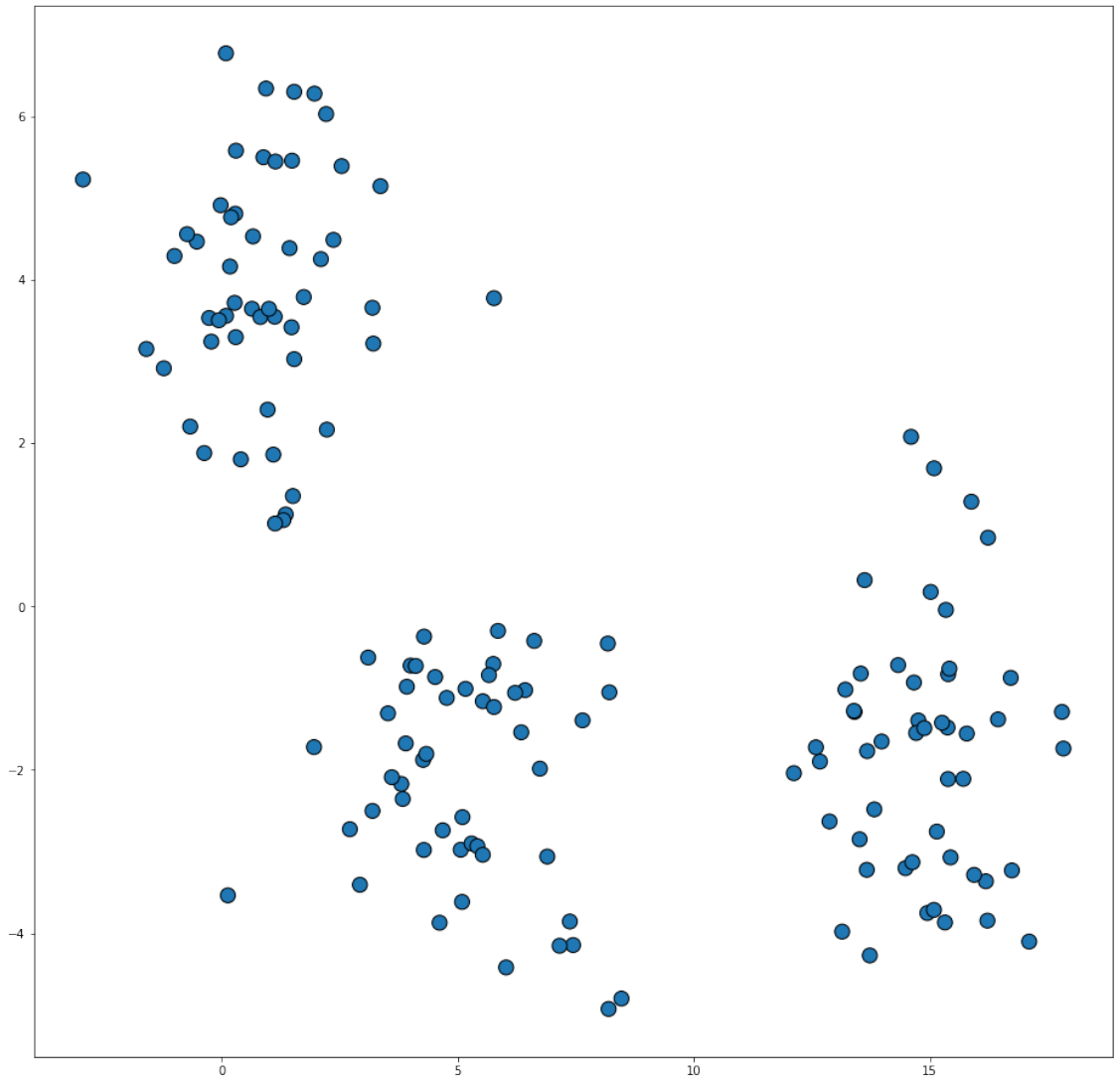
	AB	CD
AB	-	9
CD		-

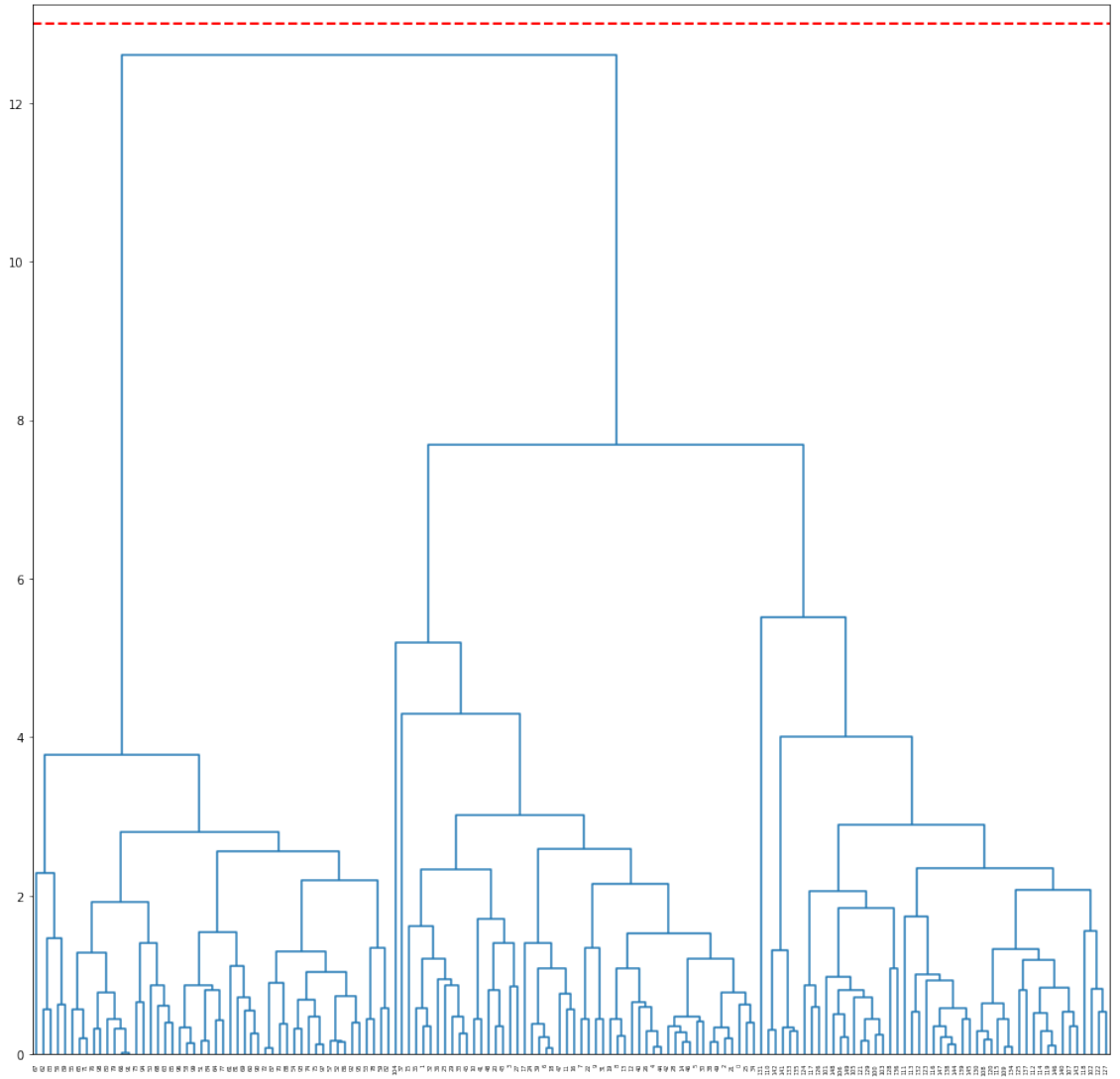




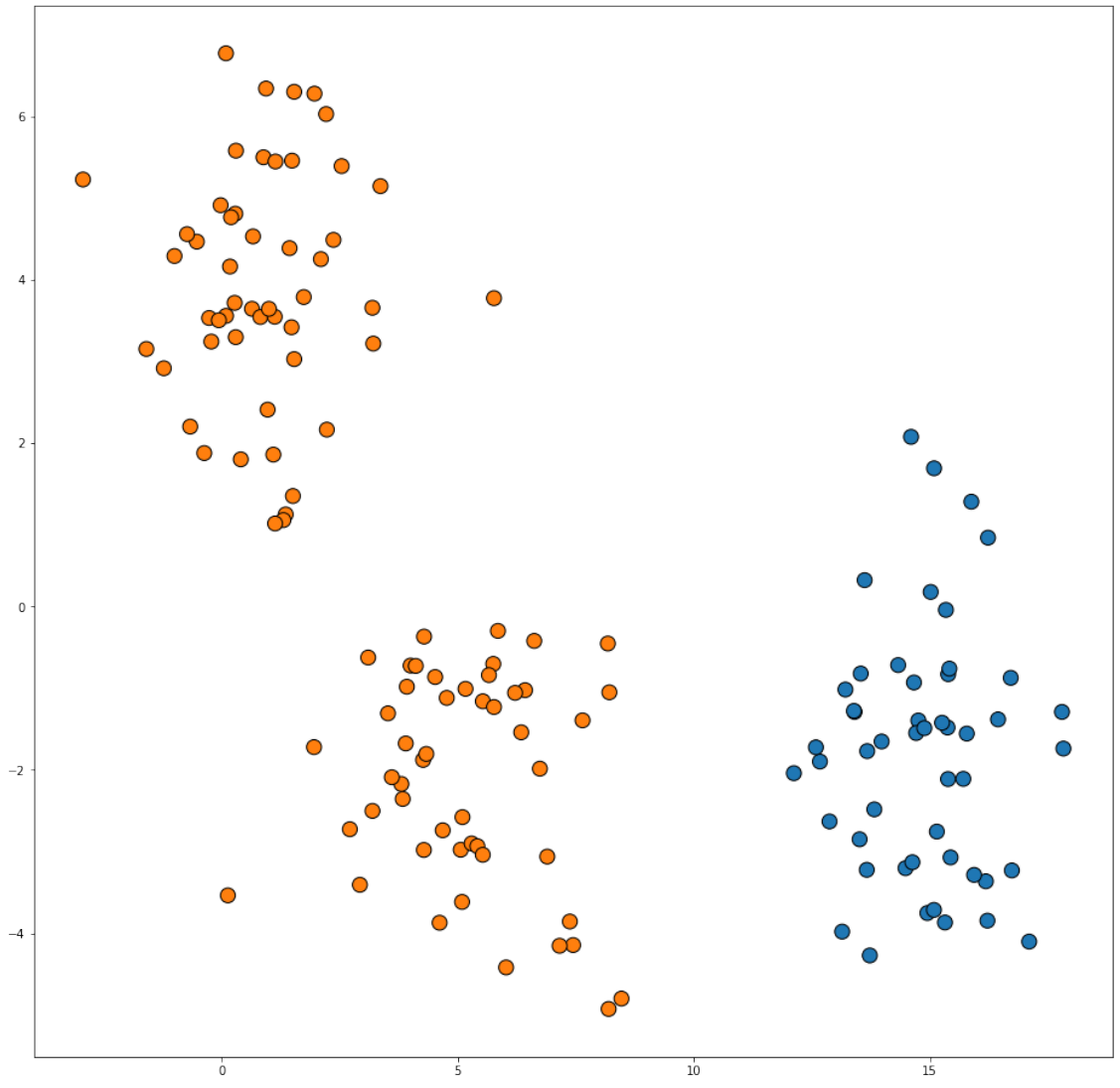
Slide adapted from Xiaoli Fern

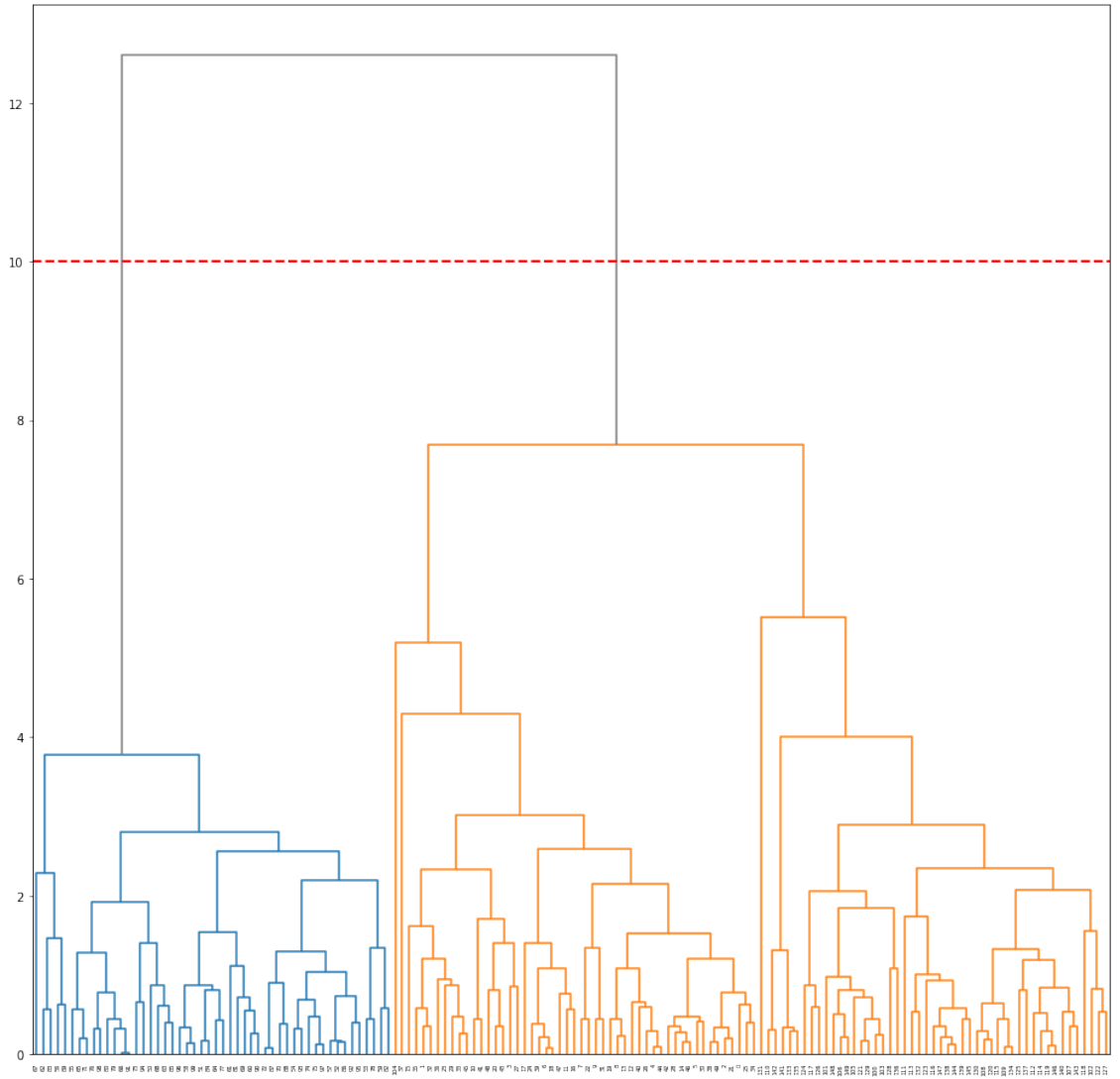
Visualizing Hierarchical Clusterings: Dendrogram





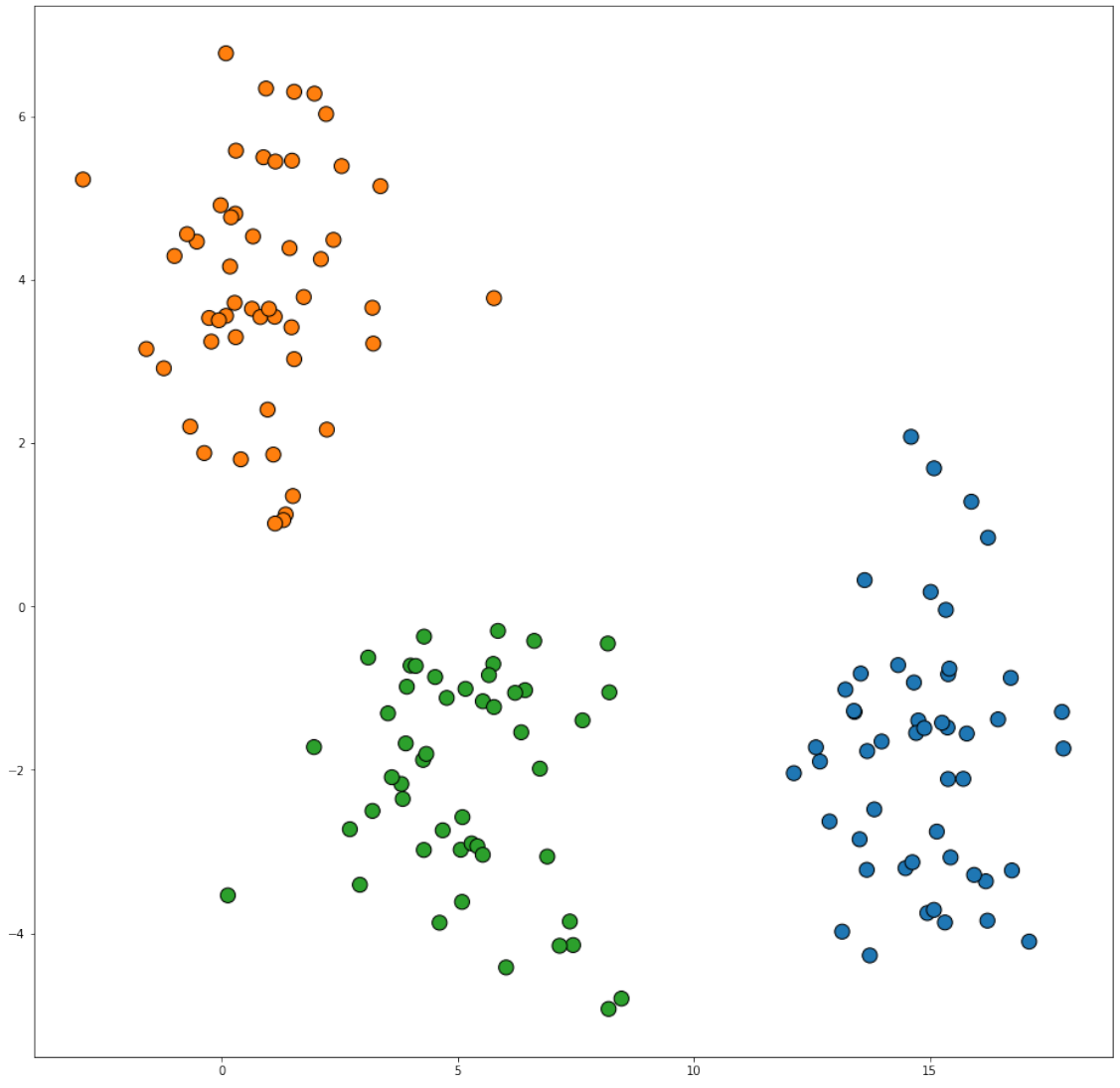
Visualizing Hierarchical Clusterings: Dendrogram

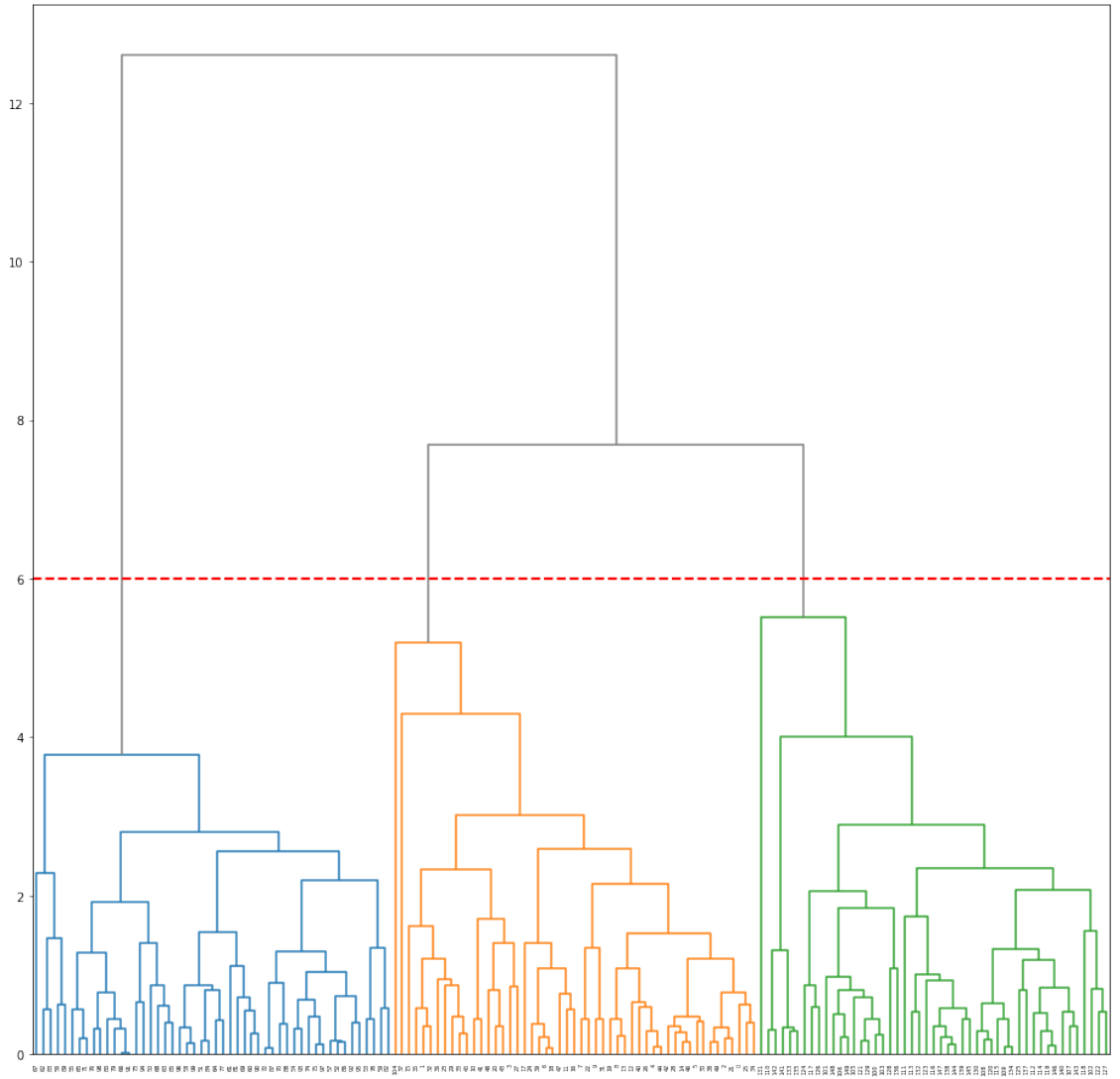




Average-link example

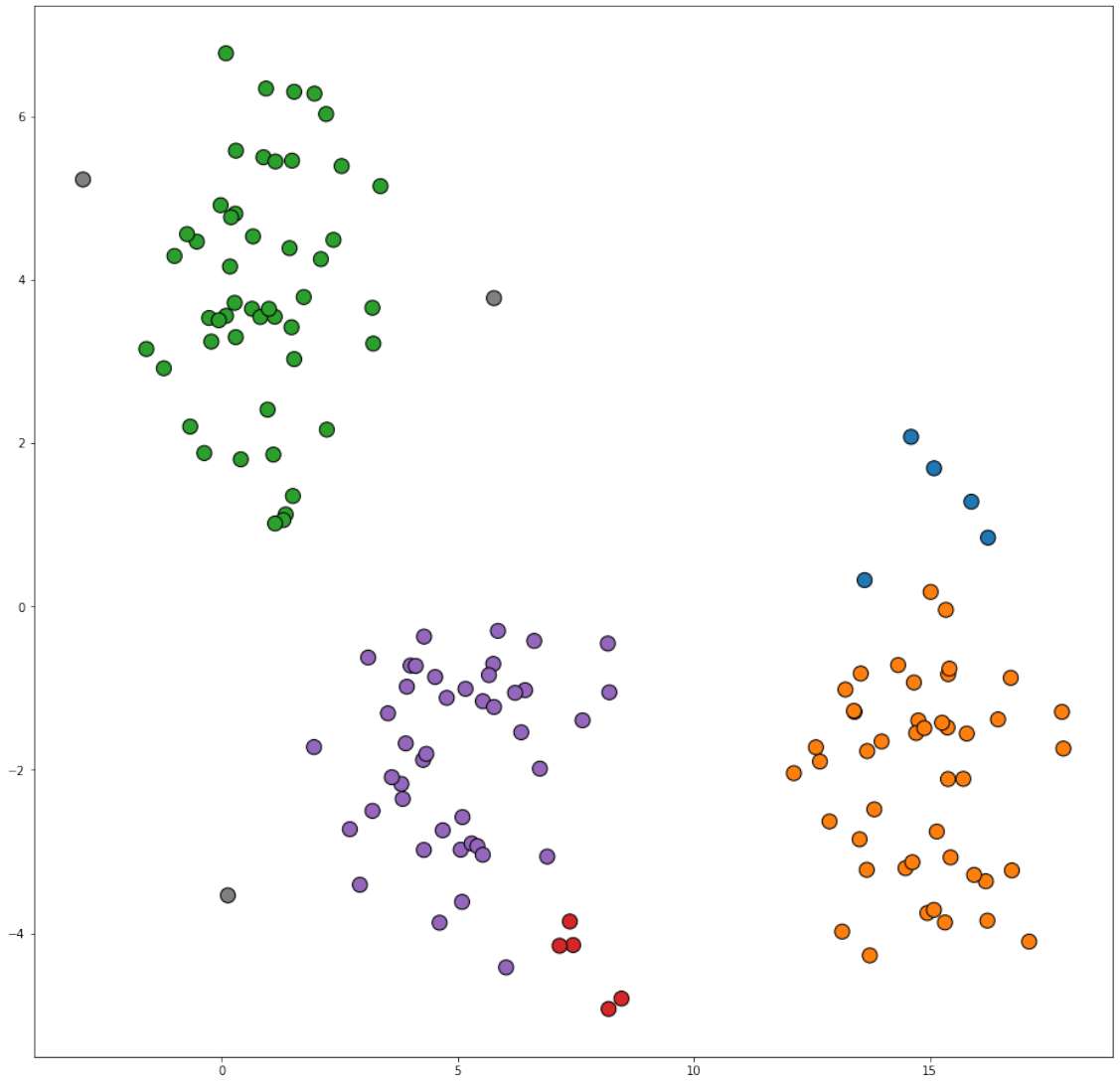
Visualizing Hierarchical Clusterings: Dendrogram

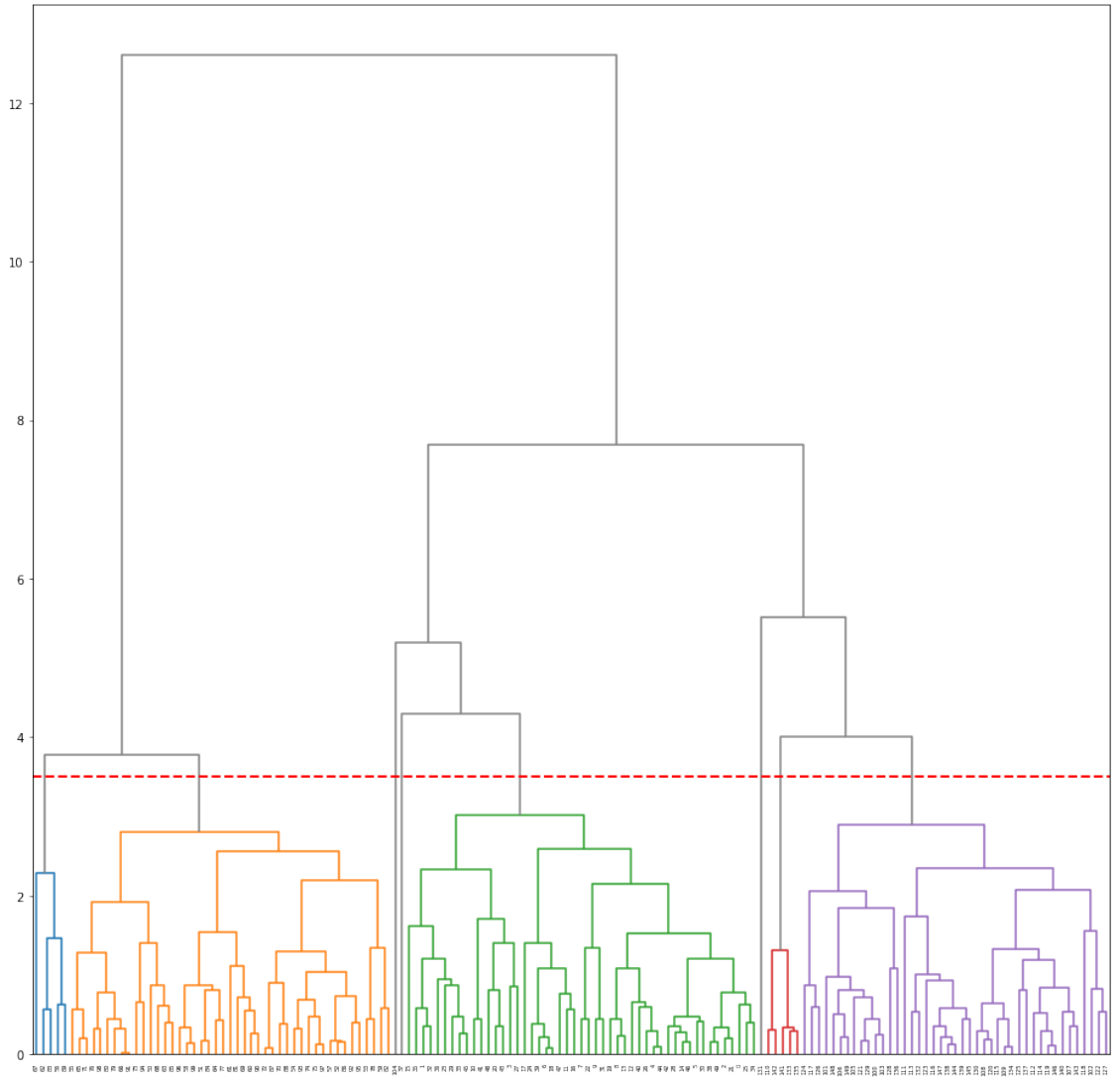




Average-link example

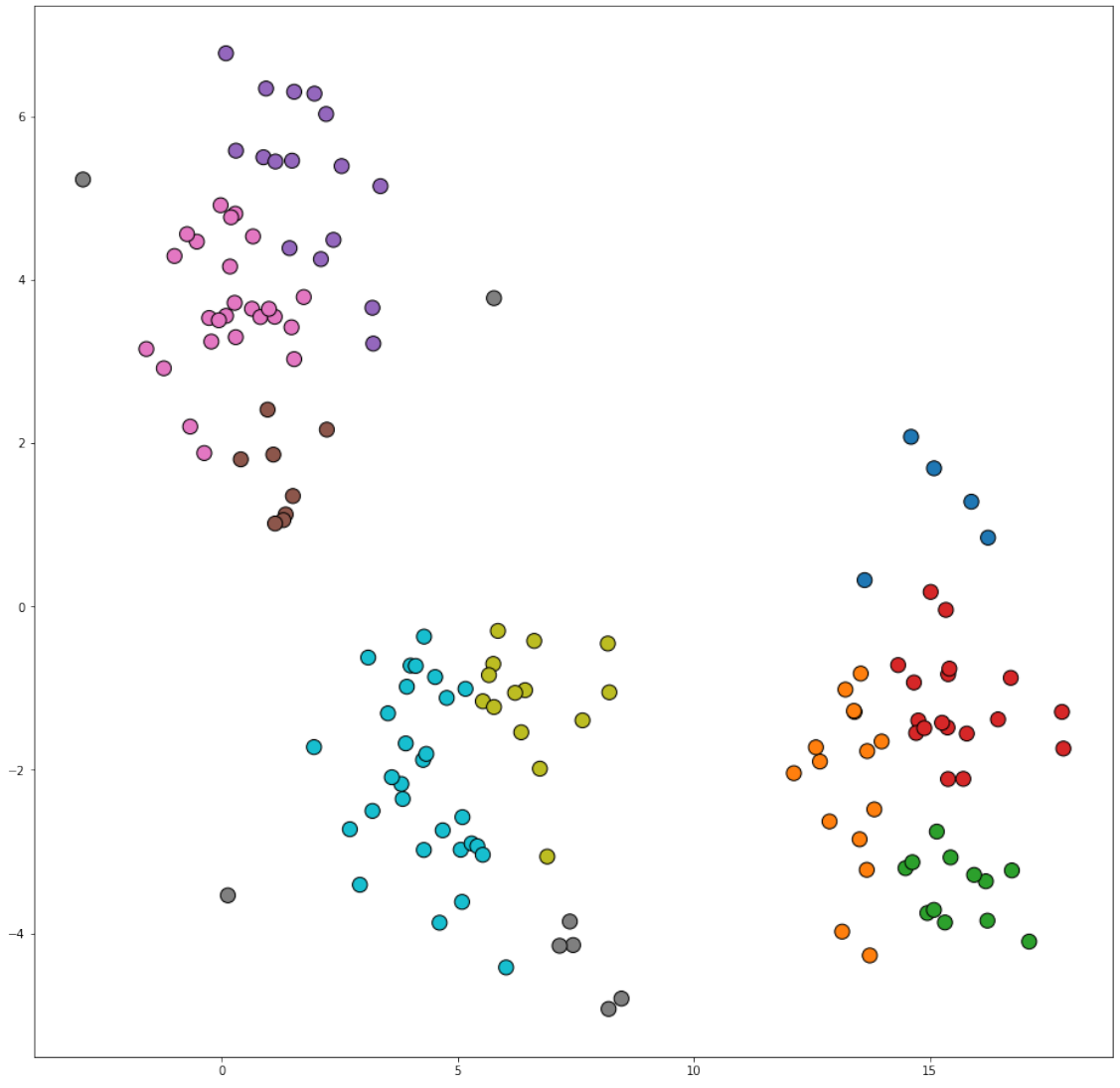
Visualizing Hierarchical Clusterings: Dendrogram

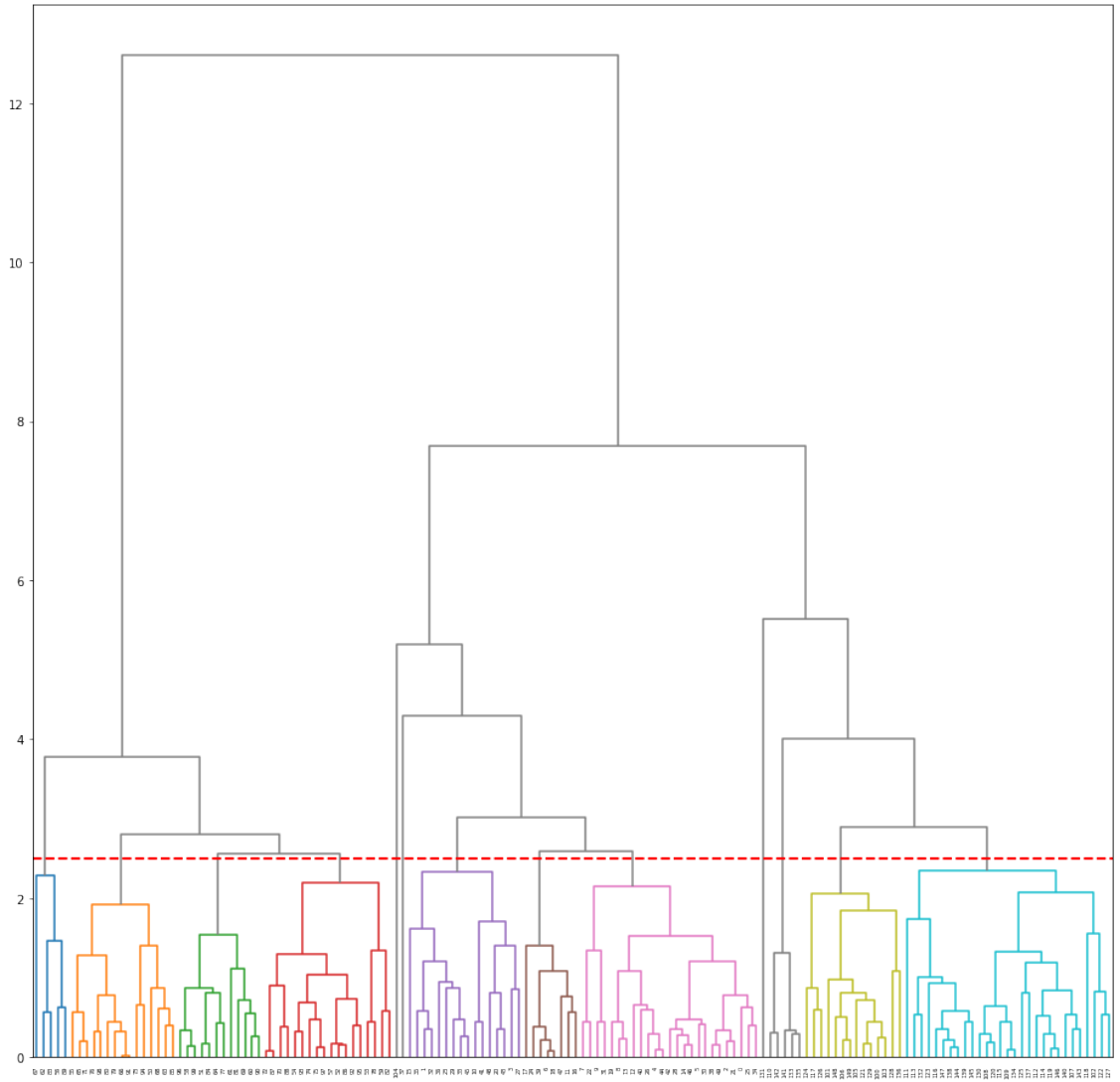




Average-link example

Visualizing Hierarchical Clusterings: Dendrogram

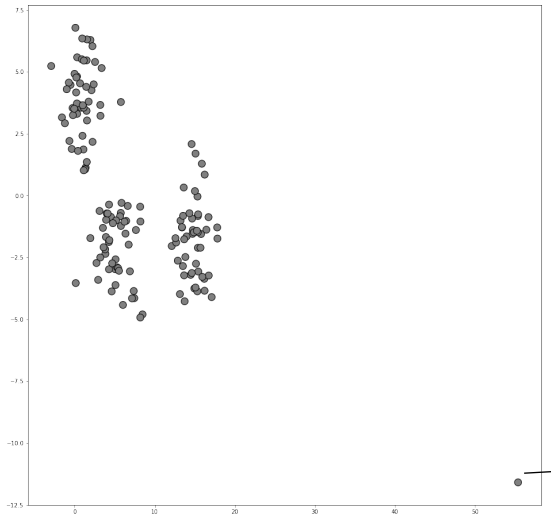


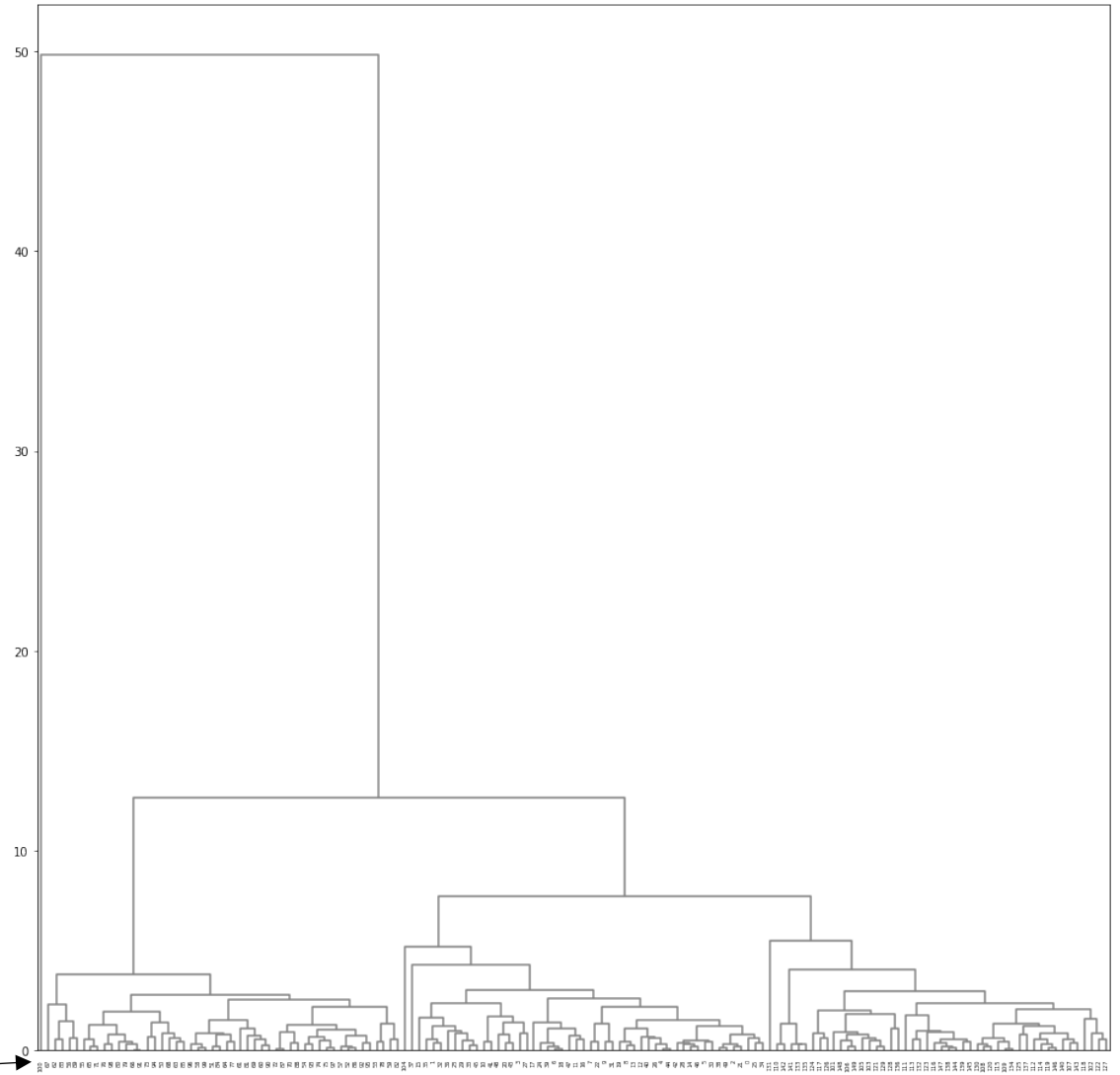


Interpreting Dendrograms

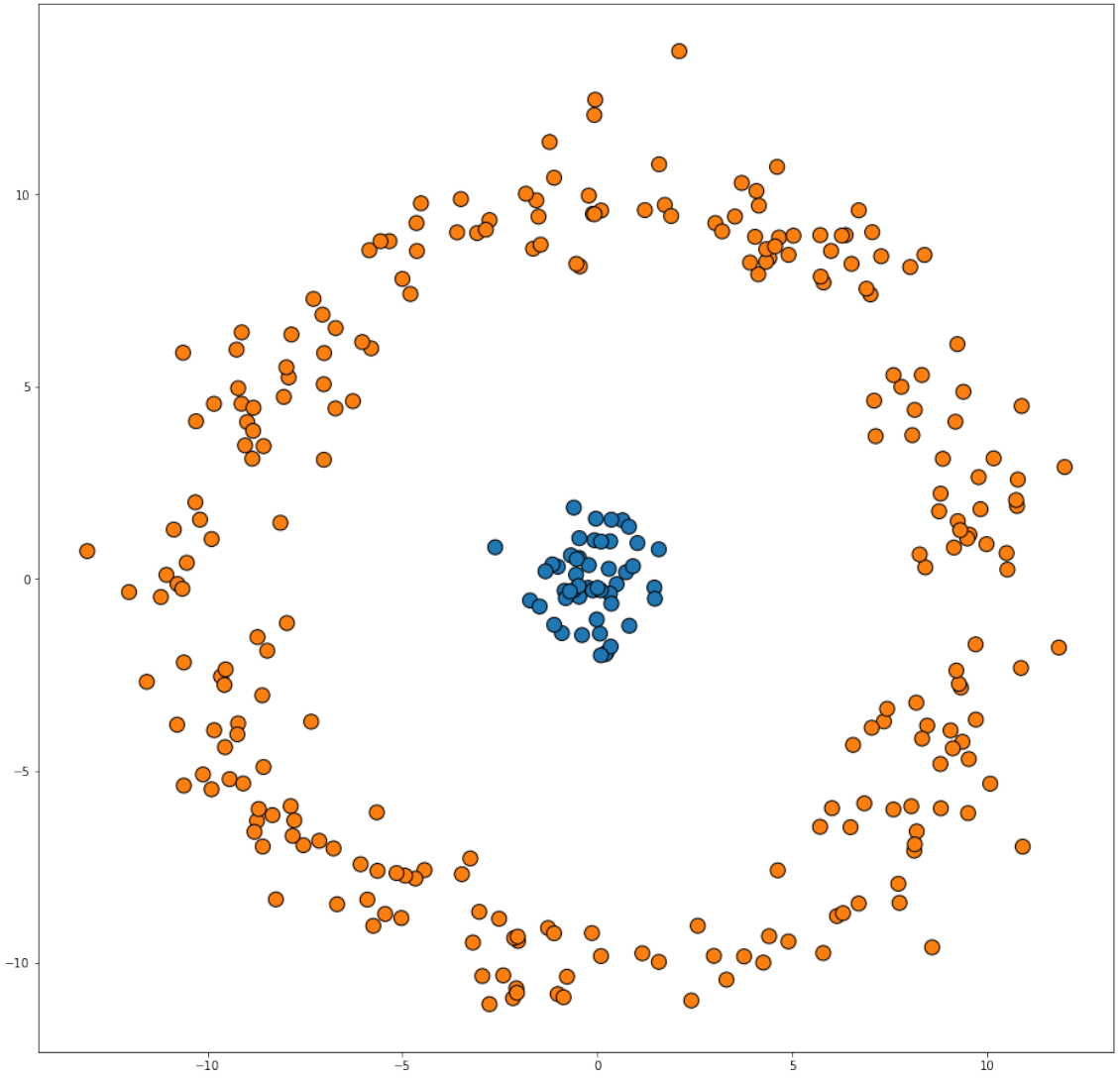
Dendrograms can be used to identify:

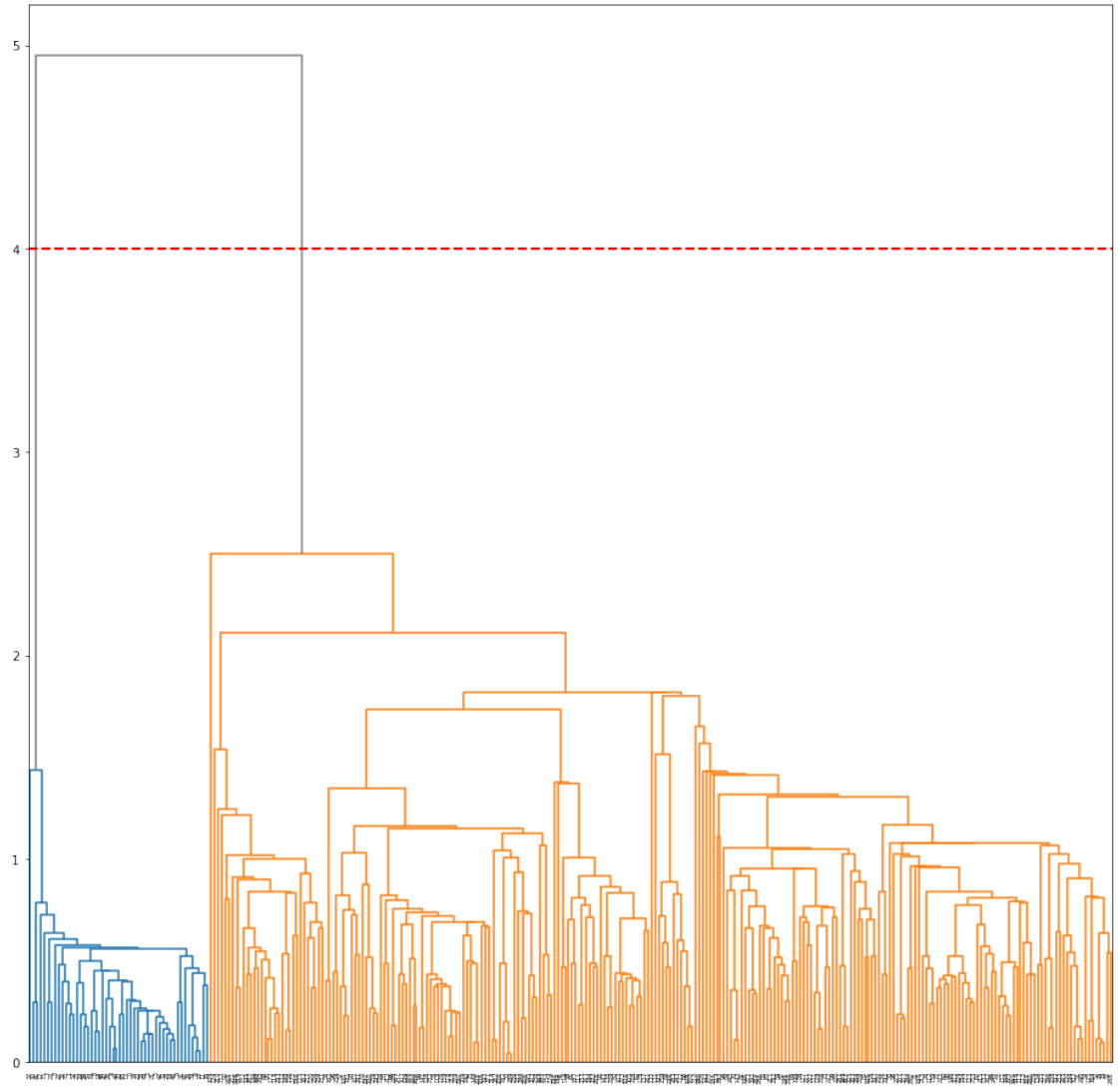
- The number of clusters (roughly)
- Well-formed clusters
- Outliers



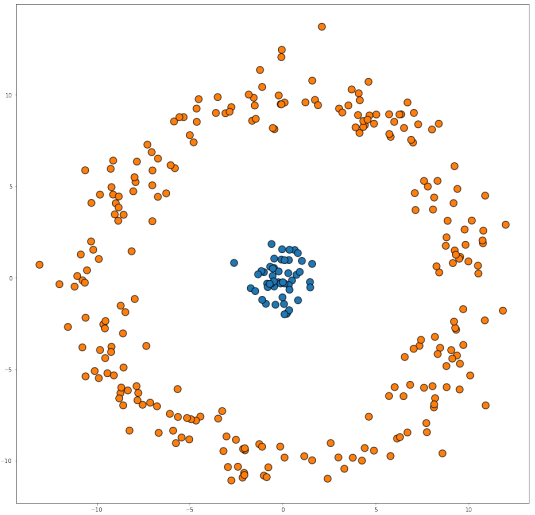


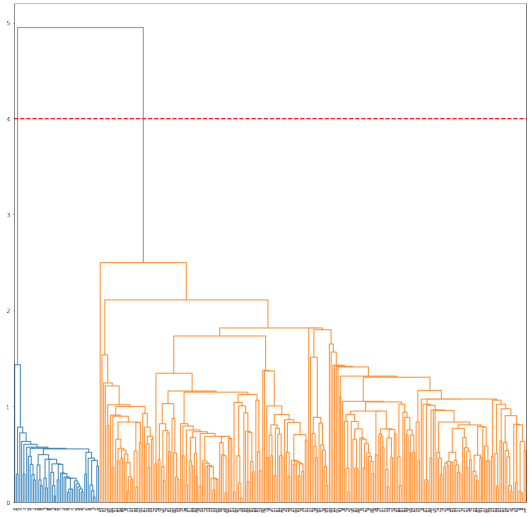
Single Link (Distance to Closest Point)





Single Link (Distance to Closest Point)



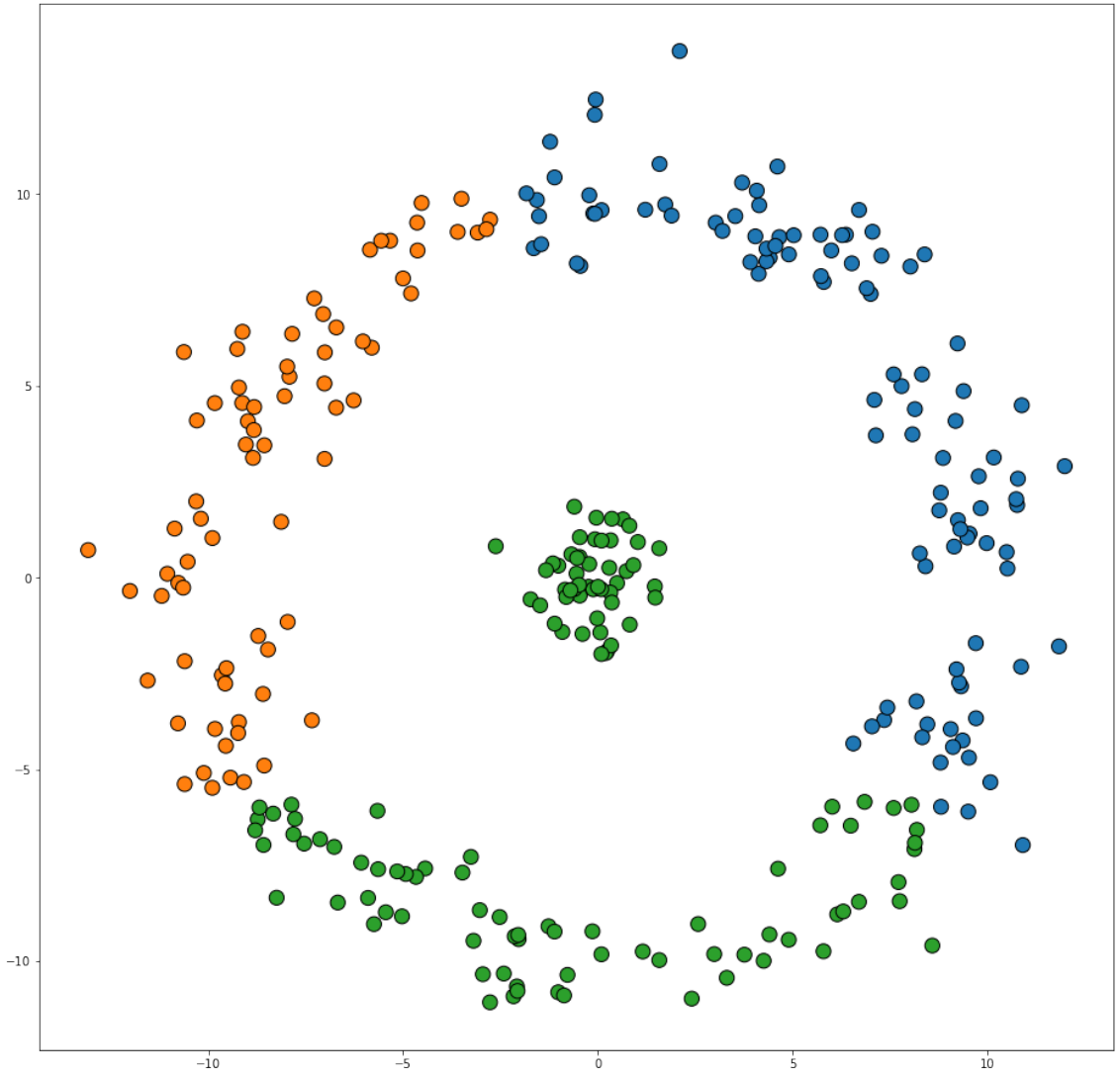


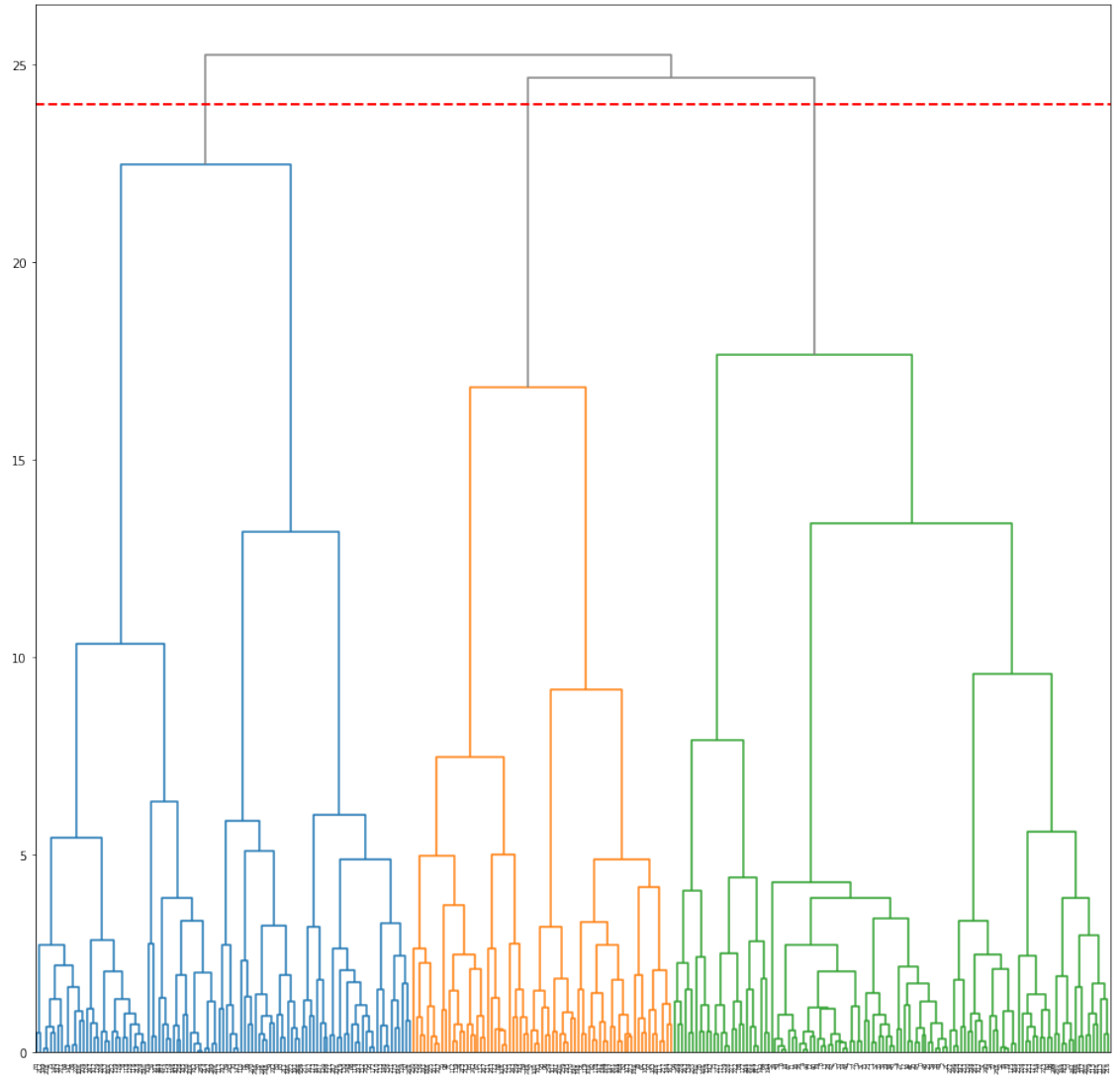
Single-link has a "chaining effect"

- Famous for its ability to gradually add more and more examples to a cluster
- Creates long, straggling cluster that are super evident in the dendrogram

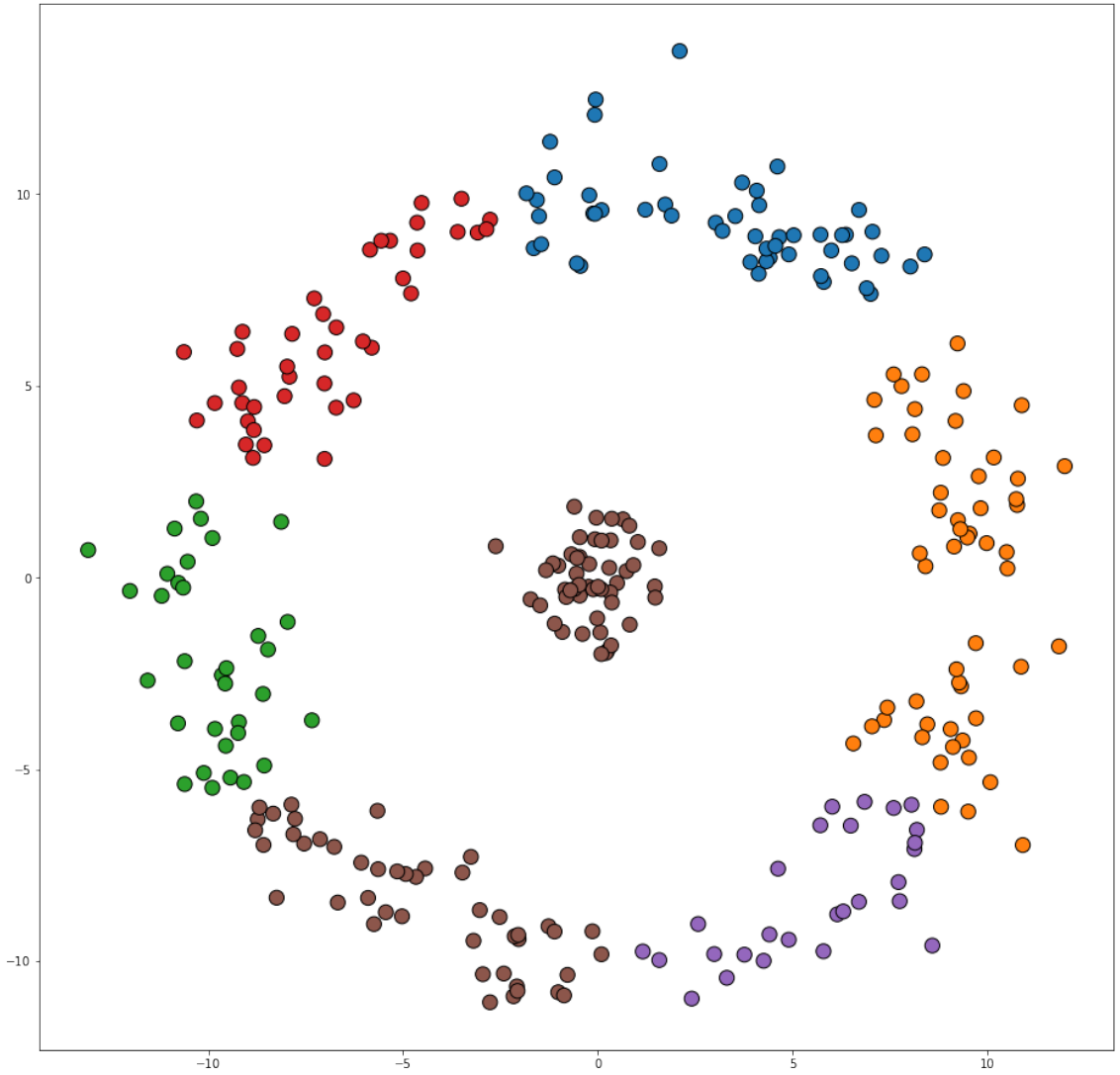
Slide adapted from Xiaoli Fern

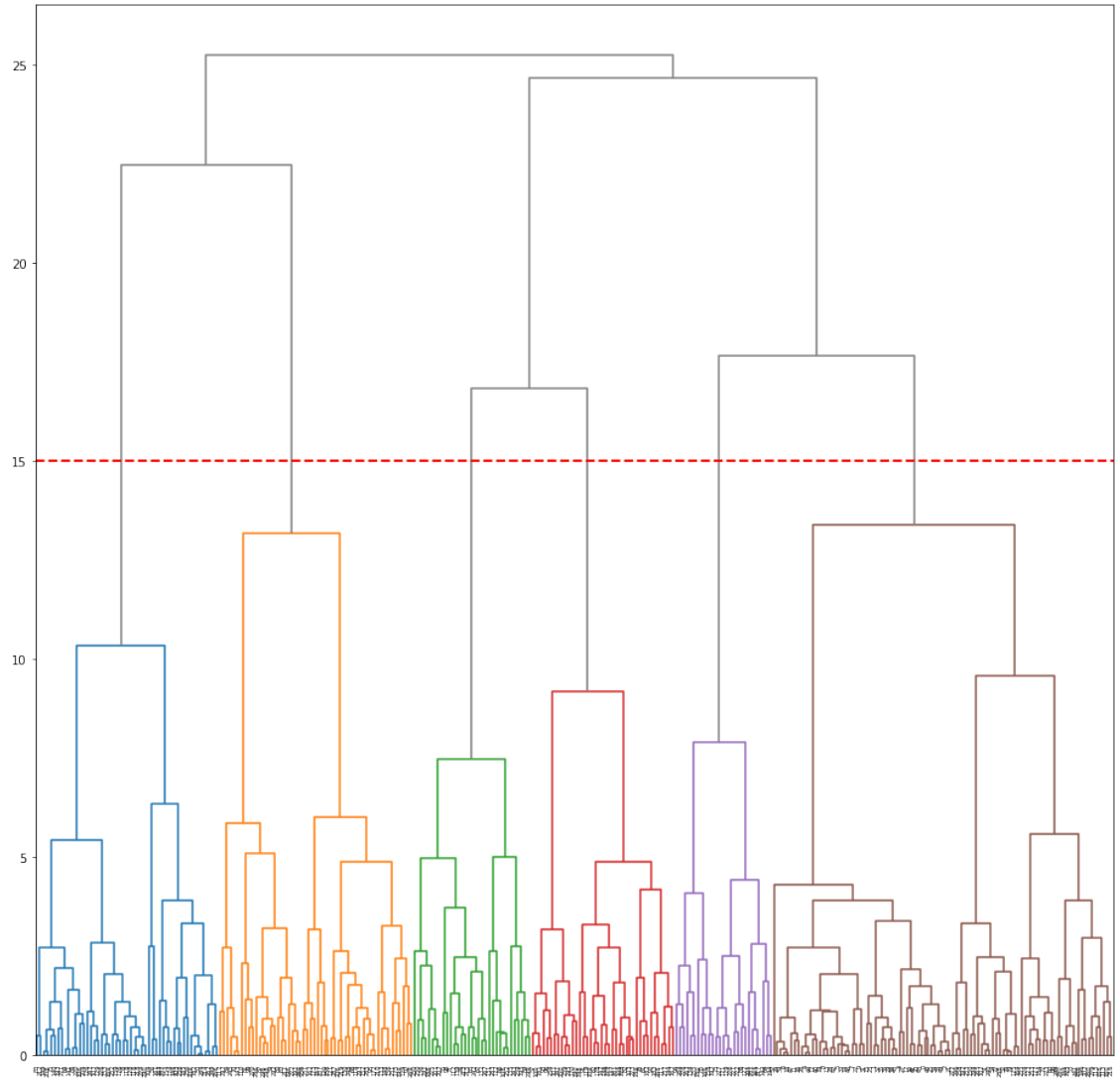
Complete Link (Distance to Furthest Point)



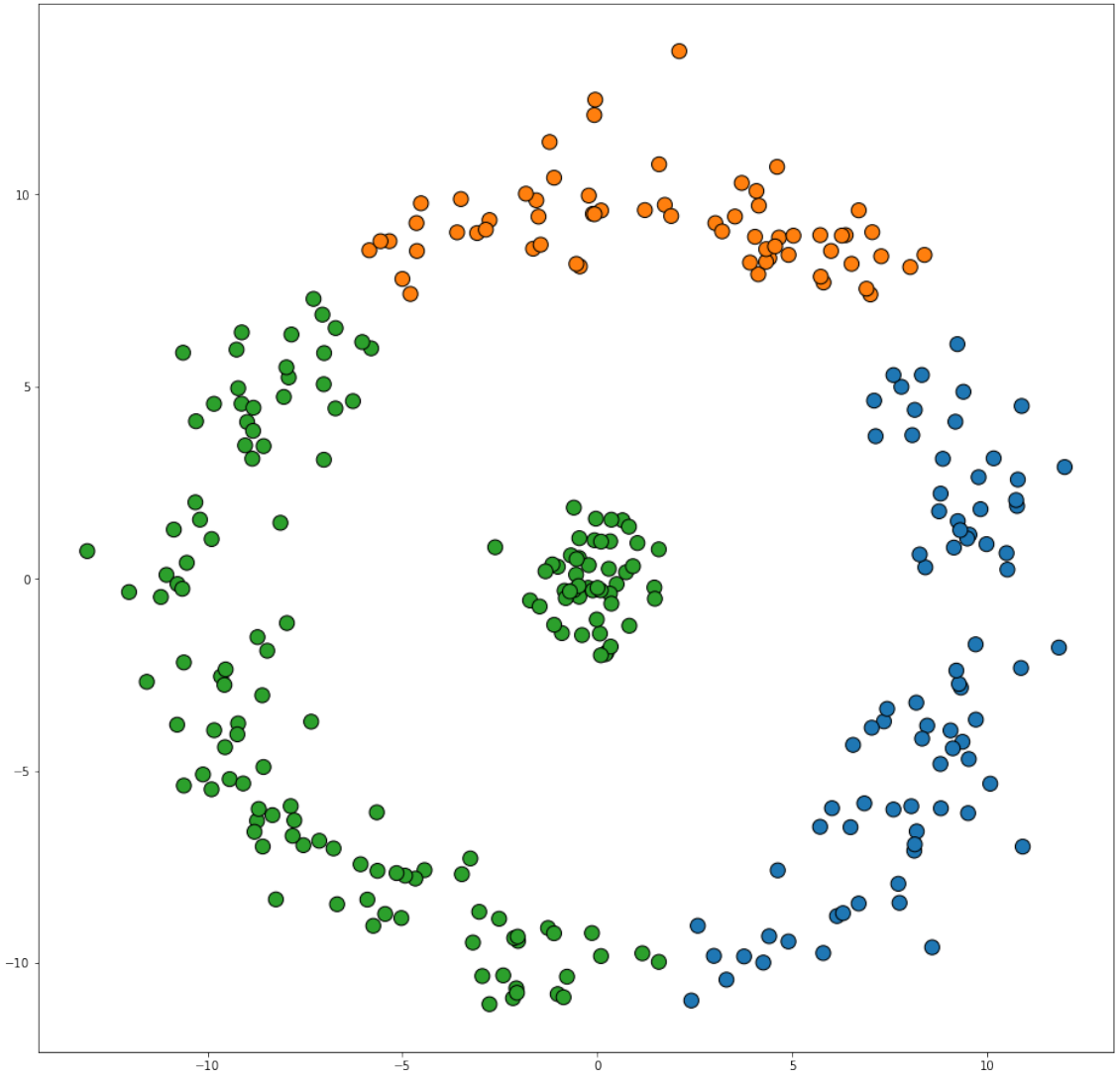


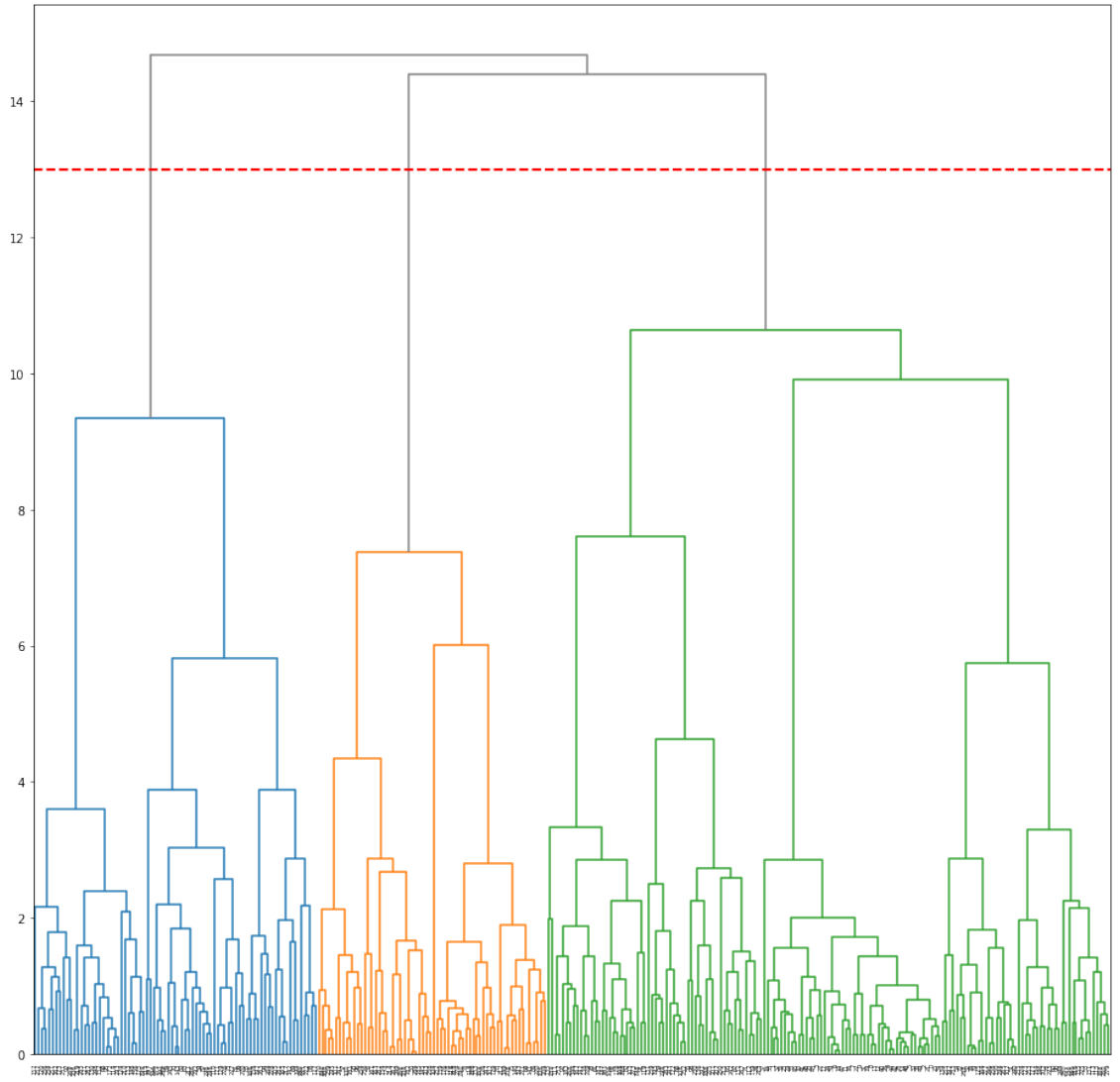
Complete Link (Distance to Furthest Point)



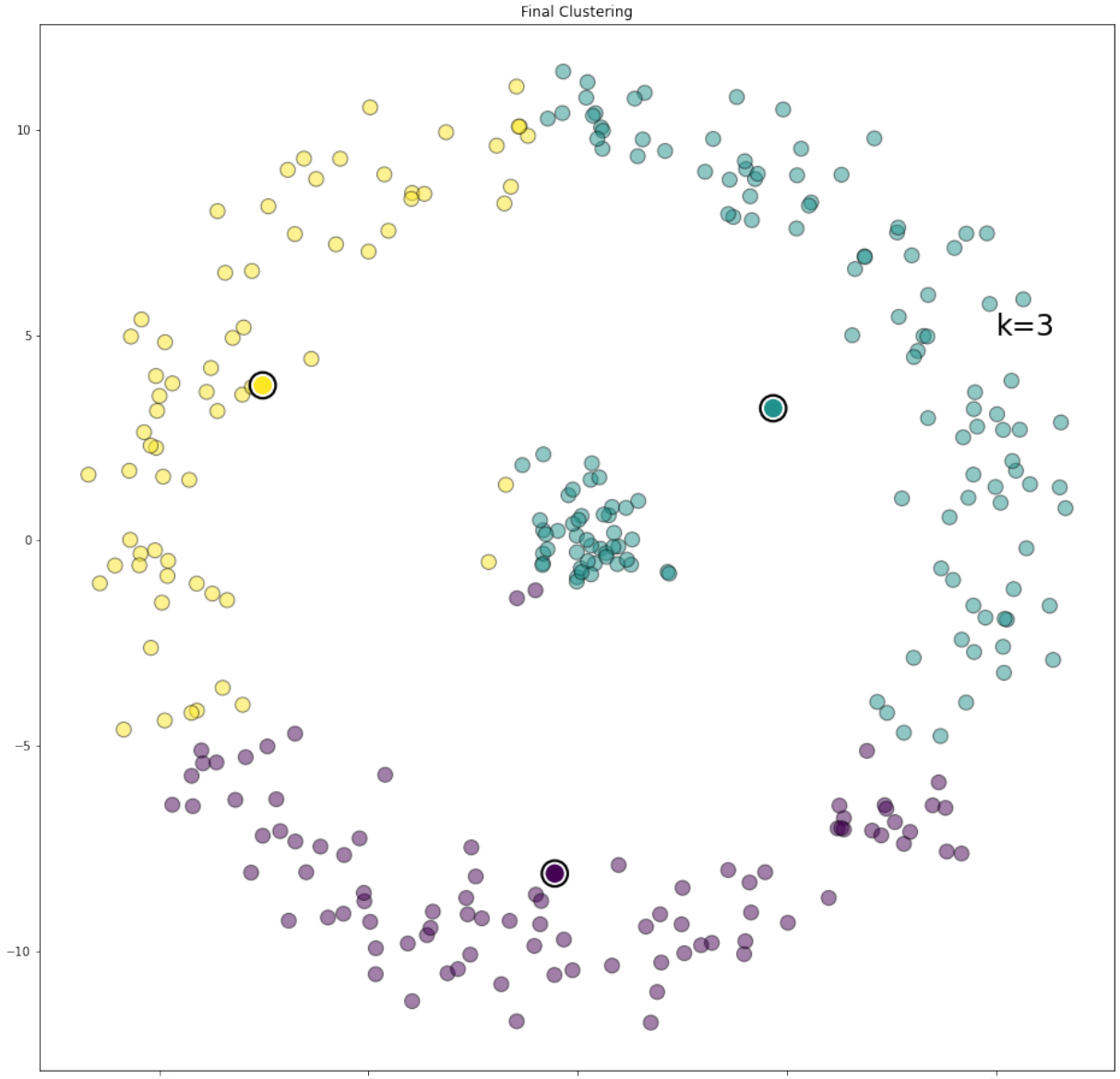


Average Link (Average Distance Between All Cross-Cluster Pairs)





k-Means



Summarizing Hierarchical Agglomerative Clustering

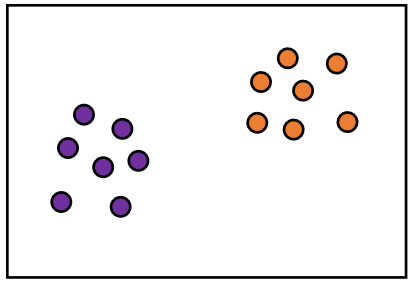
- HAC is a convenient tool that often provides interesting views of a dataset
- Primarily HAC can be viewed as an intuitively appealing clustering procedure for data analysis/exploration
- We can create clusterings of different granularity by stopping at different levels of the dendrogram
- HAC often used together with visualization of the dendrogram to decide how many clusters exist in the data
- Different linkage methods (single, complete and average) often lead to different solutions

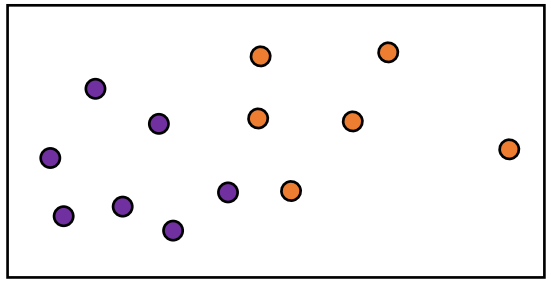
Slide adapted from Xiaoli Fern

This is all cool, but how do I know if I made a good clustering?

Without external data:

- User inspection – (aka just look at it) Does a cluster seem to have a common theme?
  - CAUTION: HUMANS ARE GOOD AT IMAGINING PATTERNS
- Internal Criterion – measure properties of a clustering presumed to be "good"
  - High within-cluster similarity: $s_w = \sum_{j=1}^{k} \sum_{x,x' \in c_j} sim(x, x')$
  - Low between-cluster similarity: $s_b = \sum_{x \in c_i} \sum_{x' \notin c_i} sim(x, x')$





  - This measure depends on the dataset and measure of distance used.

Slide adapted from Xiaoli Fern

Evaluating Clustering

With external data:

- Suppose someone comes along afterwards and gives you labels for all the points, can evaluate the clustering by how well it separated points with different labels.
- **Rand Index** – Given a clustering P and a ground truth label set G, measure the number of vector pairs that are
  - a: in the same group in both P and G (same cluster, same labels)
  - b: in the same group in P but different in G (same cluster, different labels)
  - c: in different groups in P but same in G (different cluster, same labels)
  - d: in different groups in both P and G (different clusters, different labels)

$$Rand\ Index = \frac{a + d}{a + b + c + d}$$

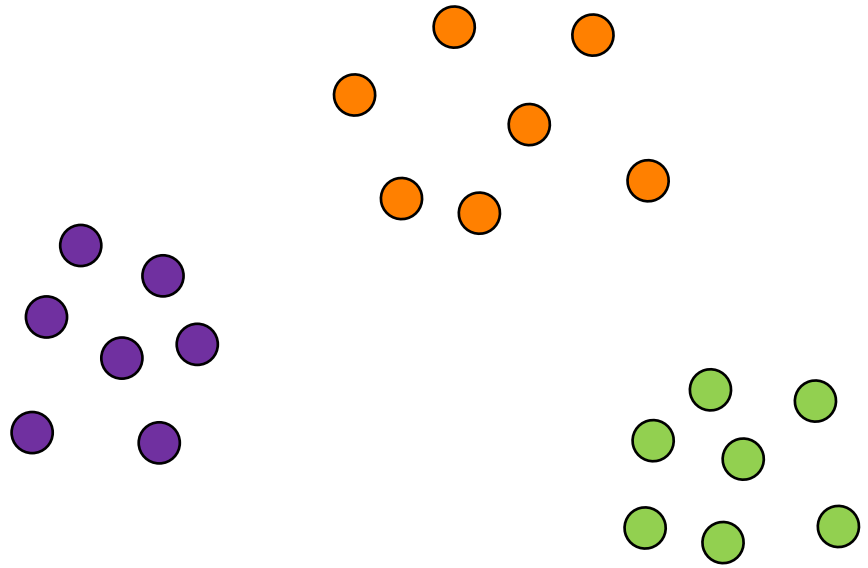
- **Adjusted Rand Index:** correct rand-index by the average rand-index of a random clustering of the data.

Slide adapted from Xiaoli Fern

Evaluating Clustering

With external data:

- Suppose someone comes along afterwards and gives you labels for all the points, can evaluate the clustering by how well it separated points with different labels.
- **Purity** – Fraction of points that would be correctly classified by a "majority vote" per cluster where all points get the label of the majority.

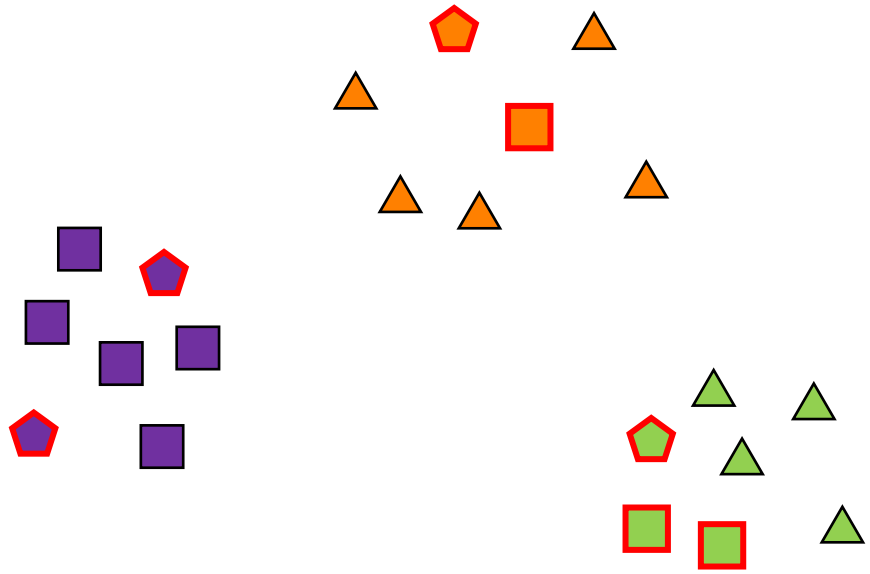


Slide adapted from Xiaoli Fern

Evaluating Clustering

With external data:

- Suppose someone comes along afterwards and gives you labels for all the points, can evaluate the clustering by how well it separated points with different labels.
- **Purity** – Fraction of points that would be correctly classified by a "majority vote" per cluster where all points get the label of the majority.



$$Purity = \frac{5 + 5 + 4}{21}$$

Slide adapted from Xiaoli Fern

Today's Learning Objectives

Be able to answer:

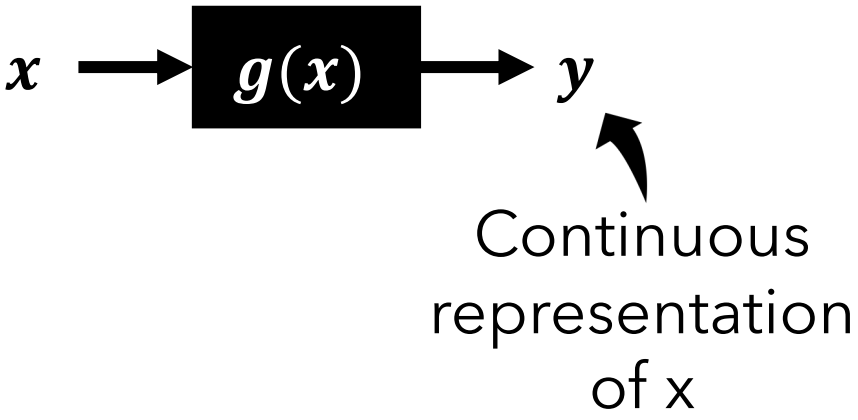
- What is Hierarchical Agglomerative Clustering (HAC)?
  - What are different linking strategies?
- How can we evaluate clustering?
- What is dimensionality reduction?
  - What is principal component analysis (PCA) and how does it work?
  - What are stochastic neighbor embeddings (SNE) and how do they work?

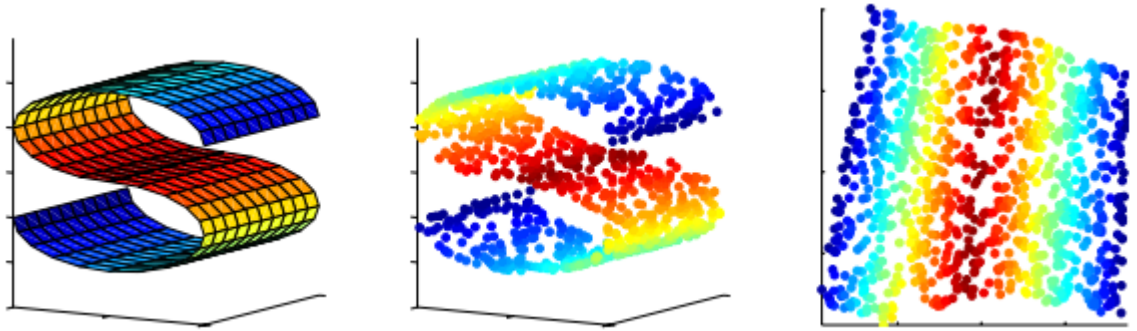
New Topic: Dimensionality Reduction

Unsupervised Learning

Only given a set of input instances (x)

Dimensionality Reduction: Represent high-dim data as low-dim data





Example: Finding a 2D subspace in a 3D feature space

New Topic: Dimensionality Reduction

Dimensionality reduction tries to find a more compact representation of the data -- creating new features with lower dimensionality that still represents the data well.

Why would we want to do this?

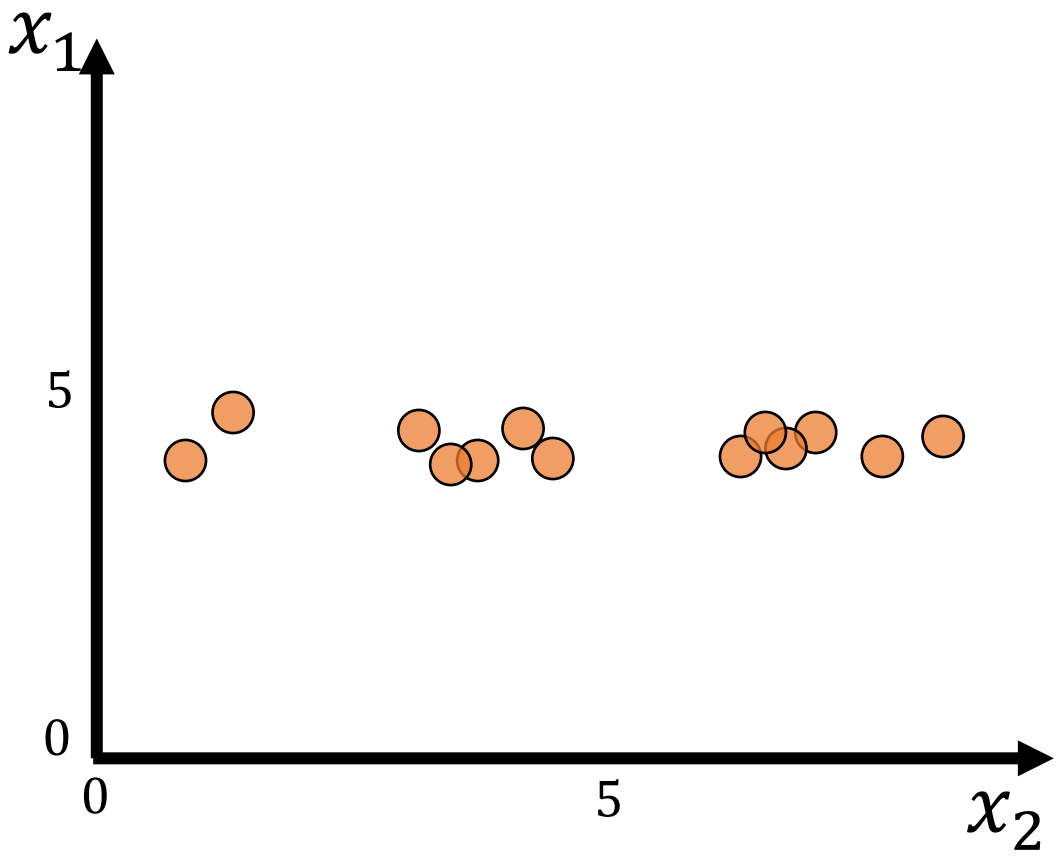
- **Visualization**: We can't plot things in >3 dimensions (and even 3D plots confuse me…). Can we find a 2-3 dimension view of our data that captures the meaningful relationships?

- **Preprocessing:** Many of the learning algorithms we've seen are more computationally expensive with large dimensionality. Further, many of them work better with lower dimensional data.

Build Some Intuition About Dimensionality Reduction

Say I came to you and said,

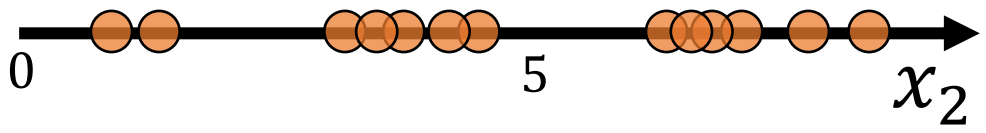
"We don't have enough memory to put both dimensions of our data through our *GIANT MODEL*™, which dimension do you think we can get rid of?"



Build Some Intuition About Dimensionality Reduction

Say I came to you and said,

"We don't have enough memory to put both dimensions of our data through our *GIANT MODEL*™, which dimension do you think we can get rid of?"

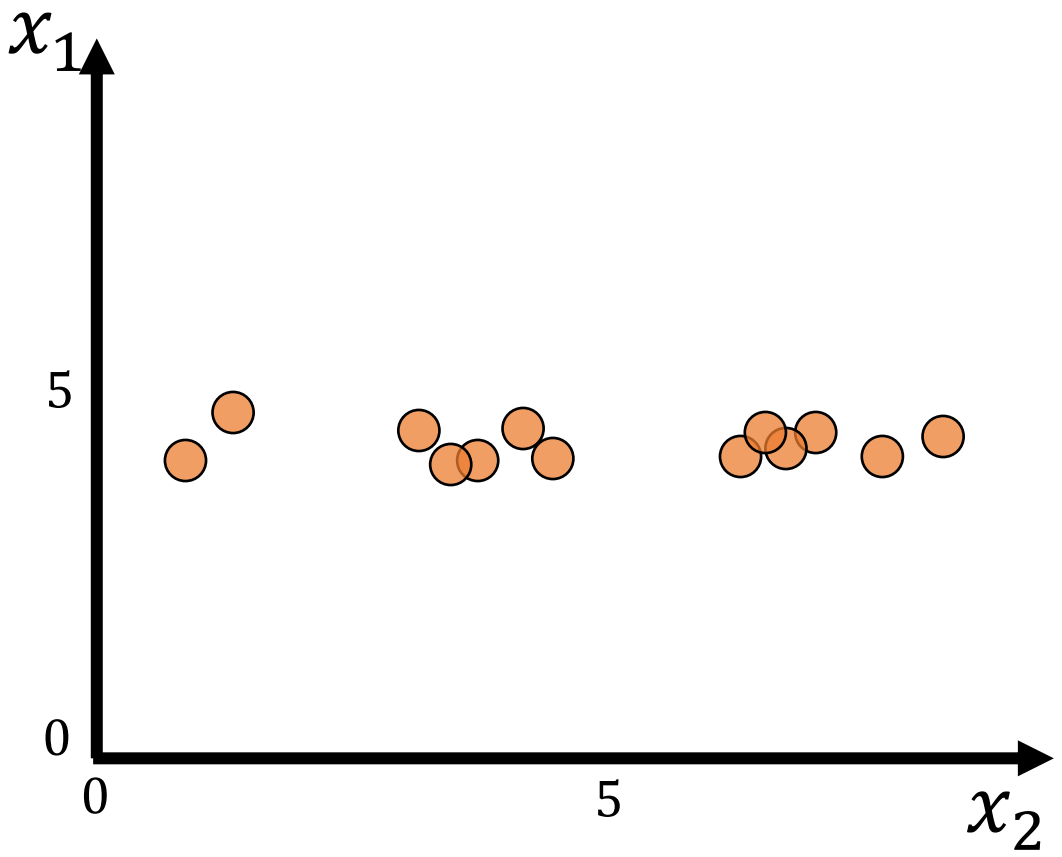


New representation just measures distance along the horizontal axis

Build Some Intuition About Dimensionality Reduction

Say I came to you and said,

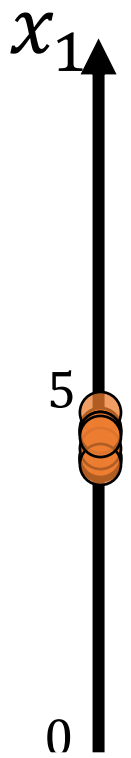
"We don't have enough memory to put both dimensions of our data through our *GIANT MODEL*™, which dimension do you think we can get rid of?"



Build Some Intuition About Dimensionality Reduction

Say I came to you and said,

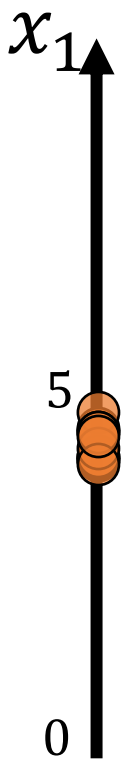
"We don't have enough memory to put both dimensions of our data through our *GIANT MODEL*™, which dimension do you think we can get rid of?"

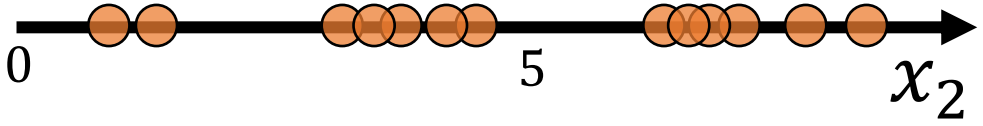


Build Some Intuition About Dimensionality Reduction

Say I came to you and said,

"We don't have enough memory to put both dimensions of our data through our *GIANT MODEL*™, which dimension do you think we can get rid of?"

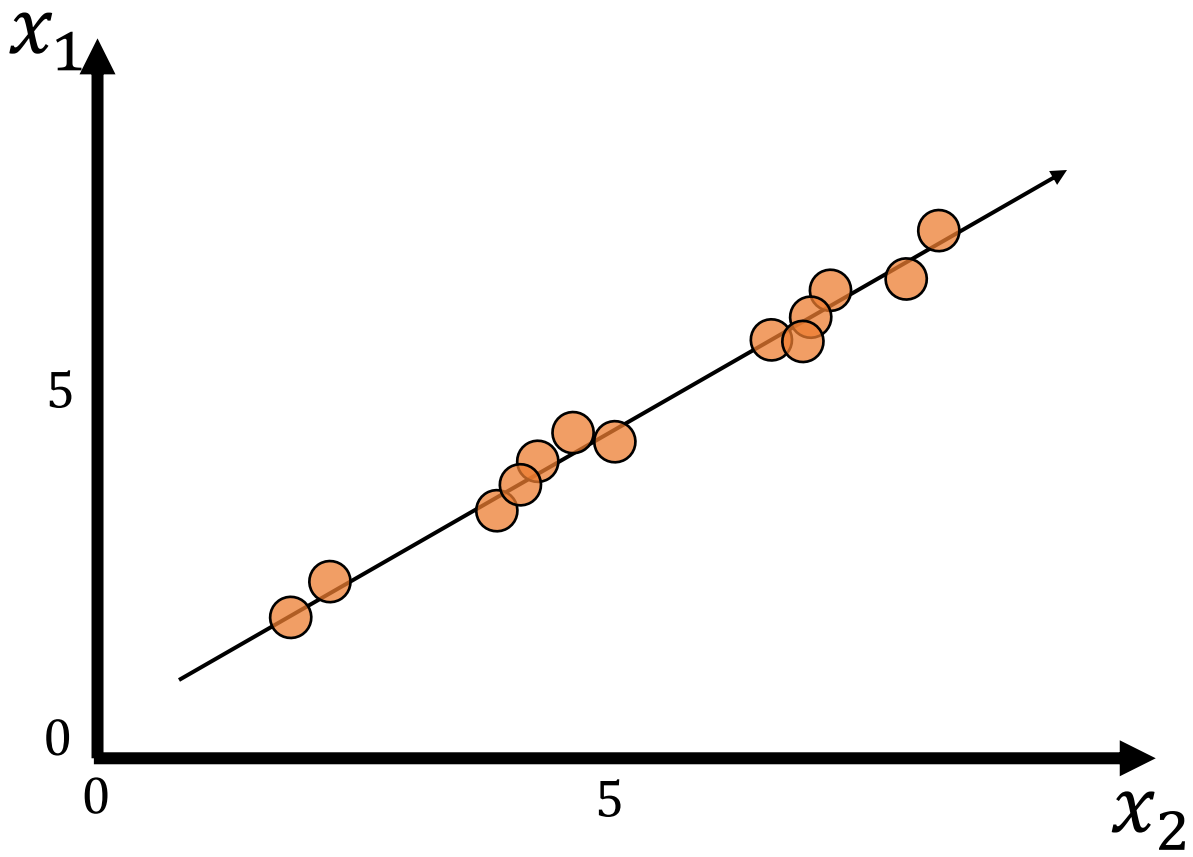




Build Some Intuition About Dimensionality Reduction

Say I came to you and said,

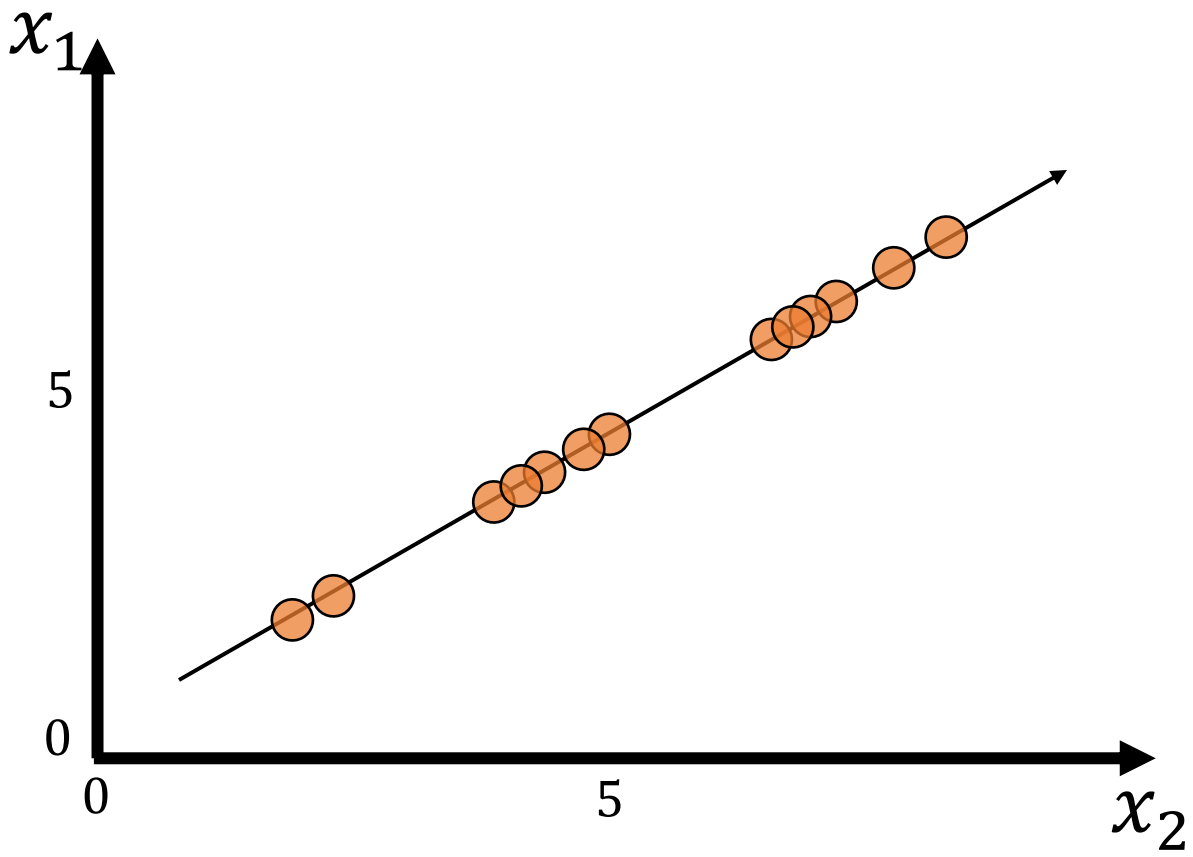
"We don't have enough memory to put both dimensions of our data through our *GIANT MODEL*™, which dimension do you think we can get rid of?"



Build Some Intuition About Dimensionality Reduction

Say I came to you and said,

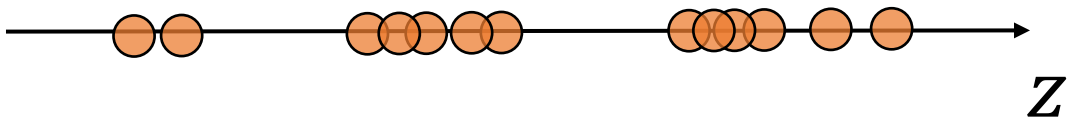
"We don't have enough memory to put both dimensions of our data through our *GIANT MODEL*™, which dimension do you think we can get rid of?"



Build Some Intuition About Dimensionality Reduction

Say I came to you and said,

"We don't have enough memory to put both dimensions of our data through our *GIANT MODEL*™, which dimension do you think we can get rid of?"

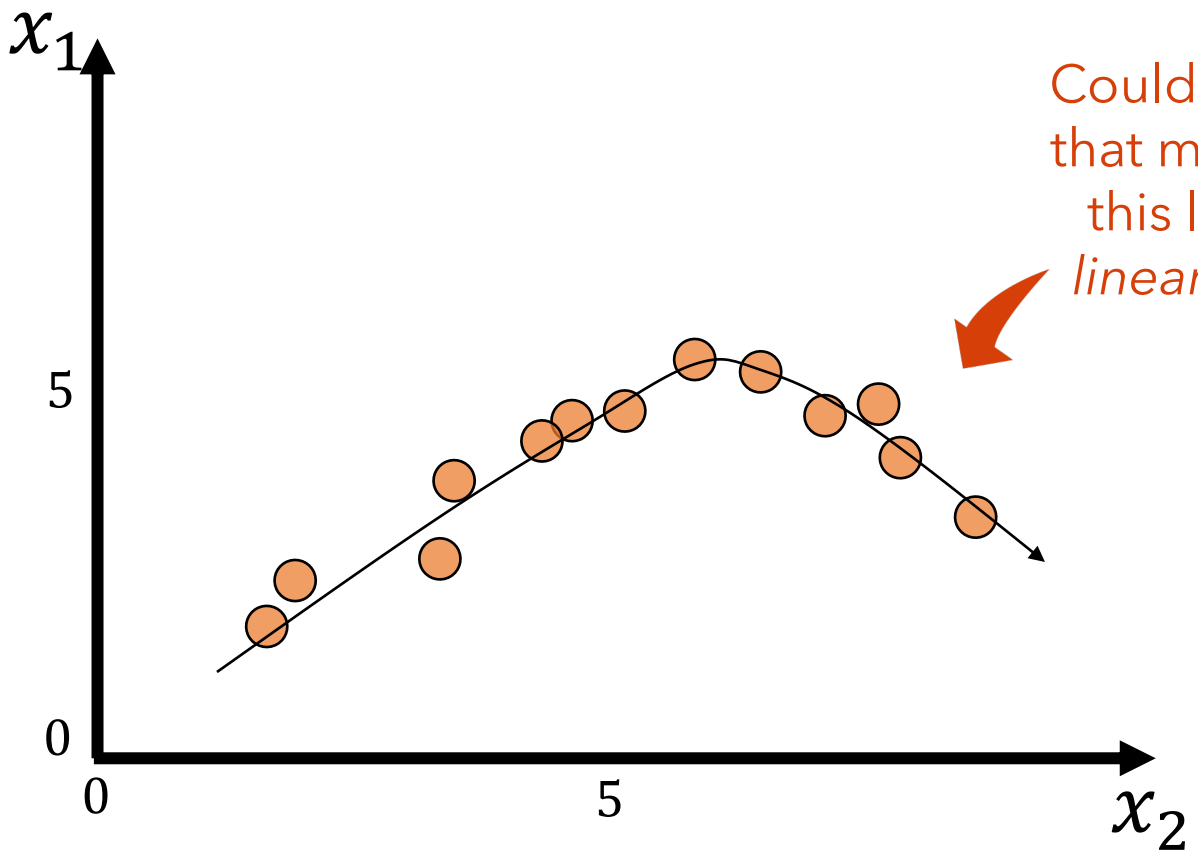


New representation just measures distance along the line we drew

Build Some Intuition About Dimensionality Reduction

Say I came to you and said,

"We don't have enough memory to put both dimensions of our data through our *GIANT MODEL*™, which dimension do you think we can get rid of?"



Could have a new representation that measures the distance along this line – this would be a *non-linear dimensionality reduction.*

An Example Dimensionality Reduction

Each image represented by a 4096-dimensional vector from a Convolutional Neural Network.

A dimensionality reduction algorithm **t-SNE** was applied to reduce to only 2 dimensions.

- A reduction of 99.9%!

Plot on the left shows a set of images organized according to the these 2 dimensions of their CNN representation.

Note that t-SNE is pretty fancy but lots of easy packages to run it (even in Python).

The big version:
https://cs.stanford.edu/people/karpathy/cnnembed/cnn_embed_6k.jpg

Image from Andrej Karpathy at https://cs.stanford.edu/people/karpathy/cnnembed/

Principal Component Analysis (PCA)

A classic dimensionality reduction technique that is widely used

- Finds a **linear** projection from a d-dimensional vector space to a k-dimensional vector space while preserving variation in a dataset (k given):
  - E.g., projecting 4096-dimensional vectors down to 2-dimensional

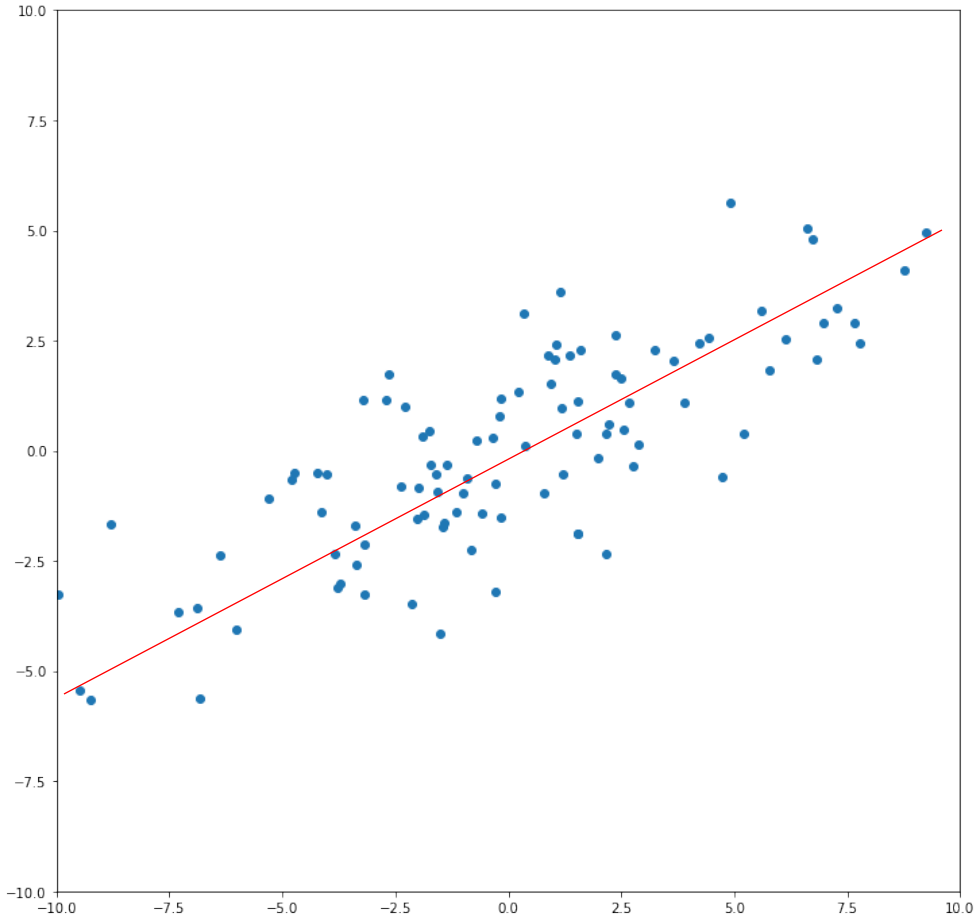
What might it mean to "preserve variation"? What rule were we using before?

- Suppose two features dim-0 and dim-1, but can only keep one:
  - For dim-0, most examples have similar values (small variance)
  - For dim-1, most examples differ (large variance)
- Keep the one with more variance! It explains more about the difference between items

Principal Component Analysis (PCA) -- Conceptually

Consider doing the following:

- Find a line such that most of the variance of the data follows along that line
- Or equivalently, a line where the dataset projected onto that line maintains as much variance.
- Call this line your first principal component.

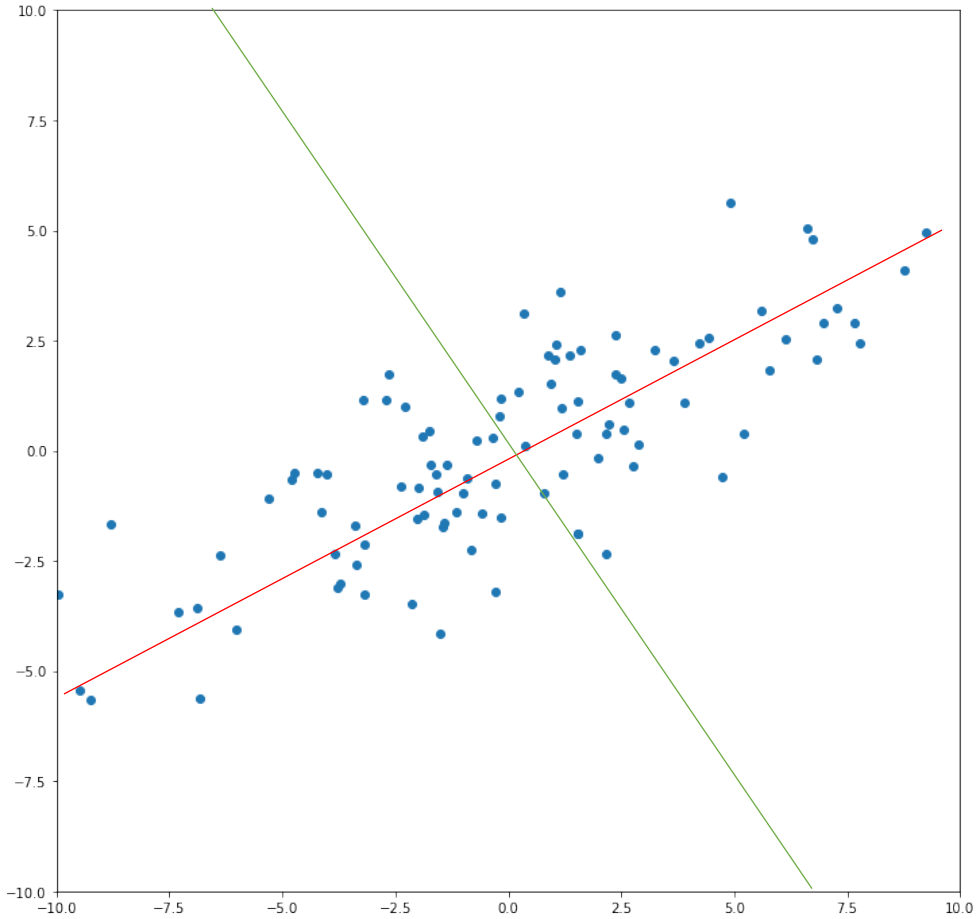


Consider doing the following:

- Repeat this by finding the line orthogonal from this that captures the most variance.
  - 2nd principal component
  - 3rd …
  - 4th …

- In 2D, there is only one line orthogonal to our 1st principal component so we must stop after 2.

- In higher dims, there will be many.



Deriving PCA (The First Line)

Setup: Assume we are given a dataset $X = \{x_i\}_{i=1}^n$ where $x_i \in \mathbb{R}^d$ that is zero-centered (i.e., the mean of our dataset is a zero-vector).

Goal: Let's focus on finding a line defined by the vector w_1 that maximizes the variance of the points in X after each is projected onto it.

How can we express the variance of points in X after projection onto w_1?

$$Var(w_1^T x) = \frac{1}{n}\sum_{i=1}^{n}(w_1^T x_i - w_1^T \bar{x})^2 = \frac{1}{n}\sum_{i=1}^{n}(w_1^T x_i)(w_1^T x_i)$$

Zero-centered

$$= \frac{1}{n}\sum_{i=1}^{n}(w_1^T x_i)(x_i^T w_1)^T = \frac{1}{n}\sum_{i=1}^{n} w_1^T x_i x_i^T w_1$$

$$= w_1^T \left(\frac{1}{n}\sum_{i=1}^{n} x_i x_i^T\right) w_i = w_1^T \, \Sigma_x \, w_1$$

Deriving PCA (The First Line)

$$Var(w_1^T x) = w_1^T \Sigma_x w_1$$

So, it seems like we can find the line w_1 that maximizes the variance of our data after projection by finding:

$$w_1 = \underset{w}{\text{argmax}} \; w^T \Sigma_x w$$

where $\Sigma_x = \frac{1}{n} \sum_{i=1}^{n} x_i x_i^T$ is the covariance of our data.

However, this optimization has some trivial solutions. Large entries in w_1 could make any bit of variation in the data seem huge. (For mathy folks, note that the covariance matrix is positive semi-definite so large w can't result in negative values)

Let's add a constraint that w_1 has to have some fixed L2 norm – say it needs to be a unit vector:

$$w_1 = \underset{w}{\text{argmax}} \; w^T \Sigma_x w$$
$$s.t. \quad w^T w = 1$$

Deriving PCA (The First Line)

$$Var(w_1^T x) = w_1^T \Sigma_x w_1 \qquad w_1 = \underset{w}{\text{argmax}} \; w^T \Sigma_x w \quad s.t. \quad w^T w = 1$$

Let's bring our old friend for this constrained optimization problem and write the Lagrangian by adding a Lagrange multiplier λ_1 and moving our constraint up:

$$L(w, \lambda) = w_1^T \Sigma_x w_1 - \lambda_1 (w_1^T w_1 - 1)$$

Setting the derivative of this equal to zero we find a system of equations defining all possible critical points of our function (minima/maxima/saddle points):

$$\frac{dL(w, \lambda)}{dw} = \Sigma_x w - \lambda_1 w = 0 \qquad \Rightarrow \qquad \Sigma_x w = \lambda_1 w$$

Those of you with a linear algebra background might recognize this as the definition of an eigenvector of Σ_x! For those of you who don't, it is a well-studied system of equations that often has multiple solutions. By solution, I mean a vector w and scalar λ_1 where the above equation holds. These are called Eigenvectors and Eigenvalues respectively.

Deriving PCA (The First Line)

$$Var(w_1^T x) = w_1^T \Sigma_x w_1 \qquad w_1 = \underset{w}{\text{argmax}} \; w^T \Sigma_x w \;\; s.t. \;\; w^T w = 1 \qquad \frac{dL(w,\lambda)}{dw} = \Sigma_x w - \lambda_1 w = 0 \Rightarrow \Sigma_x w = \lambda_1 w$$

If there are many possible solutions, which do we pick to maximize our original problem? Let's go back and consider the original problem. Maximizing variance means:

$$w = \underset{w}{\text{argmax}} \; w^T \Sigma_x w \quad s.t. \quad w^T w = 1$$

For Eigenvector w and Eigenvalue λ_1 of Σ_x

$$w = \underset{w_i}{\text{argmax}} \; w^T \lambda_i w \quad s.t. \quad w^T w = 1$$

By the constraint

$$w = \underset{w_i}{\text{argmax}} \; \lambda_i \quad s.t. \quad w^T w = 1$$

This show two interesting things.

- To maximize the variance, we should take the Eigenvector of the covariance matrix Σ_x with the largest Eigenvalue, and
- That the corresponding eigenvalue will the variance captured after projection.

What have we just done?

We wanted to find a line to project our data onto that would preserve as much variance as possible. So to find that linear transform we:

1. Wrote down mathematically what we meant by a linear projection $(w^T x)$
2. Derived an expression for the variance of the data *after* projection $(Var(w^T x) = w^T \Sigma_x w)$
3. Set up an optimization problem to find the projection that maximizes the variance. But then noticed it had trivial solutions, so constrained the representation of the line (must have $w^T w = 1$)
4. Applied the Lagrange Multiplier method $(L(w, \lambda) = w_1^T \Sigma_x w_1 - \lambda_1 (w_1^T w_1 - 1))$
5. Took the derivative and set equal to zero to find all critical points (w's that potentially could maximize our variance). This related to Eigenvectors from linear algebra and can have multiple solutions $(\Sigma_x w = \lambda w)$
6. Observed that the linear transform w that maximizes the variance will be the Eigenvector of Σ_x with the largest Eigenvalue λ. This gives us the first dimension of our PCA!

Plenty of computational packages can find Eigenvectors of matrices.

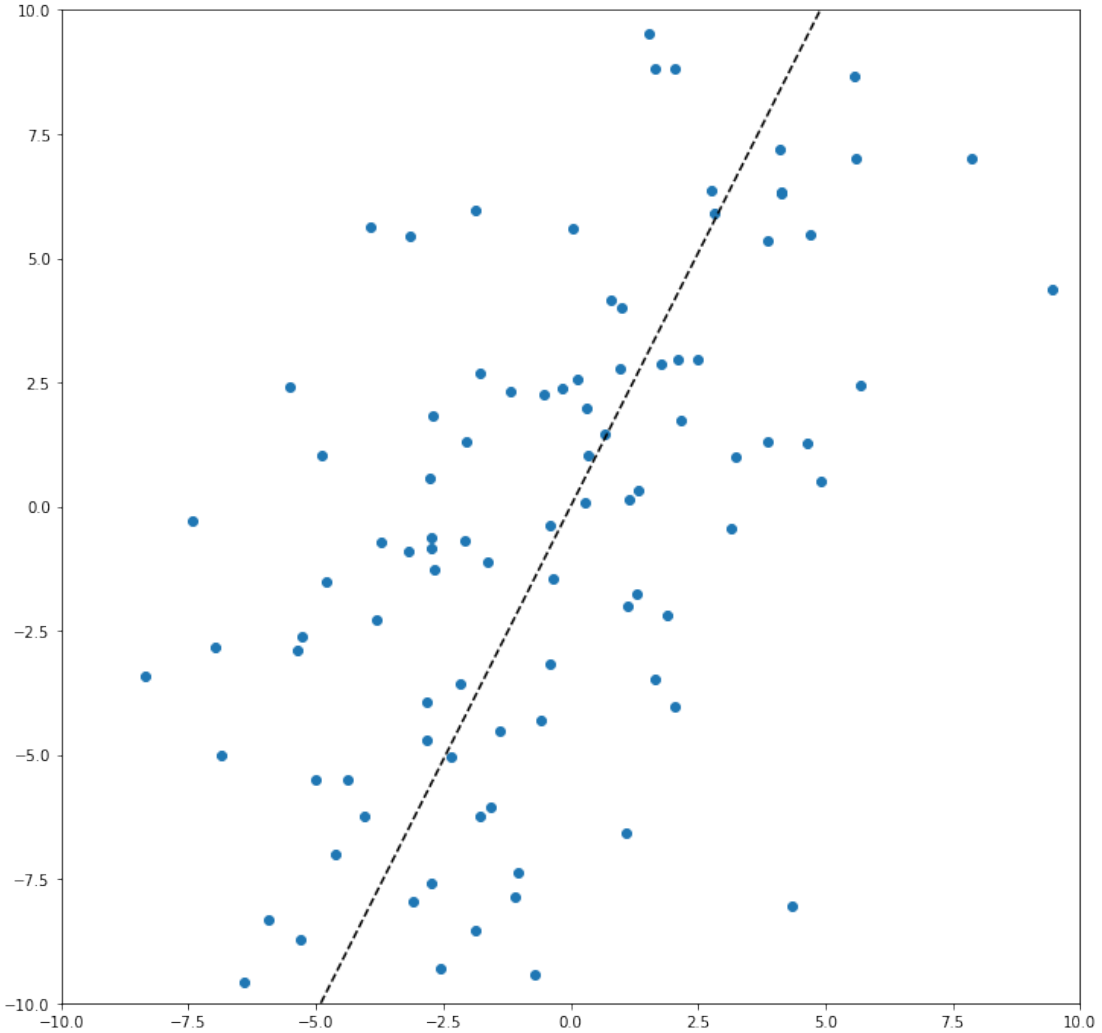
Wrapping up the derivation

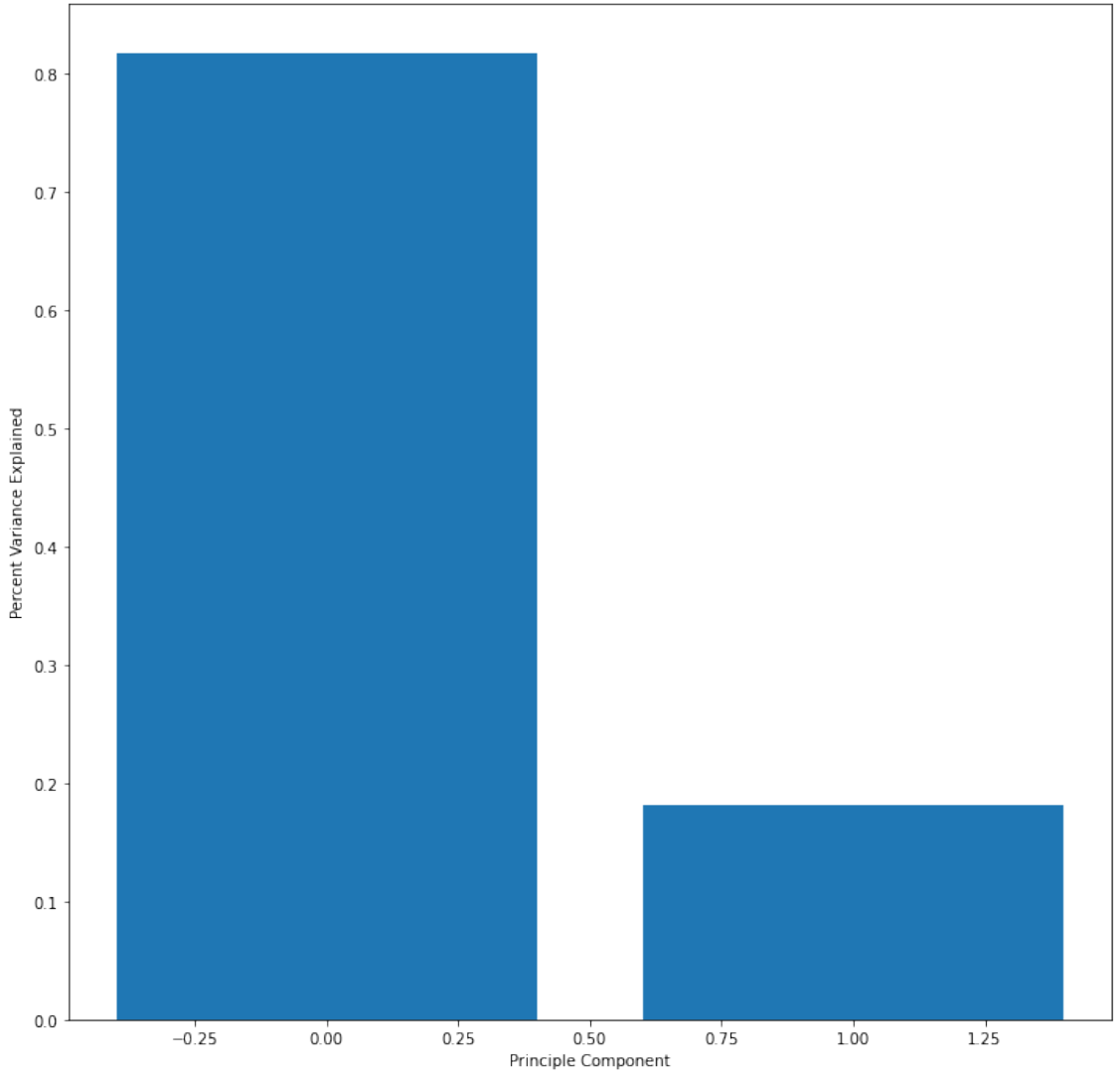
That was just for the 1st principal component, how do we find the other k-1?

Just sort the Eigenvalues in descending order and take the first k as the k principal components.

An Example

```
# X is an n x d data matrix
X = X - np.mean(X,axis=0)
cov = X.T @ X / n
eigvals, eigvecs = np.linalg.eig(cov)
```





Procedural View of PCA

Say you have data that is 100 dimensional and you want it down to 3.

```
# X is an n x d data matrix
X = X - np.mean(X,axis=0)
cov = X.T @ X / n
eigvals, eigvecs = np.linalg.eig(cov)
```

To perform PCA on your data matrix X (n x 100):

1. Compute the mean vector of your data (100 dim vector) and subtract it from X to center your date

2. Compute the covariance matrix by computing $\frac{1}{\mathrm{n}} X^T X$

3. Call a package to compute the Eigenvalues/Eigenvectors of the covariance.

4. Sort both in descending order by the Eigenvalues and return Eigenvectors corresponding to the top k. Usually as a (d x k) matrix W with each column being one Eigenvector.

To project your data to that lower dimension, simply compute the matrix product XW

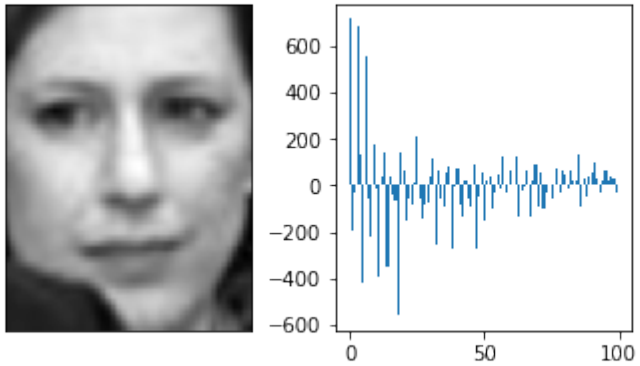
After this class is over: Just call a PCA API, they do basically the same thing.

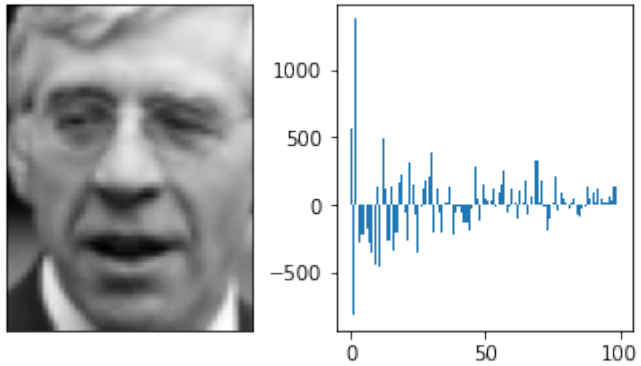
Classic Example – Eigenfaces

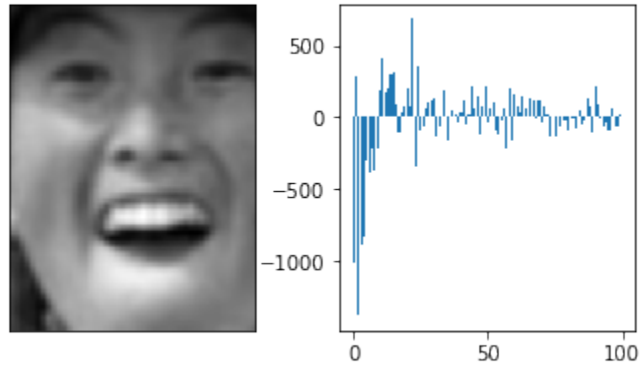
Went back to my face dataset. Original images were 62*47=2914 dimensions. Ran PCA to bring it down to 100, the principal components are shown below. Recall this is just the Eigenvectors with highest Eigenvalues, I've just rearranged them back to being images from vectors.

Classic Example – Eigenfaces

Projecting faces into this space turns them into 100 dimensional vectors.

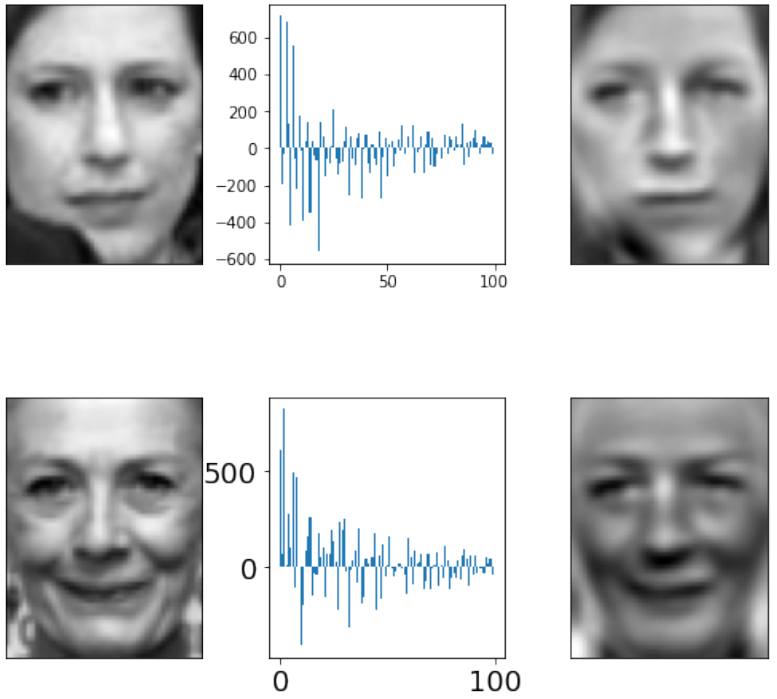






Classic Example – Eigenfaces

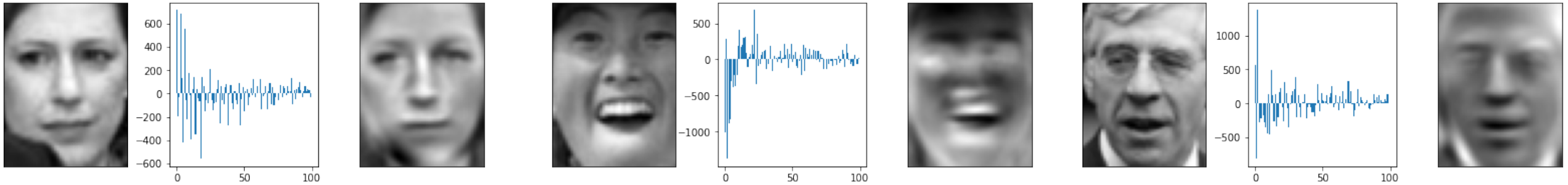
Can “back project” those feature vectors to images by taking a linear combination of the principal components



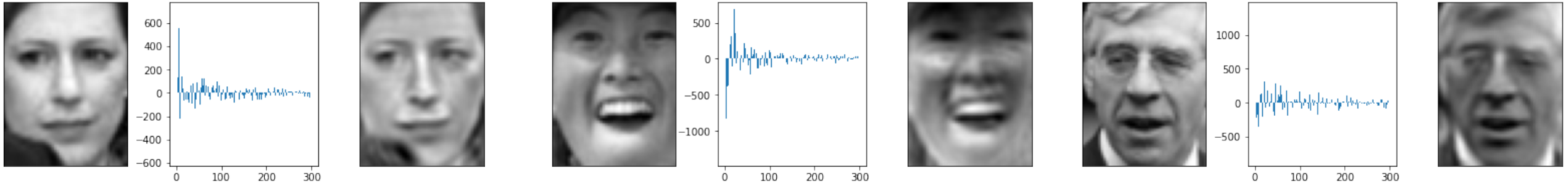
Classic Example – Eigenfaces

Can increase the dimension to 300 principal components, still 10x smaller than the initial image.

PCA with 100 Components Backprojected



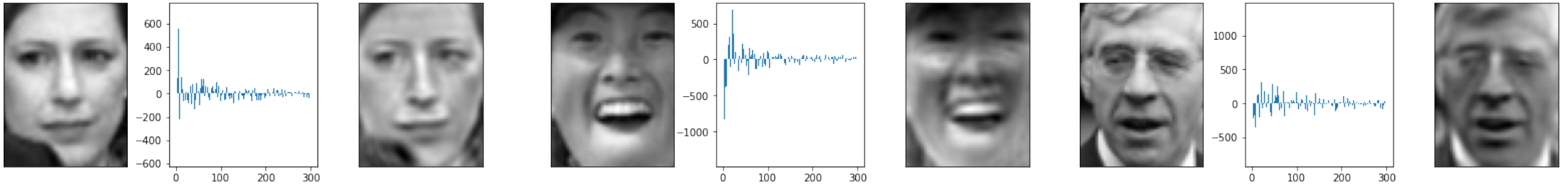
PCA with 300 Components Backprojected



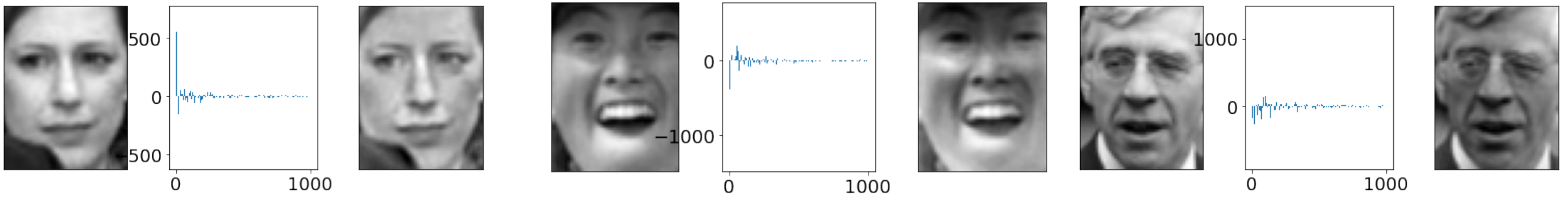
Classic Example – Eigenfaces

Can increase the dimension to 1000 principal components, still 3x smaller than the initial image.

PCA with 300 Components Backprojected



PCA with 1000 Components Backprojected



Classic Example – Eigenfaces

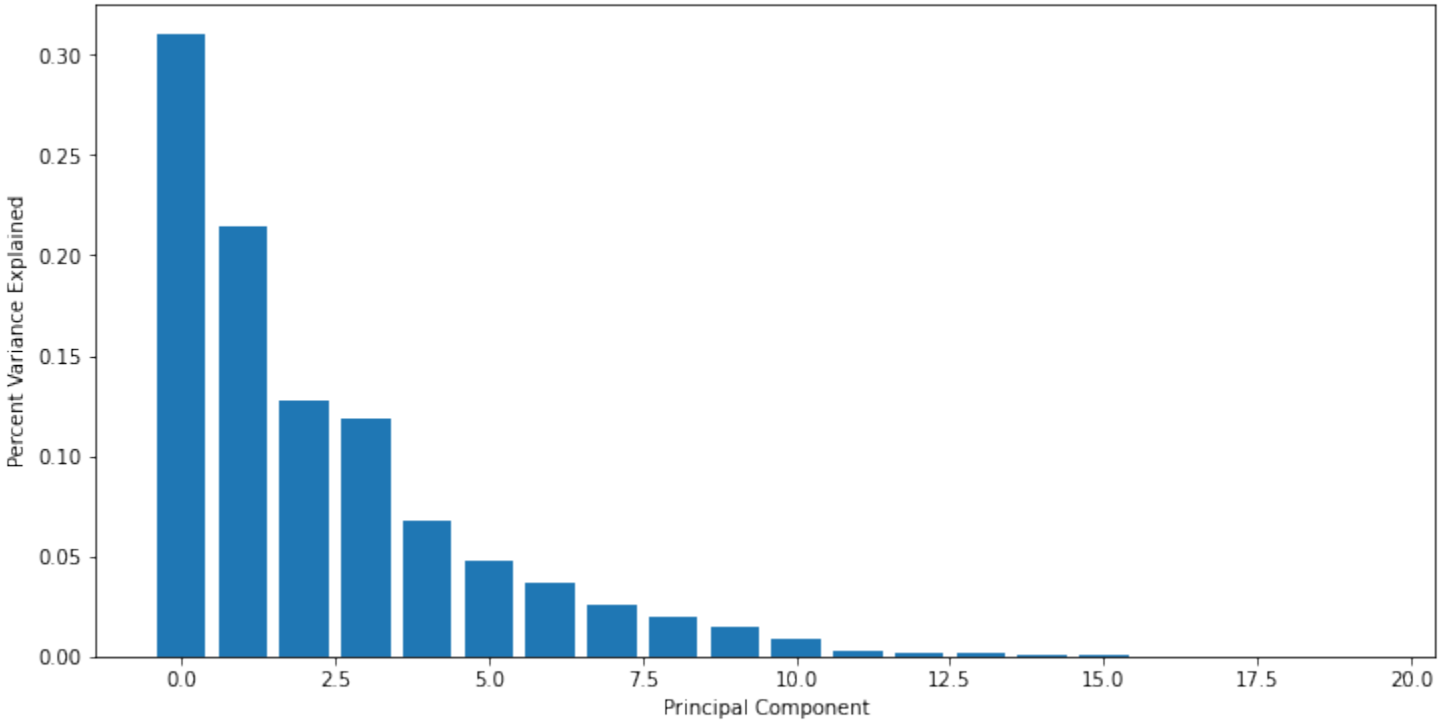
Took that 300 dimensional representation and ran k-means on it. Very sensitive to pixel intensity.

Choosing How Far To Reduce Dimensionality

For some tasks like visualization, the dimensionality you are projecting to is fixed.

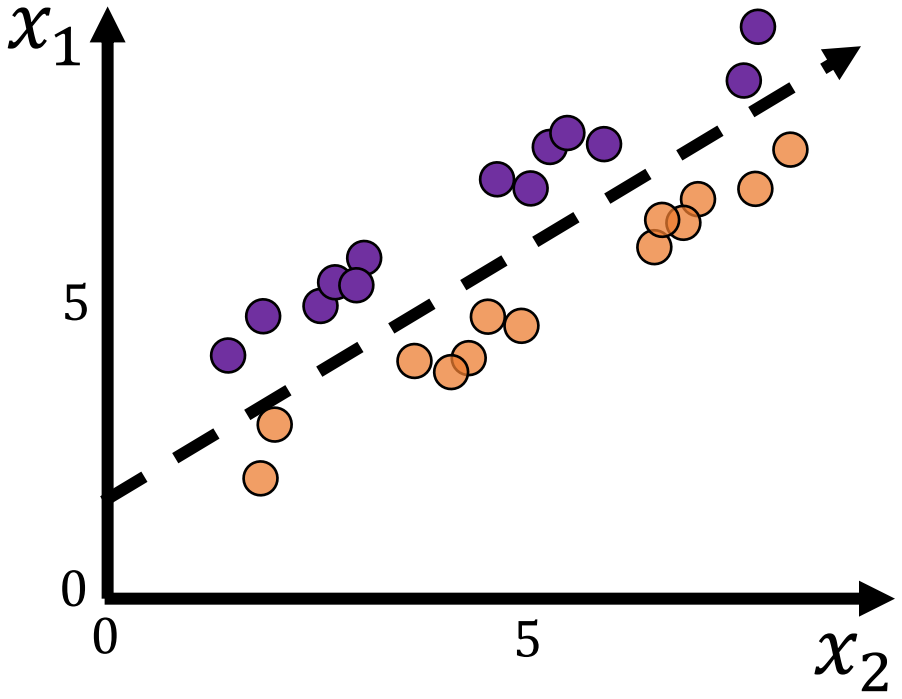
- Use fraction of variance explained to judge how representative the visualization is

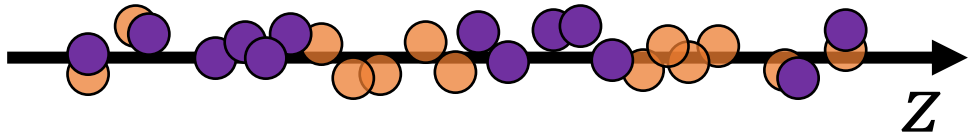
If reducing dimension for other reasons, can look at plot of Eigenvalues / Sum Eigenvalues to judge what fraction of variance has been captured for a certain dimensionality:



PCA Doesn't Care About Labels

Direction of maximum variance may not be useful for classification:

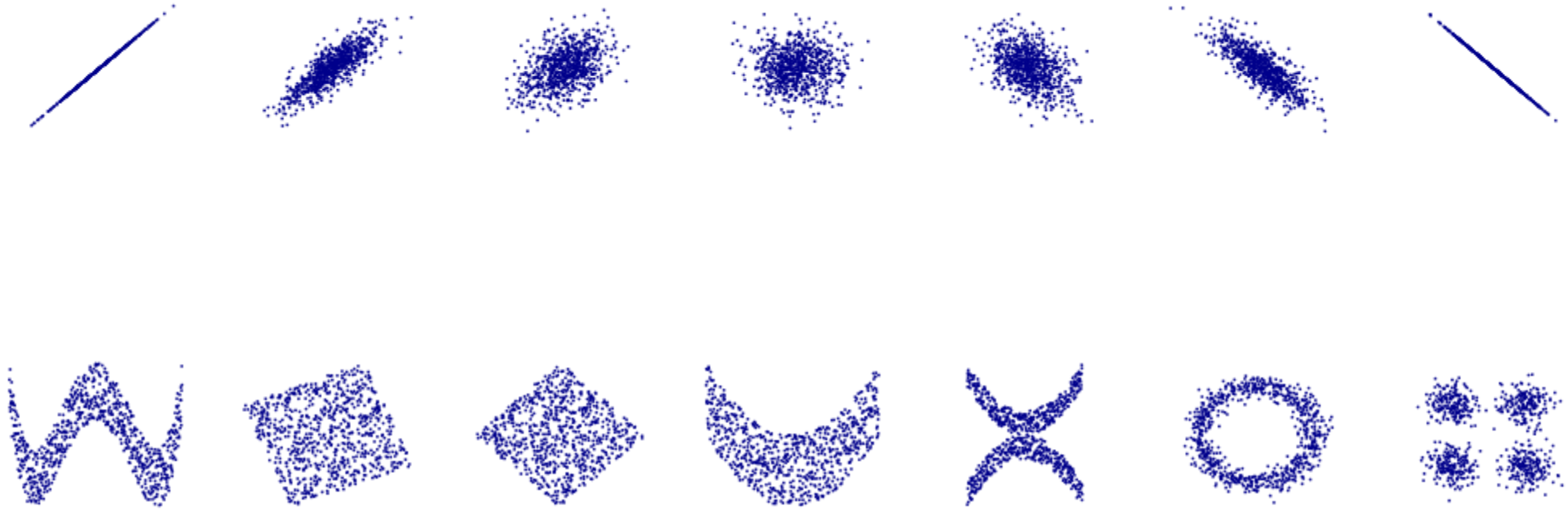




See Linear Discriminant Analysis (LDA) if you have labels and want to do linear dimensionality reduction that tries to account for this.

PCA is a Linear Dimensionality Reduction Model

Only works at finding linear subspace…



Today's Learning Objectives

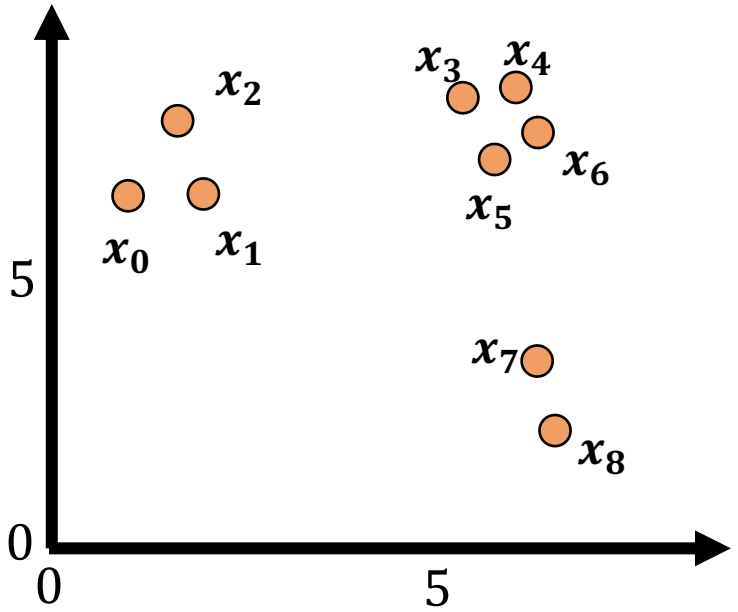
Be able to answer:

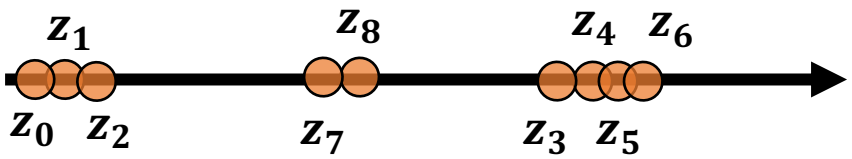
- What is Hierarchical Agglomerative Clustering (HAC)?
  - What are different linking strategies?
- How can we evaluate clustering?
- What is dimensionality reduction?
  - What is principal component analysis (PCA) and how does it work?
  - What are stochastic neighbor embeddings (SNE) and how do they work?

Non-Linear Dimensionality Reduction: t-SNE

t-Distributed Stochastic Neighbor Embedding (t-SNE)

Idea: For each **d-dimensional** input vector $\boldsymbol{x_i}$, directly optimize a **k-dimensional** vector z_i such that spatial relationships between your high dimensional datapoints are preserved between the lower dimensional ones.





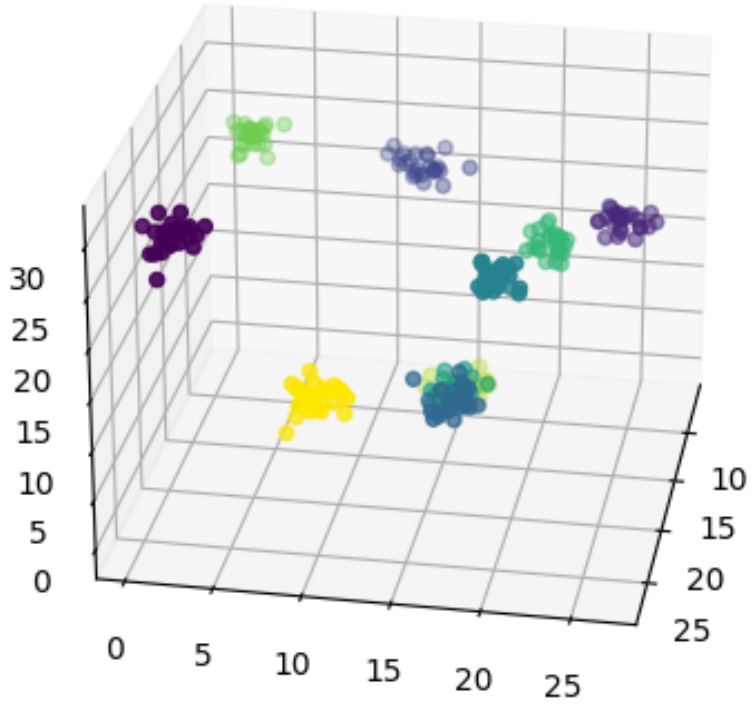
Important Note: We aren't learning a function $\boldsymbol{z_i} = f(\boldsymbol{x_i})$, we are learning the z vectors directly!

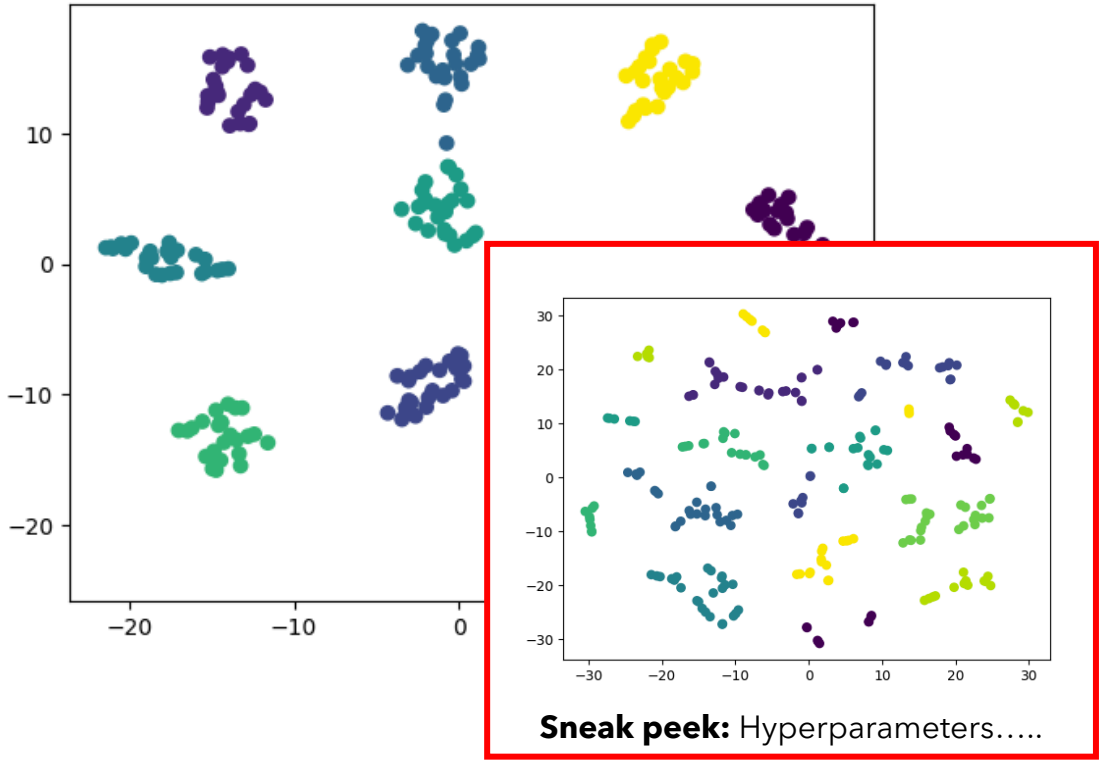
Non-Linear Dimensionality Reduction: t-SNE

t-Distributed Stochastic Neighbor Embedding (t-SNE)

Another example:

- 10 random clusters of points in 3D (colored by which cluster)
- Ran t-SNE to go to 2D (kept the coloring)





Non-Linear Dimensionality Reduction: t-SNE

t-Distributed Stochastic Neighbor Embedding (t-SNE)

Idea: For each **d-dimensional** input vector $\boldsymbol{x_i}$, directly optimize a **k-dimensional** vector z_i such that **spatial relationships between your high dimensional datapoints are preserved between the lower dimensional ones.**

How to operationalize this thought?

One path: The "neighborness" between points should stay the same.

We can start here with **Stochastic Neighbor Embeddings.**

Stochastic Neighbor Embedding (SNE)

How to capture the structure of the data?

Let's consider a "stochastic" notion of nearest neighbor where point **j** is selected as the nearest neighbor of point **i** with probability that decreases with distance:

Smaller distance, higher probability

$$p_{i|j} = \frac{e^{-\frac{\|x_i - x_j\|^2}{2\sigma_i^2}}}{\sum_{k \neq i} e^{-\frac{\|x_i - x_k\|^2}{2\sigma_i^2}}}$$

Sigma defines how "random" our nearest neighbor notion is.

Small sigma → almost all probability on closest neighbor.

Normalized over all possible neighbors

This distribution describes which points are close to point **i**, but:

- Doesn't capture specific distances. Only relative to each other.
- Only cares about "near" points. How many depends on sigma in a smooth way.

Stochastic Neighbor Embedding (SNE)

How to learn low-dimensional vectors that keep this structure around?

Compute the same sort of distribution using our learned vectors $\boldsymbol{z}$:

$$q_{i|j} = \frac{e^{-\frac{\|z_i - z_j\|^2}{2\sigma_i^2}}}{\sum_{k \neq i} e^{-\frac{\|z_i - z_k\|^2}{2\sigma_i^2}}}$$

Same as we computed for the high-dimensional vectors but based on distances between our low-dimensional z vectors.

If we can find vectors $\boldsymbol{z_i}$ that make the distribution $\boldsymbol{q_{i|*}}$ match $\boldsymbol{p_{i|*}}$ then we would have the same structure in this neighborly sense.

Stochastic Neighbor Embedding (SNE)

Use KL-Divergence to measure differences between these distributions:

$$\boldsymbol{KL(p_i \,||\, q_i) = \sum_j p_{i|j} log_2 \left(\frac{p_{i|j}}{q_{i|j}} \right)}$$

Let our loss be the sum over all these distributions:

$$\boldsymbol{loss = \sum_i KL(p_i \,||\, q_i) = \sum_i \sum_j p_{i|j} log_2 \left(\frac{p_{i|j}}{q_{i|j}} \right)}$$

Optimization time! How do we find low-dimensional vectors $z_0, z_1, \ldots, z_n$ that minimize this loss.

Stochastic Neighbor Embedding (SNE)

Recall that our loss is a function of the low-dimensional vectors $z_0, z_1, \ldots, z_n$ because we compute the q distribution using their relative distances:

$$\boldsymbol{loss}(\boldsymbol{z_0, \ldots, z_n}) = \sum_i \sum_j \boldsymbol{p_{i|j}} \boldsymbol{log_2} \left(\frac{\boldsymbol{p_{i|j}}}{\boldsymbol{q_{i|j}(z_0, \ldots, z_n)}} \right)$$

Take the derivative with respect to the low-dimensional vectors and perform gradient descent:

$$\frac{\boldsymbol{\delta loss}}{\boldsymbol{\delta z_i}} = \boldsymbol{2} \sum_j \boldsymbol{(z_j - z_i)(p_{i|j} - q_{i|j} + p_{j|i} - q_{j|i})}$$

(Taken from the 2002 paper "Stochastic Neighbor Embeddings" without derivation)

t-Distributed Stochastic Neighbor Embedding (t-SNE)

t-SNE runs on similar ideas but adds a few tricks to make things better:

- Uses a Student-t distribution for computing $q_{i|j}$

$$q_{ij} = \frac{\left(1 + \|z_i - z_j\|^2\right)^{-1}}{\sum_{k \neq i}(1 + \|z_i - z_k\|^2)^{-1}}$$

  Has "heavier" tails than the Gaussian-ish ones used in SNE. Let's points be further away in the lower-dimensional space without paying big costs.

- Compute p distributions using both directions:

$$p_{ij} = \frac{p_{i|j} + p_{j|i}}{2n}$$

t-Distributed Stochastic Neighbor Embedding (t-SNE)

Hyperpamameters:

- The number of lower dimensions **k**
- The sigma's for each datapoint:
  - No good way to pick a global sigma for all points to use
    - Potentially different density of points in different regions of the input space
  - t-SNE introduces a "**perplexity**" hyperparameter to implicitly control sigma's
    - Bigger perplexity leads to bigger sigma's and more neighbors considered.
    - Each point's sigma is independently optimized to match the perplexity.
- Distance functions can also be changed

$$p_{i|j} = \frac{e^{-\frac{\|x_i - x_j\|^2}{2\sigma_i^2}}}{\sum_{k \neq i} e^{-\frac{\|x_i - x_k\|^2}{2\sigma_i^2}}}$$

t-Distributed Stochastic Neighbor Embedding (t-SNE)

I implemented this from scratch based on the 2008 paper only.

- https://colab.research.google.com/drive/1JpEd8TXekrDBiuDf5rYWyg2VVNHWVnnM?usp=sharing
- Not a very polished implementation….

Very good article discussing the nuance of using t-SNE

- https://distill.pub/2016/misread-tsne/
- Some take-aways:
  - The value of perplexity matters a ton
  - Density of points in high dimension does not map to density in low dimension
  - Distance between clusters may not be meaningful
  - Random noise may look meaningful

t-Distributed Stochastic Neighbor Embedding (t-SNE)

I implemented this from scratch based on the 2008 paper only.

- https://colab.research.google.com/drive/1JpEd8TXekrDBiuDf5rYWyg2VVNHWVnnM?usp=sharing
- Not a very polished implementation…., but we can see a few things!

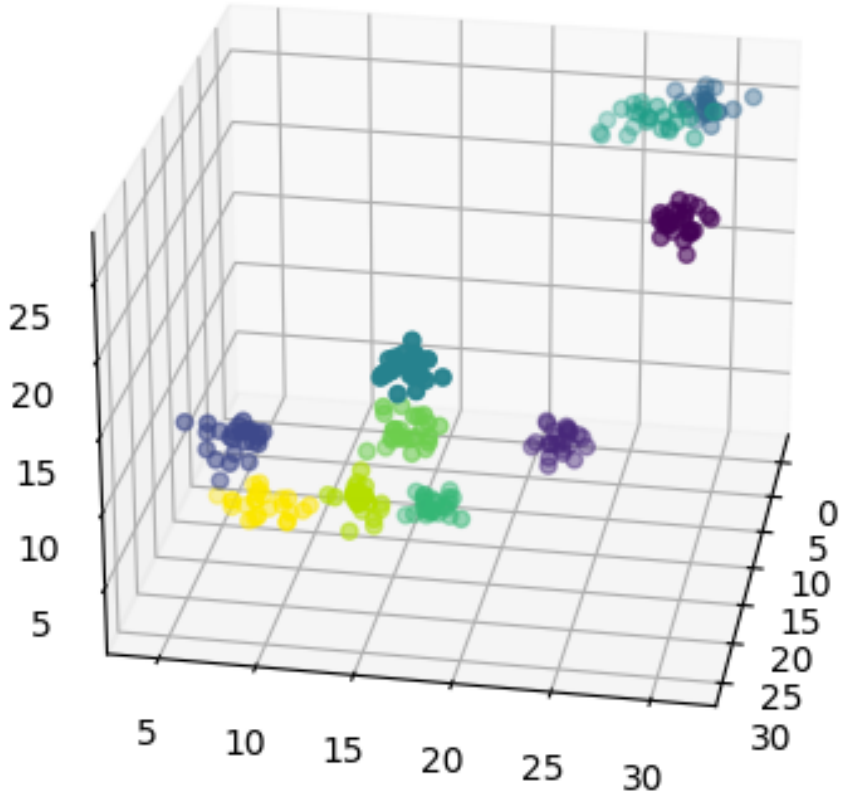
Very good article discussing the nuance of using t-SNE

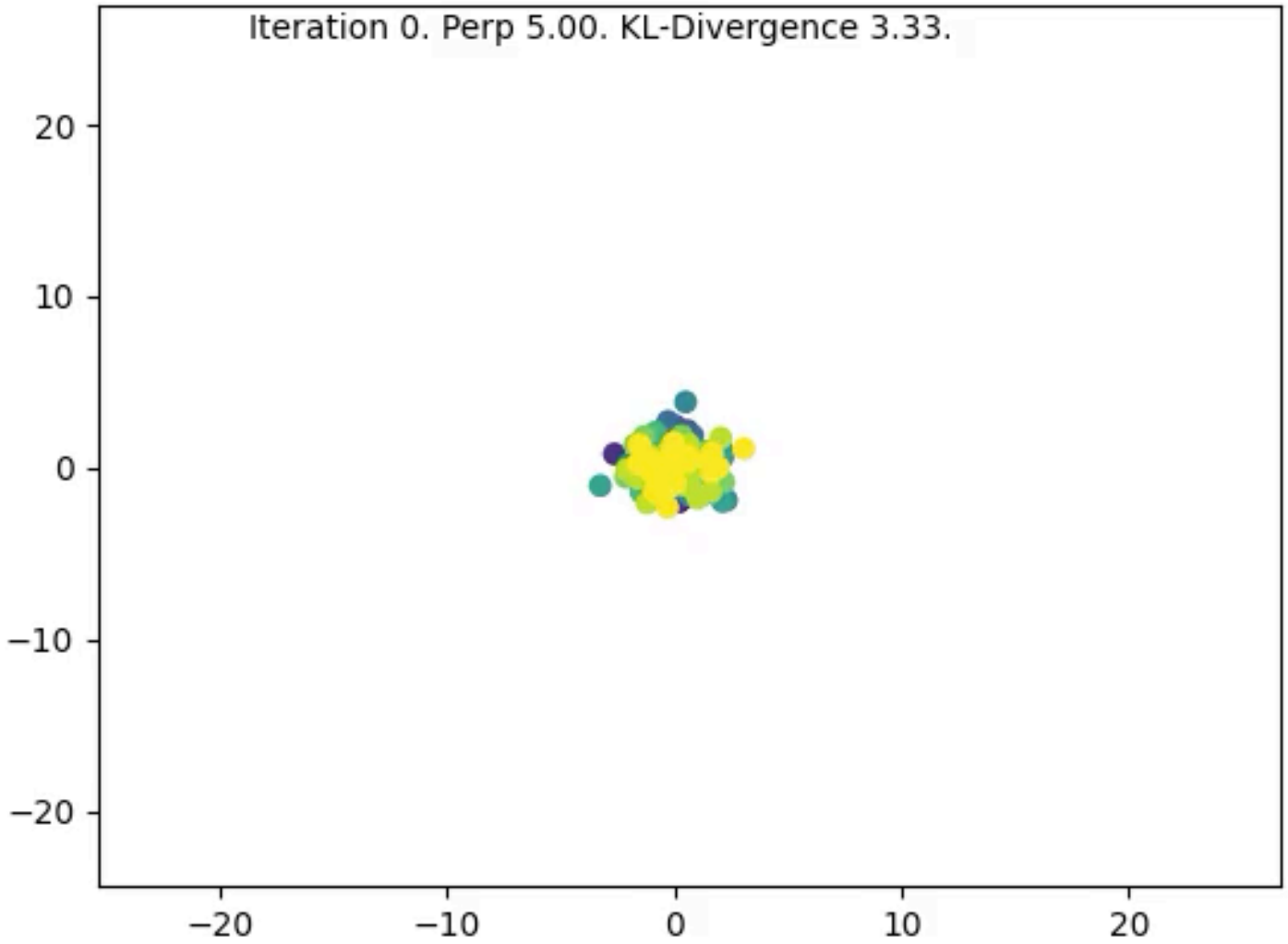
- https://distill.pub/2016/misread-tsne/

t-Distributed Stochastic Neighbor Embedding (t-SNE)

Back to my implementation to get a sense for things:

- 10 random clusters of points in 3D (colored by which cluster)
- Ran t-SNE to go to 2D (kept the coloring)

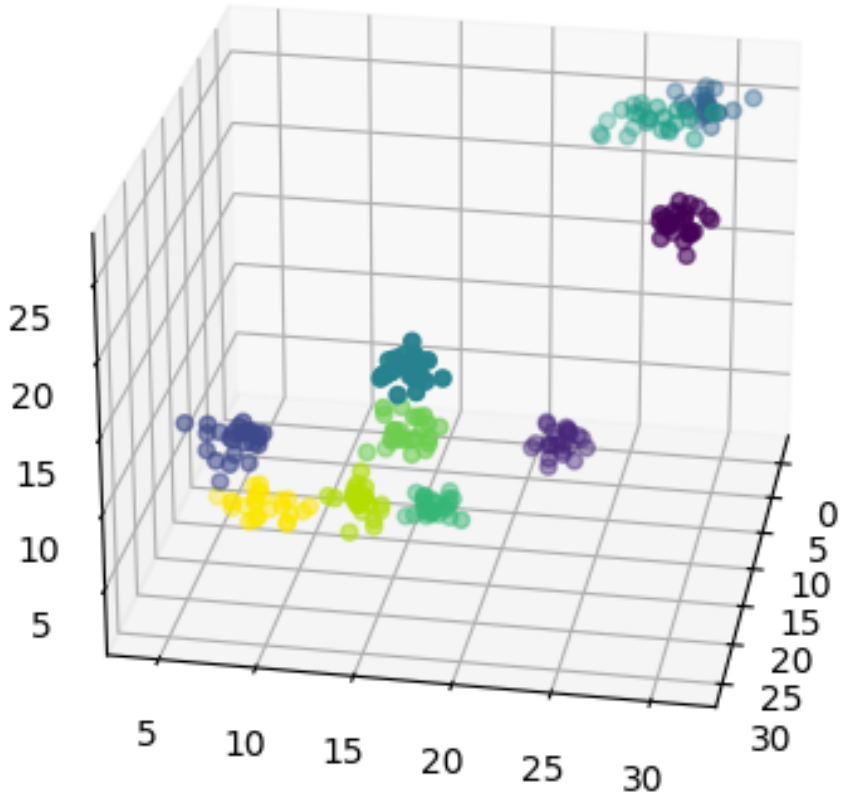


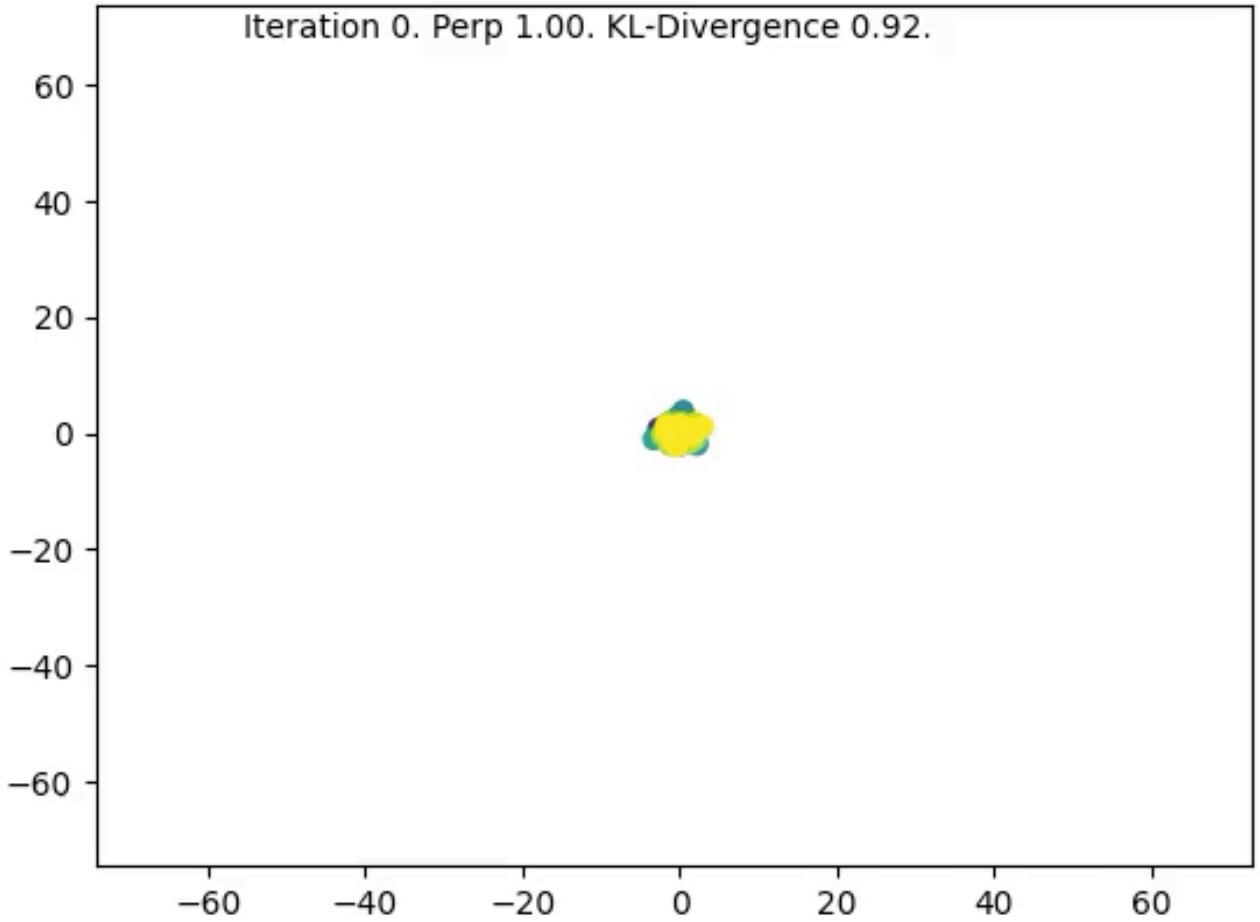


t-Distributed Stochastic Neighbor Embedding (t-SNE)

Back to my implementation to get a sense for things:

- 10 random clusters of points in 3D (colored by which cluster)
- Ran t-SNE to go to 2D (kept the coloring)

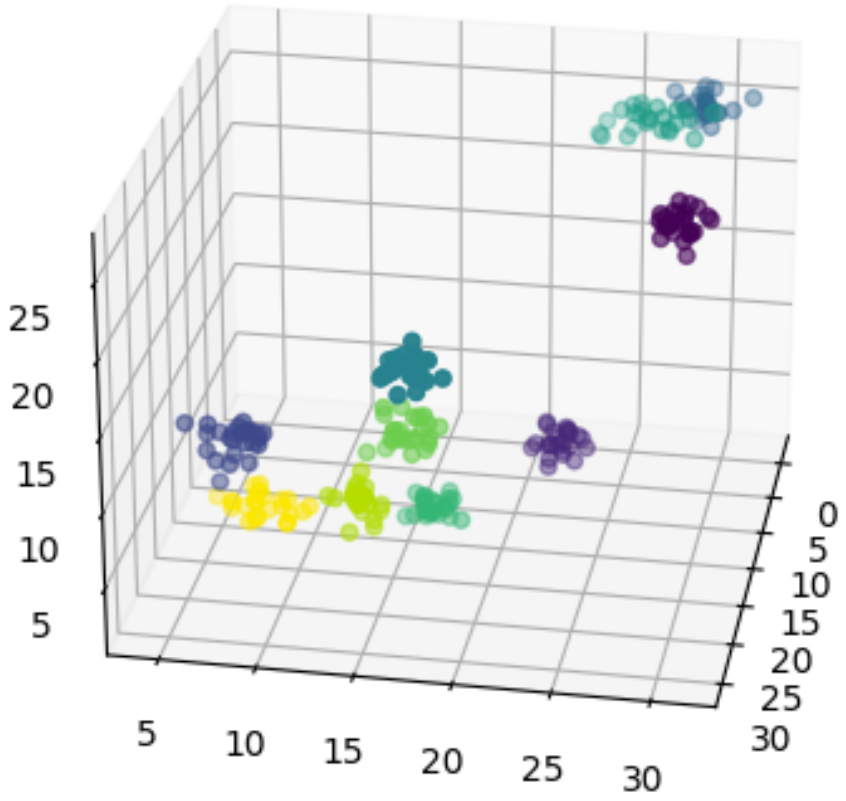


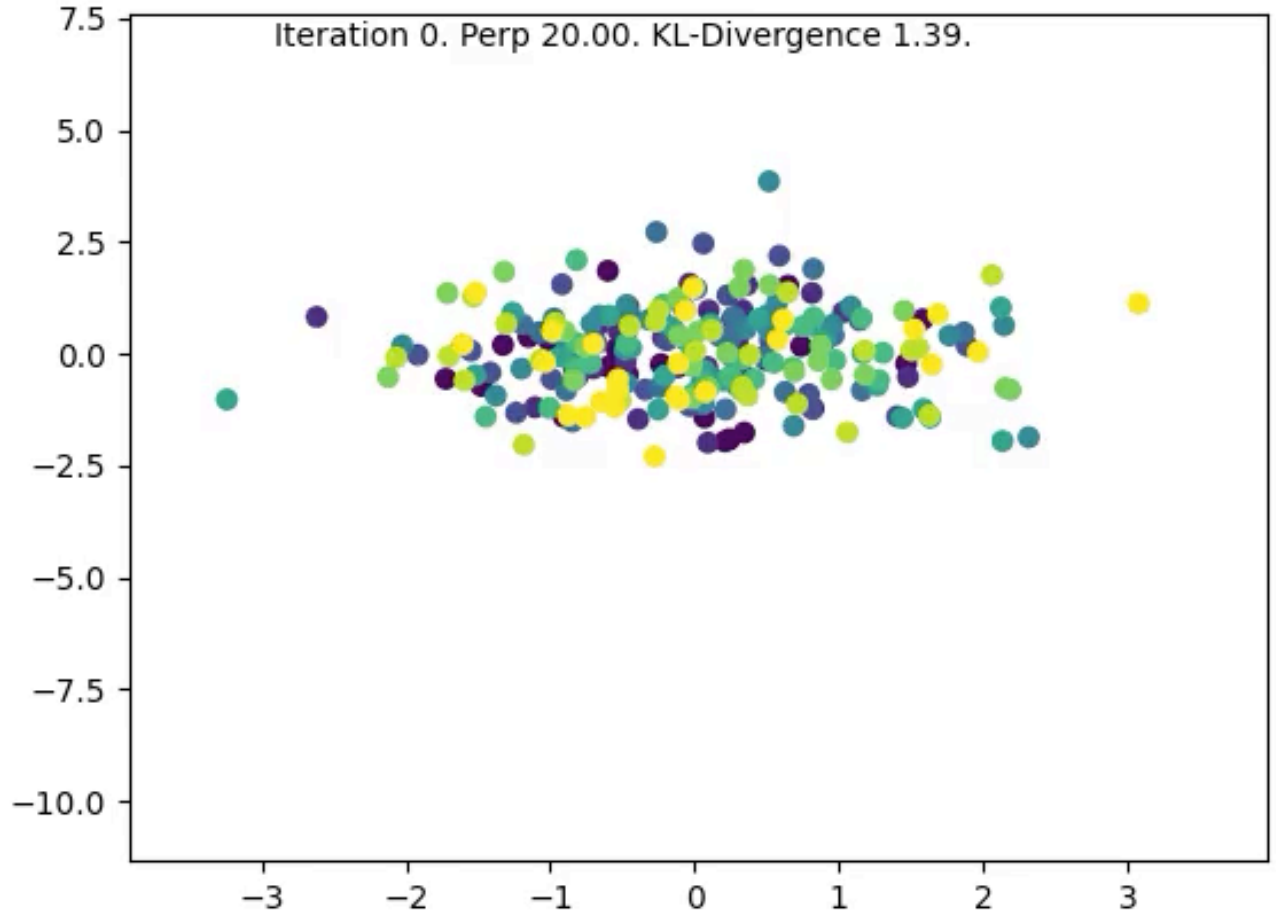


t-Distributed Stochastic Neighbor Embedding (t-SNE)

Back to my implementation to get a sense for things:

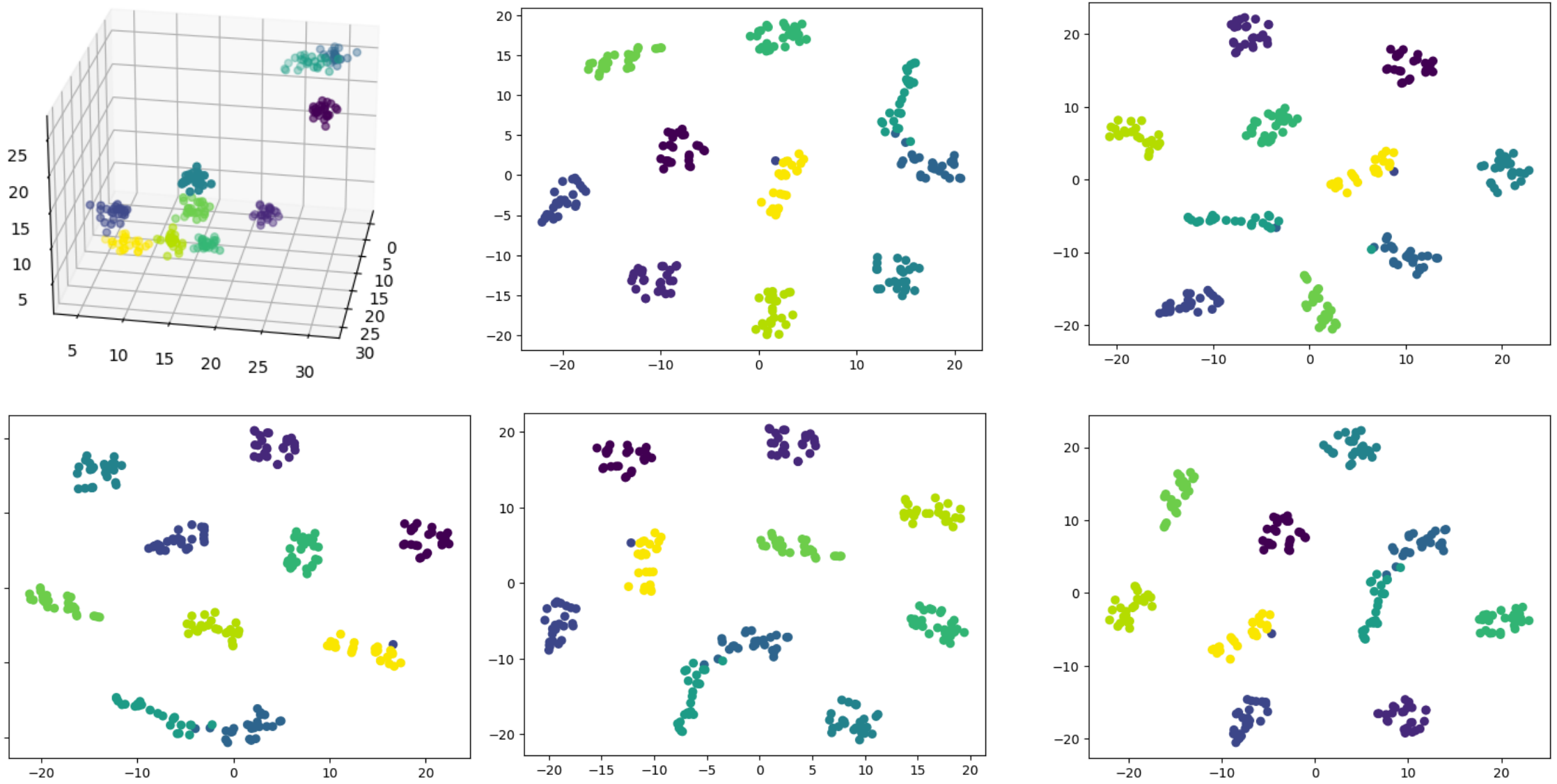
- 10 random clusters of points in 3D (colored by which cluster)
- Ran t-SNE to go to 2D (kept the coloring)





t-Distributed Stochastic Neighbor Embedding (t-SNE)

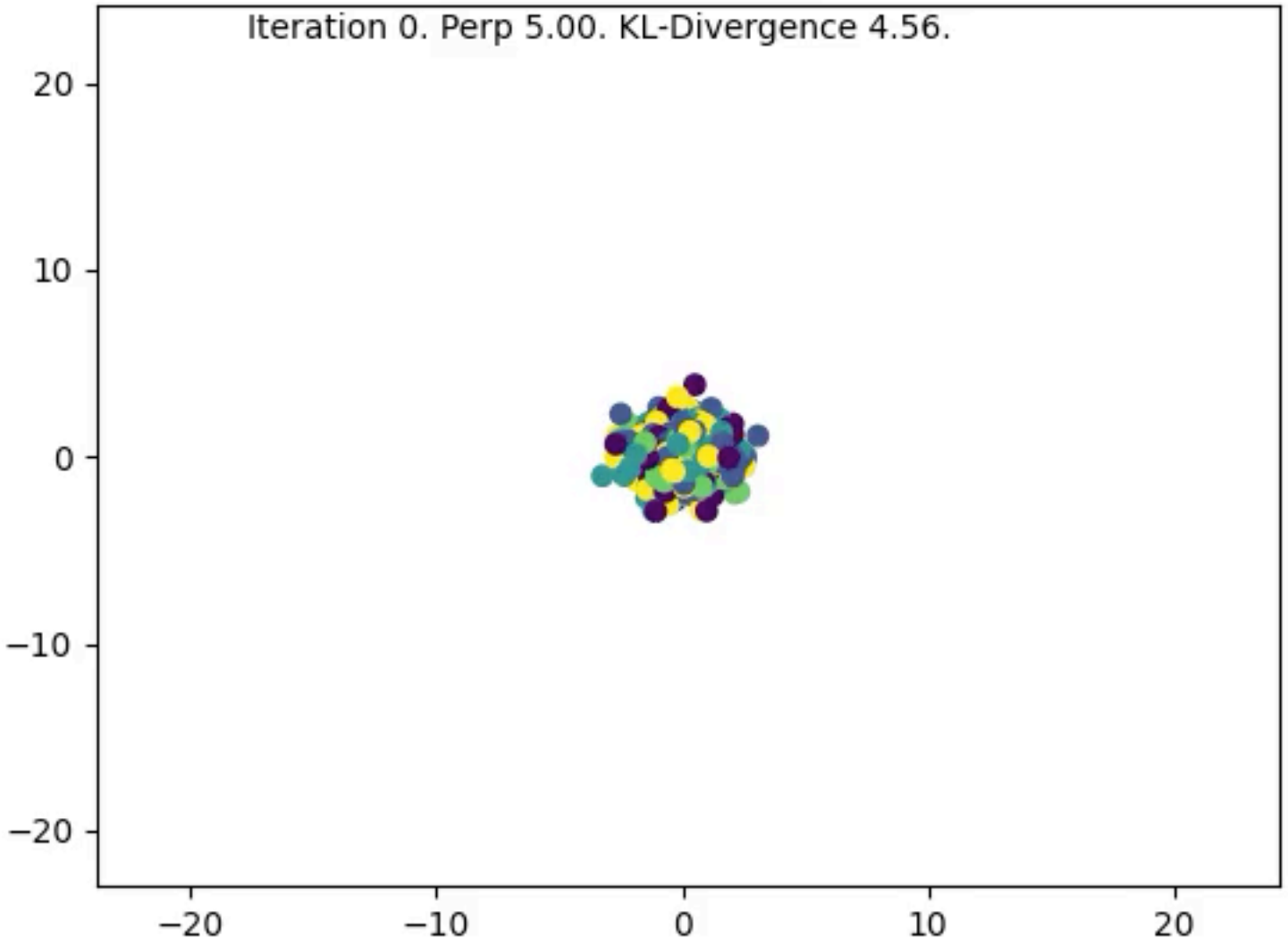
Changing random seeds:



Perplexity = 5

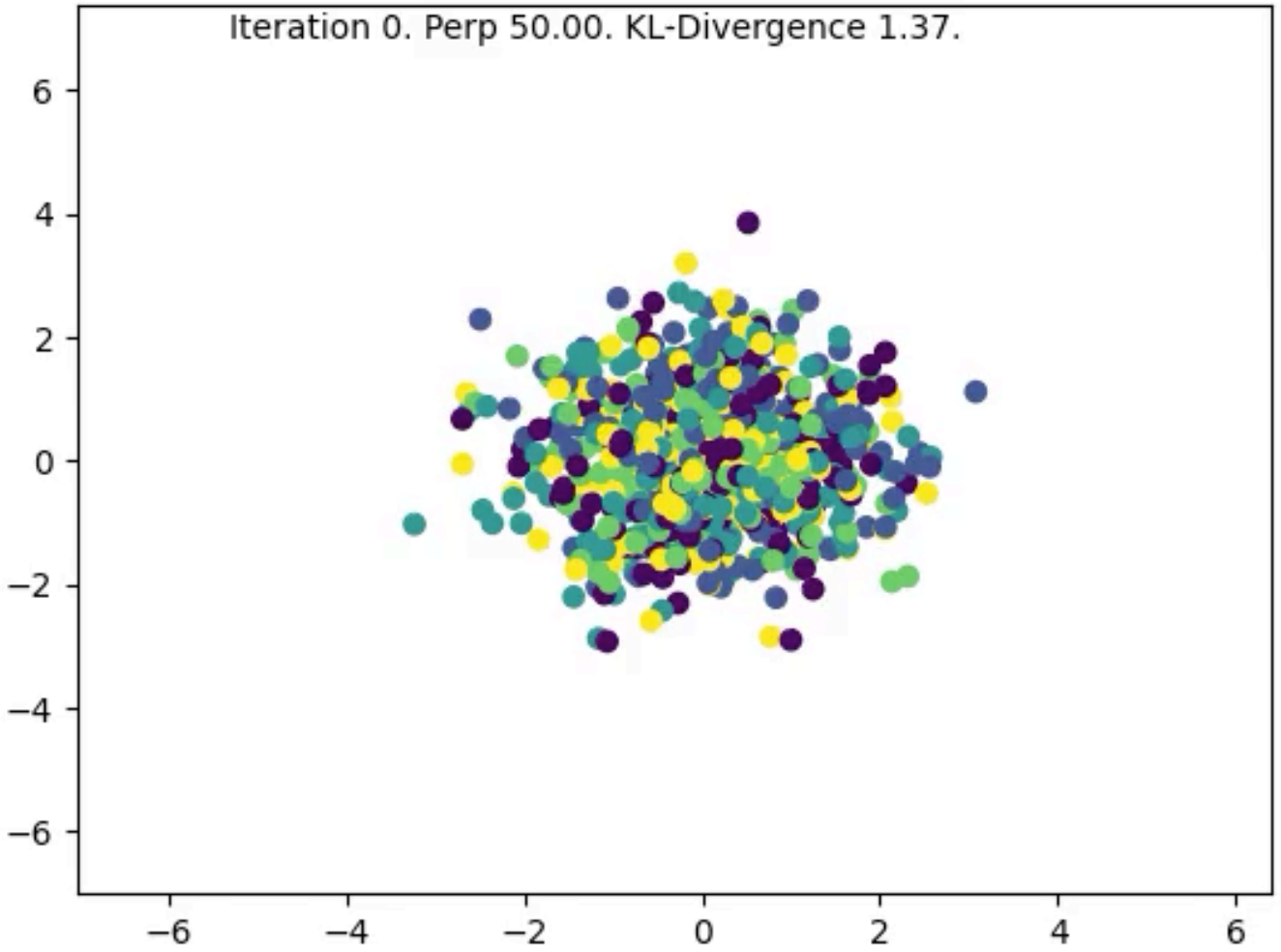
t-Distributed Stochastic Neighbor Embedding (t-SNE)

A more impressive example: MNIST digits (8x8 images in this case and only using 0-4). Colored by class.



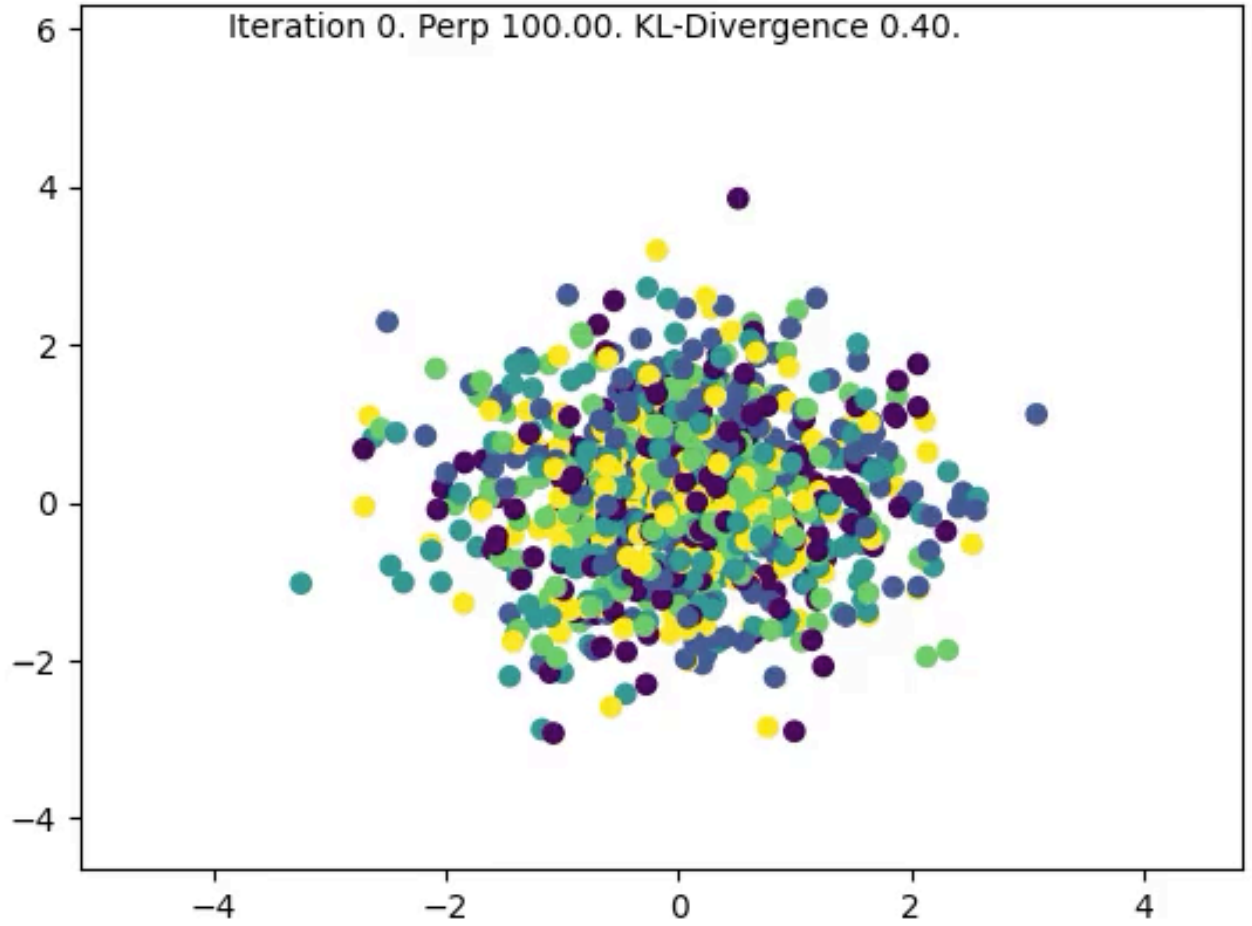
t-Distributed Stochastic Neighbor Embedding (t-SNE)

A more impressive example: MNIST digits (8x8 images in this case and only using 0-4). Colored by class.



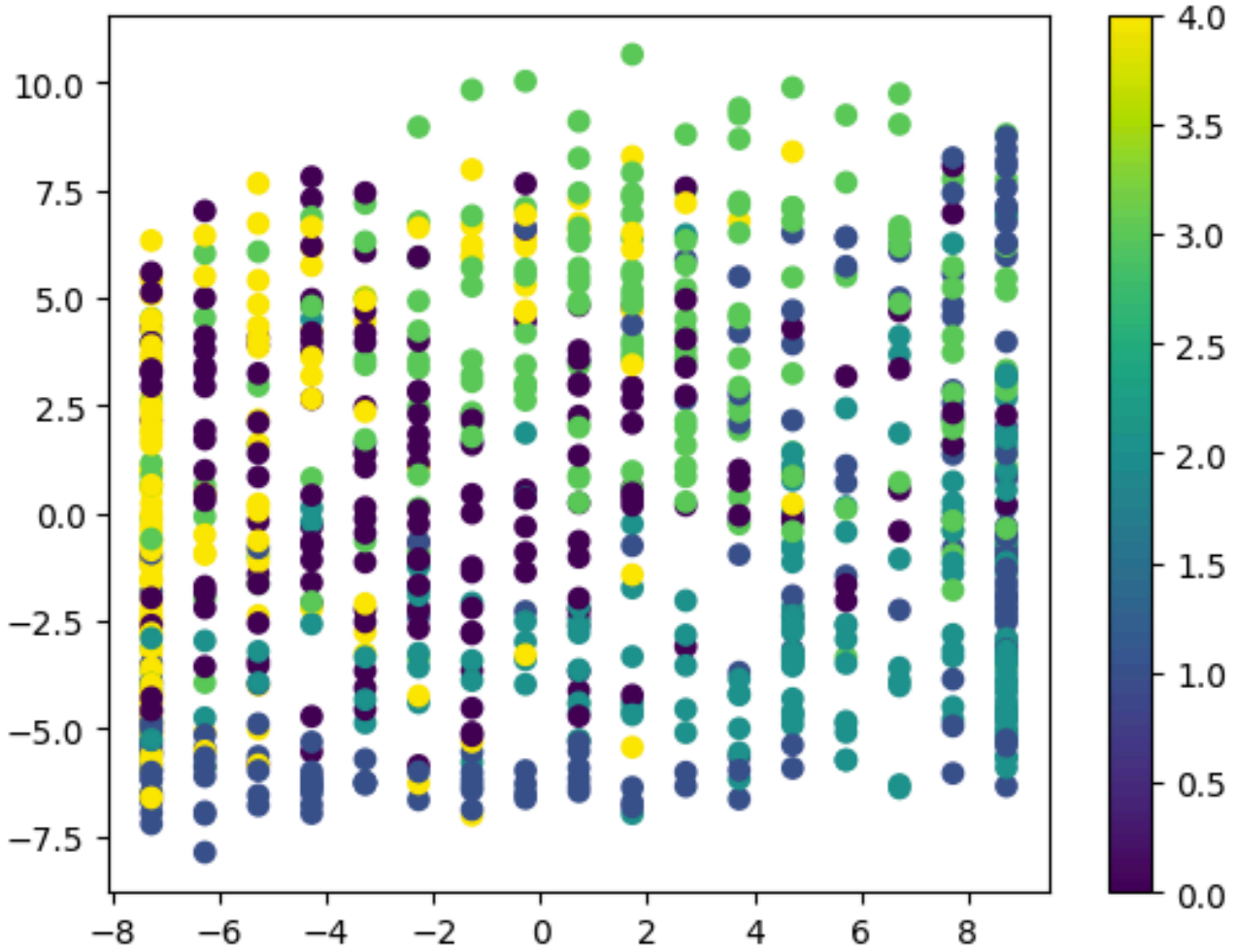
t-Distributed Stochastic Neighbor Embedding (t-SNE)

A more impressive example: MNIST digits (8x8 images in this case and only using 0-4). Colored by class.



Repeating this PCA

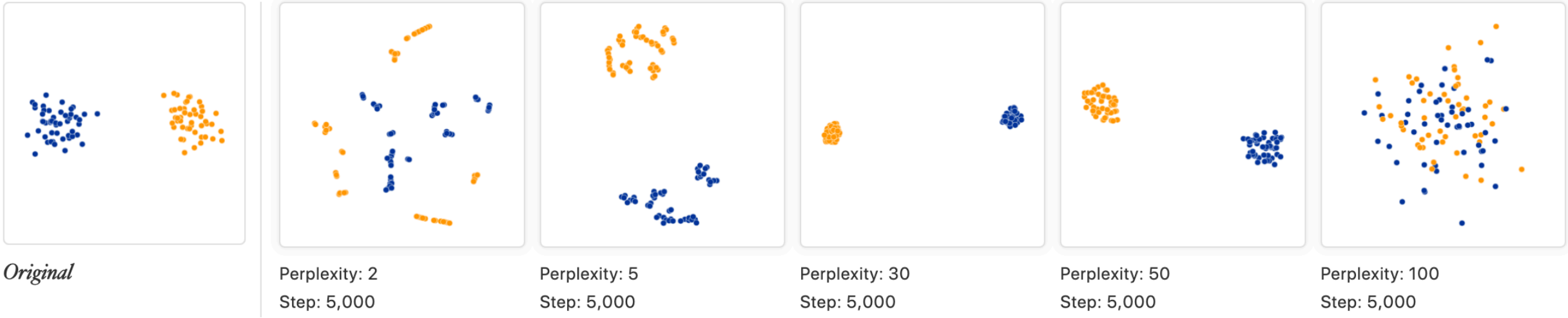
A more impressive example: MNIST digits (8x8 images in this case and only using 0-4). Colored by class.



t-Distributed Stochastic Neighbor Embedding (t-SNE)

Some comments on using t-SNE (taken from https://distill.pub/2016/misread-tsne/):

Hyperparameters really matter.



Original

Perplexity: 2
Step: 5,000

Perplexity: 5
Step: 5,000

Perplexity: 30
Step: 5,000

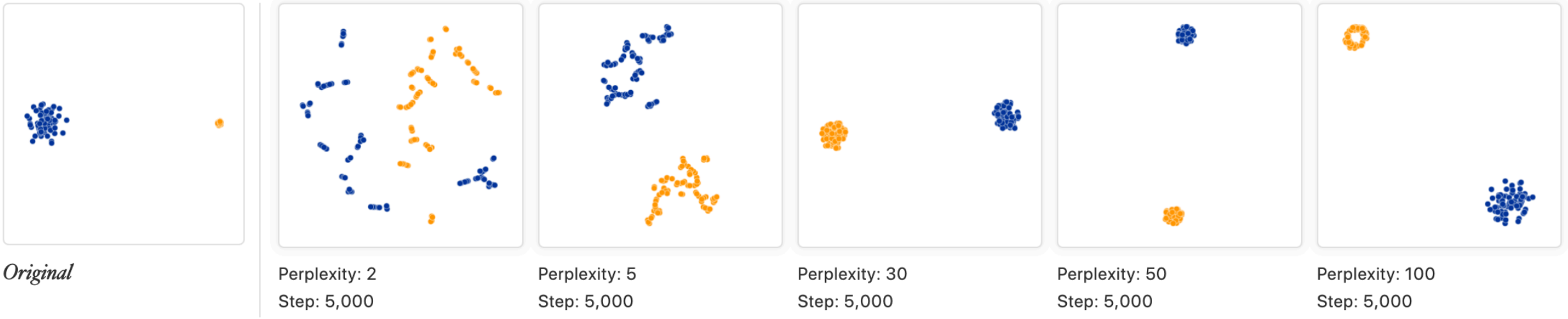
Perplexity: 50
Step: 5,000

Perplexity: 100
Step: 5,000

t-Distributed Stochastic Neighbor Embedding (t-SNE)

Some comments on using t-SNE (taken from https://distill.pub/2016/misread-tsne/):

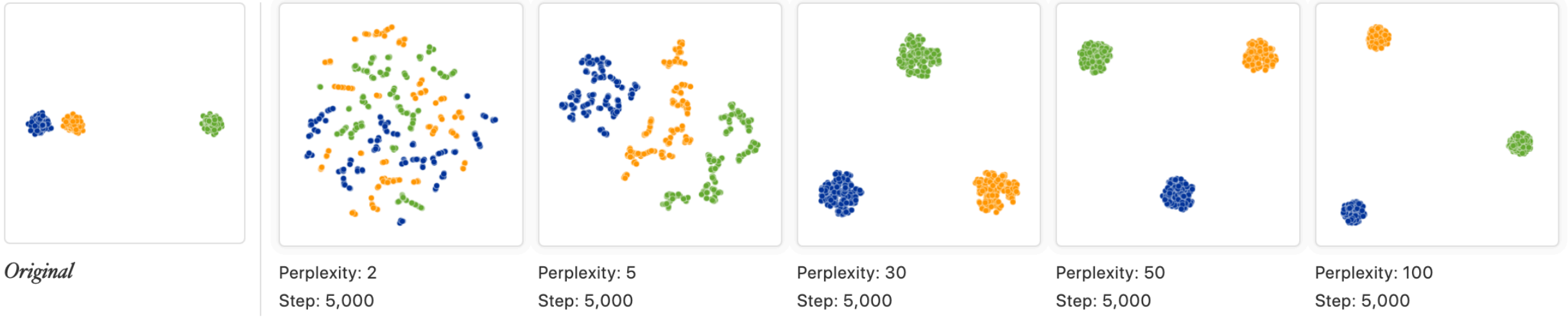
Cluster size doesn't matter. (Sigma's are set to normalize different local point densities.)



t-Distributed Stochastic Neighbor Embedding (t-SNE)

Some comments on using t-SNE (taken from https://distill.pub/2016/misread-tsne/):

Distance between clusters might not be meaningful.



Original

Perplexity: 2
Step: 5,000

Perplexity: 5
Step: 5,000

Perplexity: 30
Step: 5,000

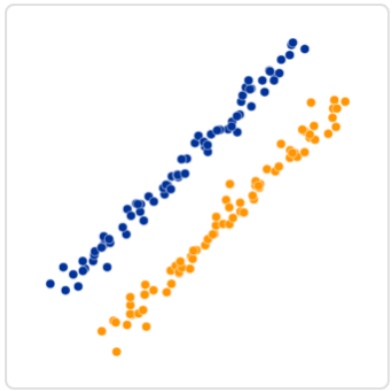
Perplexity: 50
Step: 5,000

Perplexity: 100
Step: 5,000

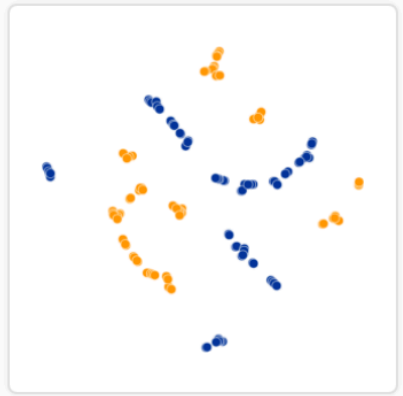
t-Distributed Stochastic Neighbor Embedding (t-SNE)

Some comments on using t-SNE (taken from https://distill.pub/2016/misread-tsne/):

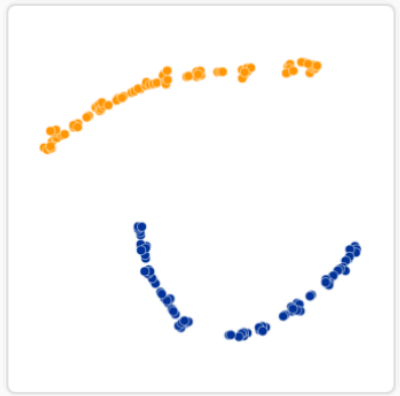
Shapes may be distorted.



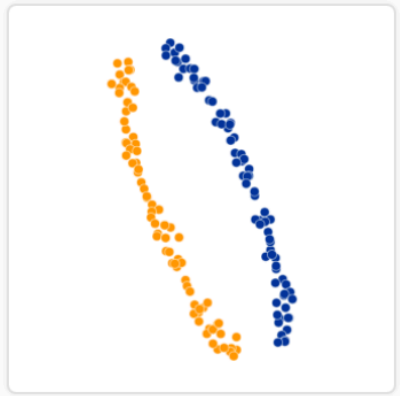
Original



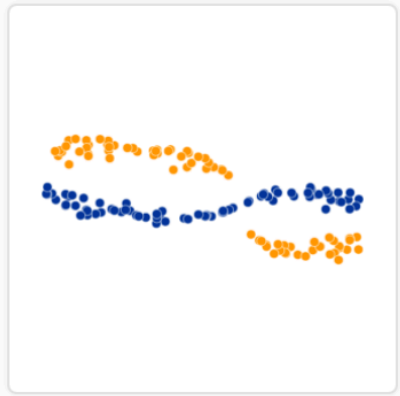
Perplexity: 2
Step: 5,000



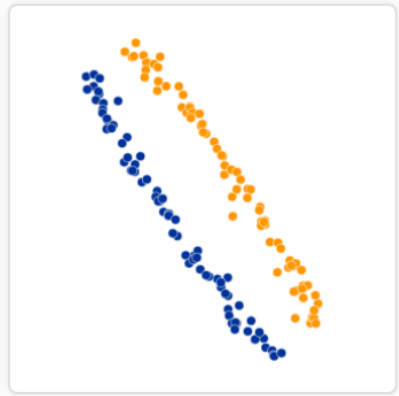
Perplexity: 5
Step: 5,000



Perplexity: 30
Step: 5,000



Perplexity: 50
Step: 5,000

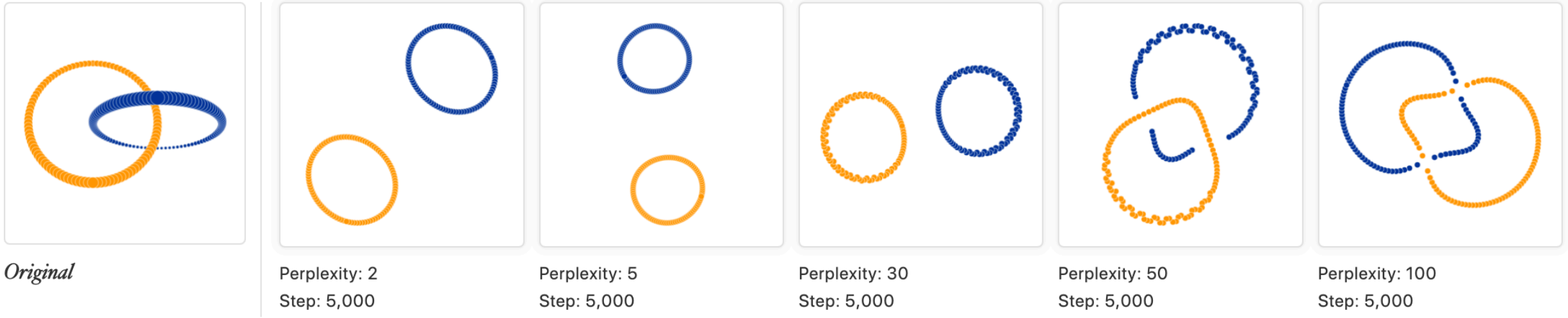


Perplexity: 100
Step: 5,000

t-Distributed Stochastic Neighbor Embedding (t-SNE)

Some comments on using t-SNE (taken from https://distill.pub/2016/misread-tsne/):

Topology is not always preserved.



Original

Perplexity: 2
Step: 5,000

Perplexity: 5
Step: 5,000

Perplexity: 30
Step: 5,000

Perplexity: 50
Step: 5,000

Perplexity: 100
Step: 5,000

Next Time: We'll talk a bit about kernel density estimation and reinforcement learning.