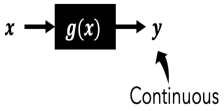
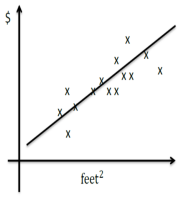
Unsupervised Learning

- What is unsupervised learning and how does it differ form supervised learning?
- What are some problems in unsupervised learning?
- What is clustering?
- How does k-means clustering work?
- What is k-means optimizing?
- What is a coordinate descent algorithm?
- What is a Gaussian Mixture Model?
- What can this tell us about k-Means?
- How do we evaluate clustering?

Supervised Learning
Learn to predict output from input given example (x,y) pairs

Regression: Predicting a (or multiple) continuous values as output

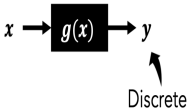


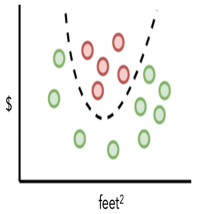


Example: Predicting the price of a house based on square footage

Supervised Learning
Learn to predict output from input given example (x,y) pairs

Classification: Predicting a (or multiple) discrete variable as output

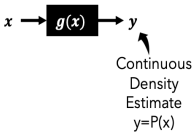


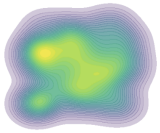


Example: Predicting if a house will sell based on price and square footage

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

Density Estimation: Estimate the underlying distribution generating our data

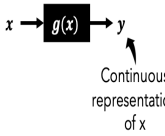


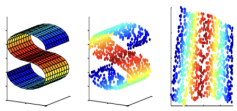


Example: Estimate how likely is a given house with these properties

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

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



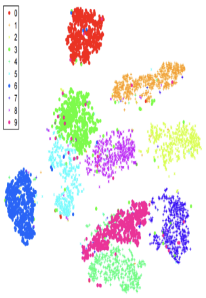


Example: Finding a 2D subspace in a 3D feature space

In general, clustering can be viewed as an exploratory procedure for finding interesting subgroups in a dataset.

Most work in clustering is on the setting where we want to:

- Group all given examples **(exhaustive)** into disjoint clusters **(partitional)** such that
  - Examples within a cluster are similar
  - Examples in different clusters are different



Our First Clustering Algorithm: **k-Means**

Conceptual Overview of k-Means:

- Start with random initial points to represent the center of the k clusters.
- Form initial clusters based on these random starting points.
- Iteratively refine these clusters and stop when no longer making changes.

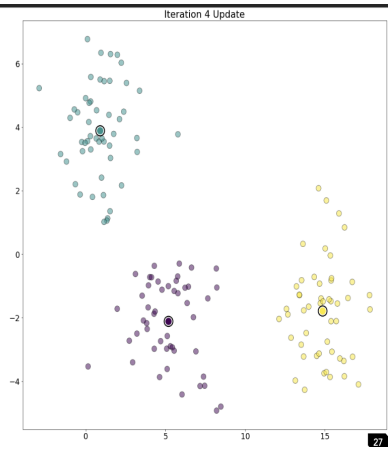
Hyperparameters:

- k - the number of clusters
- Some distance function, we'll assume Euclidean (aka L2) for now

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



Formalizing K-Mean Clustering

Given a dataset $X = \{\boldsymbol{x}_i\}_{i=1}^n$ where $\boldsymbol{x}_i \in \mathbb{R}^d$ and the number of desired clusters k, **produce** a set of assignments $Z = \{z_i\}_{i=1}^n$ where $z_i \in \{1,2,\ldots,k\}$ and a set of centroids $C = \{\boldsymbol{c}_j\}_{j=1}^k$ where $\boldsymbol{c}_j \in \mathbb{R}^d$.

Do this by alternating the following steps:

a) Assignment: For each datapoint i, update z_i:

$$z_i = \underset{j=1,2,..,k}{\operatorname{argmin}} \left\|\boldsymbol{x}_i - \boldsymbol{c}_j\right\|_2^2$$

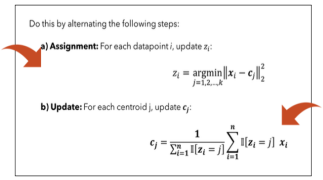
b) Update: For each centroid j, update $\boldsymbol{c}_j$:

$$\boldsymbol{c}_j = \frac{1}{\sum_{i=1}^n \mathbb{I}[z_i = j]} \sum_{i=1}^n \mathbb{I}[z_i = j]\ \boldsymbol{x}_i$$

Complexity of k-Means

Computing L2 distance between two points is O(d).

Need distance between each datapoint and each centroid. **O(knd)**

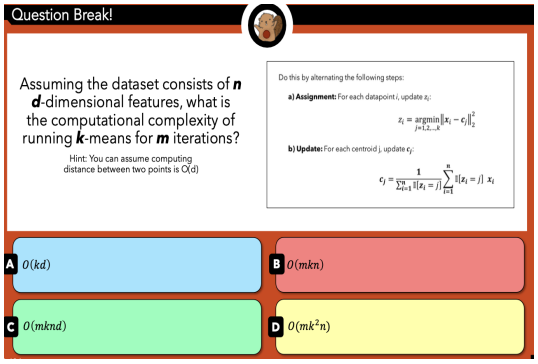


Each datapoint contributes to only one cluster, total of n vector additions. **O(nd)**

Repeat this for $\boldsymbol{m}$ iterations.

Overall computational complexity is therefor O(mknd)

- linear in relevant inputs (for fixed iteration count)



What is k-Means optimizing?

Claim: k-Means is minimizing the sum of squared error between points and their associated cluster centroid. That is, the optimal centroids and assignments are:

$$C^*, Z^* = \underset{C,Z}{\operatorname{argmin}}\ SSE(X, C, Z) = \underset{C,Z}{\operatorname{argmin}} \sum_{i=1}^n \left\|\boldsymbol{x}_i - \boldsymbol{c}_{z_i}\right\|_2^2$$

This is an optimization over two sets of variable - assignments and centroids - and mixes discrete / continuous variables. Very hard in general.

How do we get from this problem to the 2-step process we defined before?

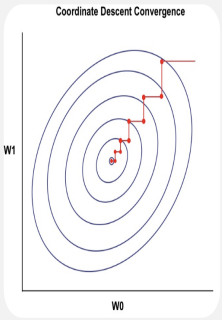
Coordinate Descent

Suppose we have some function $f(w_1, w_2)$ we want to minimize.

Coordinate Descent would alternative between the following steps:

1. Fix w_2 as a constant and optimize w_1
2. Fix w_1 as a constant and optimize w_2

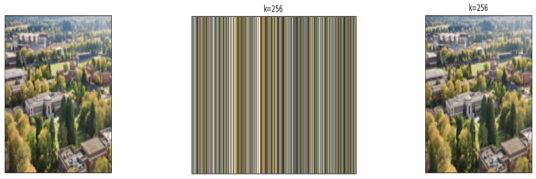
Why do this? Sometimes functions have closed-form / easy solutions in some variables if the others are fixed.



Does it converge to an optima? Yes for convex, differentiable f. Not generally.

Image Compression / Vector Quantization: Let my dataset be the set of all pixels in an image and I want to find clusters to replace each pixel with its centroids - significant size reduction!

Original | Centroid Colors | Reconstruction



Only store id of centroid at each pixel. Store the centroids as well.

Question Break!

Intuition Check: Following this coordinate descent algorithm we described; k-Means is _____.

A Guaranteed to converge in finite steps to the global minima of SSE

B Guaranteed to converge in finite steps to a local minima of SSE

C Not guaranteed to converge in finite steps

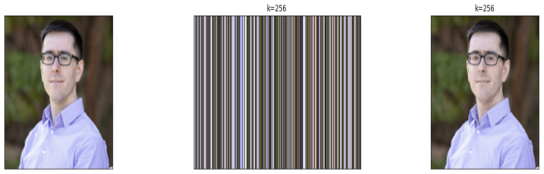
D I don't know how to think about this.

CS 434

Some Examples of k-Means in Action

Image Compression / Vector Quantization: Let my dataset be the set of all pixels in an image and I want to find clusters to replace each pixel with its centroids - significant size reduction!

Original | Centroid Colors | Reconstruction



Only store id of centroid at each pixel. Store the centroids as well.

Discovering Similar Datapoints: Took a dataset of faces and represented them with an edge-sensitive image feature called HoG (see your homework for more details). Clustered them!

One of the Clusters | Random Sample

ity of k-Means Optimization

Each iteration of k-Means strictly decreases the SSE until convergence. Proof relies it both the assignment and update steps cannot increase SSE.

ent: Each point only switches to a new cluster if the squared error is *lower*:

$$z_i = \underset{j=1,2,\ldots,k}{\operatorname{argmin}} \|\boldsymbol{x}_i - \boldsymbol{c}_j\|_2^2$$

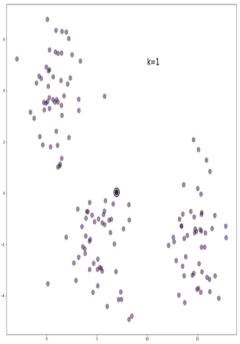
Each centroid is updated to the mean of its assignments, which we already showed ne sum-of-squared error with respect to a fixed assignment.

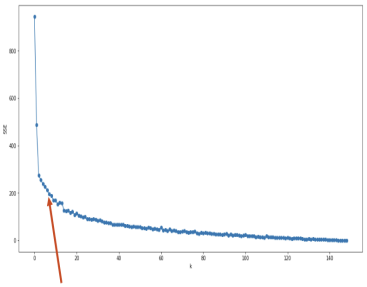
$$\boldsymbol{c}_j = \frac{1}{\sum_{i=1}^{n} \mathbb{I}[\boldsymbol{z}_i = j]} \sum_{i=1}^{n} \mathbb{I}[\boldsymbol{z}_i = j]\ \boldsymbol{x}_i$$

Properties: How do we choose k?

The number of clusters **k** is an important hyperpameter. How can we choose it?

- Unfortunately, SSE decreases with k so training set won't tell us much.



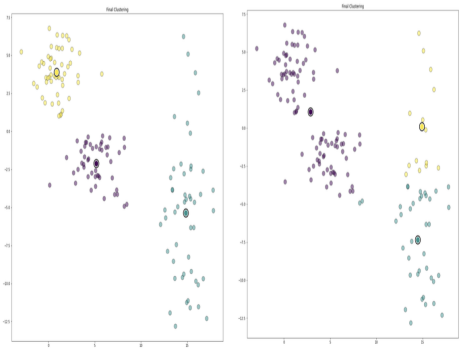


One common heuristic is to pick the "elbow/knee" of the SSE vs. k graph.

Properties: Highly Sensitive to Initialization

k-Means is highly sensitive to initialization of the centroids.

- Sum-of-squared error has many local minima where no local reassignments can reduce SSE.



Properties: Highly Sensitive to Initialization

How can we deal with this sensitivity?

Run multiple trials and choose the one with the lowest SSE (often used in practice)

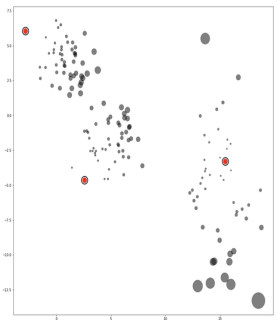
Try to initialize the centroids "well" - not clear how to do this. Some common heuristics:

- **Random Points:** Initialize the centroids by copying random datapoints - ensures your centroids are near data.
- **Far-Away Points:** Try to make the cluster centers far from each other:
  - Samples a datapoint and set the first centroid c_1 to its value
  - Compute a weight w_i for each datapoint proportional to its distance from c_1. Then sample according to this distribution.
  - Repeat with weights proportional to the nearest centroid.

Properties: Highly Sensitive to Initialization

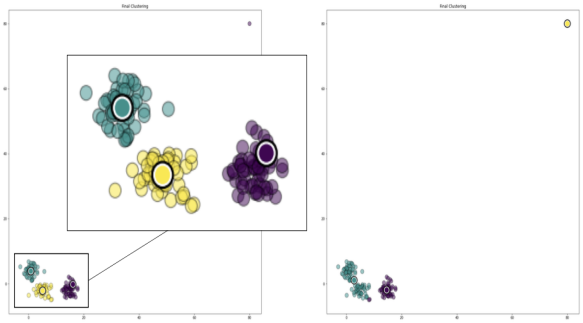
Example of Iterative Distance-Based Sampling Initialization: $w_i \propto e^{\left(\min_{c \in C}(x_i - c)^2\right)}$

Point size proportional to probability of being selected this iteration.



Properties: Sensitive to Outliers

Every point needs to have a cluster and every cluster center is the average of its members.



Properties: Sensitive to Outliers

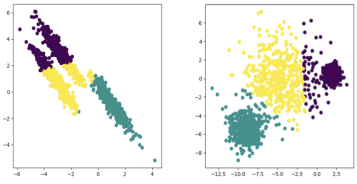
Rough Idea of k-medoids Algorithm: Instead of taking the mean in the centroid update step, take the median - i.e., the data point that is closest to the other points in the cluster.

$$c_j = \frac{1}{\sum_{i=1}^{n} \mathbb{I}[z_i = j]} \sum_{i=1}^{n} \mathbb{I}[z_i = j]\, x_i \quad \Rightarrow \quad c_j = \underset{z \in M_j}{\operatorname{argmin}} \sum_{x_i \in M_j} \|x_i - z\|_2^2$$

K-Medoids is computationally more expensive but more robust to outliers.

Properties: Some Hidden Assumptions

k-Means performs poorly when the clusters are not spherical or when different clusters have very different spreads. Seems like there are some hidden assumptions here…



Where do these assumptions come from? Spoiler: Its Gaussian again!

Summary of k-Means

Given a dataset, **k-Means splits it into k disjoint groups.**

- Setting k is largely heuristic as larger k produces lower SSE
  - Unsupervised learning has no validation set because it has no labels
- Guaranteed to converge in finite steps to a local minima
  - Often in a few iterations in practice
- O(mknd) computational complexity makes running k-Means tractable

Properties of k-means:

- Not particularly resistant to outliers
- Very sensitive to initialization of centroids - need to run multiple times
- Only seems to work on spherical clusters with similar sizes

Gaussian Mixture Model

Gaussian Mixture Models (GMM)

Given: a dataset $D = \{\boldsymbol{x}_i\}_{i=1}^n$ where $x \in \mathbb{R}^d$

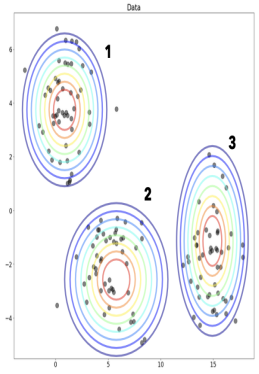
Goal: Fit the distribution $P(\boldsymbol{x})$

But this seems like a weird distribution… what if I approximate it as multiple Gaussians?

An Idea: Assume the following generative story of how this data was created.

Assume each point was generated by:

1. Sampling one of the three clusters, then
2. Sampling a point from a Gaussian defining that cluster

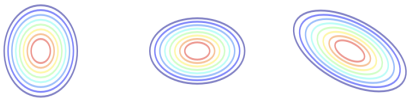


Gaussians can be defined in multiple dimensions. Describe the probability of a vector given the

- Mean vector μ – describing the location / center of the distribution
- Covariance matrix Σ – describing the spread of the distribution along different directions

$$\mathcal{N}(x;\mu,\Sigma) = \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}} e^{-(\boldsymbol{x}-\boldsymbol{\mu})^T\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}$$

Can draw them in 2D as a contour plot where lines show rings of equal probability and color shows relative probability:

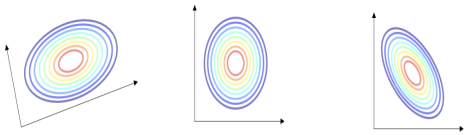


CS 434

Question Break!

What would you describe these Gaussians as? (Be as specific as possible)

Isotropic ⊂ Axis-Aligned ⊂ Full Covariance



A Isotropic, Isotropic, Axis-Aligned

B Full Covariance, Isotropic, Full Covariance

C Axis-Aligned, Axis-Aligned, Isotropic

D Axis-Aligned, Isotropic, Full Covariance

Gaussian Mixture Models (GMM)

Given: a dataset $D = \{\boldsymbol{x}_i\}_{i=1}^n$ where $x \in \mathbb{R}^d$

Goal: Fit the distribution $P(\boldsymbol{x})$ as a Gaussian Mixture Model with k components.

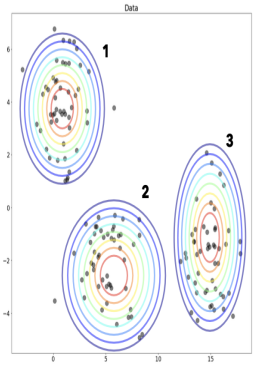
Formalizing our generative story: Assume each point x_i was generated by the process:

$$z_i \sim P(z)$$

$$\boldsymbol{x}_i \sim \mathcal{N}(x;\mu_{z_i},\Sigma_{z_i})$$

Call the z's "latent or hidden variables" as they are not observed from our data and must be inferred.

Need to learn:
- The means and covariances of the k Gaussians.
- The Categorical distribution P(z) over the k Gaussians.



CS 434

Let's Start Simple – A 2d Case with k=2

What if we magically *knew* what value z_i… wouldn't need to marginalize anymore

$$LL_{magic} = \sum_{i=1}^{n} \log\left(P(z_i)\mathcal{N}(\boldsymbol{x}_i;\boldsymbol{\mu}_{z_i},\Sigma_{z_i})\right) = \sum_{i=1}^{n} \log \mathcal{N}(\boldsymbol{x}_i;\boldsymbol{\mu}_{z_i},\Sigma_{z_i}) + \log P(z_i)$$

Could easily find the MLE solutions to this problem:

$$\theta_c^* = \frac{\sum_i \mathbb{I}[z_i = c]}{n}$$

Fraction of points with z = c

$$\boldsymbol{\mu}_c^* = \frac{1}{\sum_i \mathbb{I}[z_i = c]} \sum_i \mathbb{I}[z_i = c]\, \boldsymbol{x}_i$$

Mean of points with z = c

$$\Sigma_c^* = \frac{1}{\sum_i \mathbb{I}[z_i = c]} \sum_i \mathbb{I}[z_i = c]\, (\boldsymbol{x}_i - \boldsymbol{\mu}_c)(\boldsymbol{x}_i - \boldsymbol{\mu}_c)^T$$

Covariance of points with z = c

Could do something very similar if someone gave us "soft" values for z's too.

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 $\theta_1, \ldots, \theta_k$ all to 1/k
means of the Gaussians $\mu_1, \ldots, \mu_k$ to random points
covariances of the Gaussians $\Sigma_1, \ldots, \Sigma_k$ to identity matrices

E-Step: Compute fractional assignment of point i coming from class 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

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

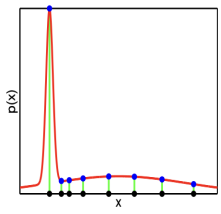
Weighted mean of fractional points assigned to c

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

Weighted covariance of fractional points assigned to c

Properties: Singularities in GMM

Log-likelihood can go to infinity if one of the Gaussians "collapses". Assume one of the Gaussians has only one point assigned and the covariance heads towards zero -- $\mu = x$ and $|\Sigma| \to 0$



$$\mathcal{N}(x;\mu,\Sigma) = \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}} e^{-(x-\mu)^T\Sigma^{-1}(x-\mu)} = \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}}$$

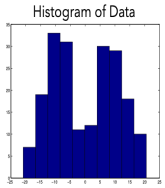
$$\lim_{|\Sigma|\to 0} \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}} = \infty$$

How to deal with this? Monitor each component and reset it to something random if it starts collapsing (random value, large covariance). Or add a prior to the covariance and do MAP in the M-Step of the EM algorithm.

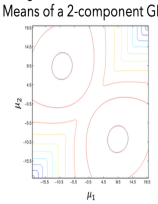
Properties: Identifiability Issues

The log-likelihood in Gaussian Mixture Models has multiple identical maxima. Simple consider swapping the parameters between different models to see why.

Histogram of Data



Log-Likelihood for Different Means of a 2-component GMM



No easy fix. Also limited harm, part of the reason convergence may be slow.

© Stefan Lee

Behavior of Expectation Maximization for GMM

It is guaranteed to converge in finitely many steps:

- Proof looks like the k-Means version but harder – shows that log-likelihood must increase or remain the same between iterations
- In practice, it may converge slowly. Can stop early if no progress is made on log-likelihood for a long time.

Not guaranteed to converge to the global optima:

- Like k-Means, do multiple restarts and then choose the one with highest log likelihood.
- Has a couple of "divergent" solutions

Note: Expectation Maximization (EM) is a whole family of algorithms for dealing with hidden/latent variables. Not just used in Gaussian Mixture Models.

Summary of Gaussian Mixture Models

Assumes data is generated from *k* independent Gaussians:

- Can model more complex configurations than k-Means but is slightly more costly and difficult to implement.
- Produces a full density model of the data → enables you to sample new synthetic data or evaluate the probability of some new point.
  - More on density estimation next class.
- Fractional assignments can be turned into hard clusterings by taking the argmax for each point.

Uses the expectation maximization algorithm for optimization:

- May be slow to converge or end up in trivial optima.
- May need multiple restarts.

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

How to get k-Means back from GMMs:

- Assume hard-assignment rather than fractional.
- Assume all Gaussians have the same isotropic covariance.

The Expectation Maximization (EM) Algorithm for GMM

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

E-Step: Compute fractional assignment of point i coming from class c

$P(z_i = c \mid x_i) \propto P(z_i = c)\mathcal{N}(x_i; \mu_c, \Sigma_c)$ denote this $p_{c|x_i}$

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

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

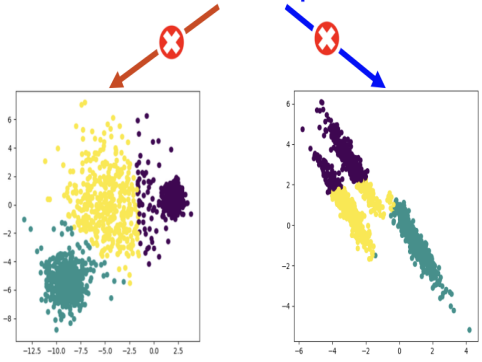
$$\theta_c^* = \frac{\sum_i p_{c|x_i}}{n} \qquad \mu_c^* = \frac{1}{\sum_i p_{c|x_i}} \sum_i p_{c|x_i} x_i \qquad \Sigma_c^* = \frac{1}{\sum_i p_{c|x_i}} \sum_i p_{c|x_i} (x_i - \mu_c)(x_i - \mu_c)^T$$

Fraction of mass assigned to c | Weighted mean of fractional points assigned to c | Weighted covariance of fractional points assigned to c

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

How to get k-Means back from GMMs:

- Assume hard-assignment rather than fractional.
- Assume all Gaussians have the **same isotropic** covariance.



Question Break!

What all can Gaussian Mixture Models do that k-Means can't?

- **A** Fit clusters with different spreads
- **B** Fit clusters that aren't isotropic / spherical
- **C** Provide an estimate of P(X)
- **D** Allow for new synthetic points to be sampled from the distribution of the training data

Evaluating Clustering

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.

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.

Evaluating Clustering

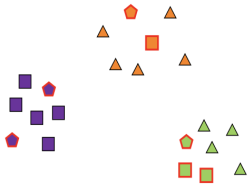
With external data:

- 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.

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.



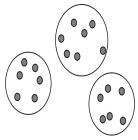
$$Purity = \frac{5+5+4}{21}$$

What is Hierarchical Agglomerative Clustering (HAC)?

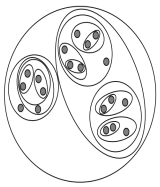
- What are different linking strategies?
- How can we evaluate clustering?
- What is dimensionality reduction?
- What is principal component analysis (PCA) and how does it work?
- What are stochastic neighbor embeddings (SNE) and how do they work?
- What is kernel density estimation (KDE)?
- How does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

Two Categories of Clustering Algorithms

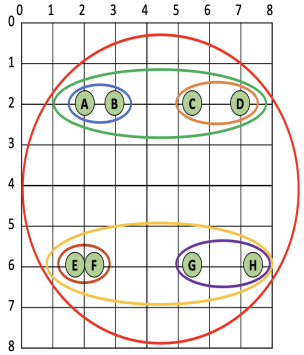
Partition Algorithms ("Flat" Clusterings)
- k-Means / k-Medians
- Gaussian Mixture Models

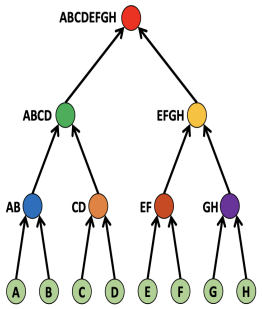


Hierarchical Algorithms
- Bottom-up - Agglomerative
- Top-down - Divisive



Hierarchical Agglomerative Clustering (HAC)

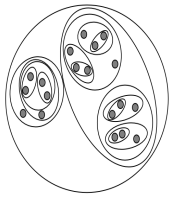




Hierarchical Agglomerative Clustering (HAC)

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.

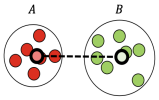
Question: We have a measure of distance between points, how do we use it to measure "closeness" of clusters?

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 $d(x, y)$:

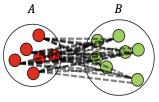
Centroid
- The distance between the cluster means

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



Average-link
- Average distance between all cross-cluster pairs

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



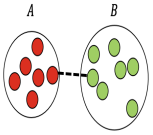
- **Most robust and most commonly used**

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 $d(x, y)$:

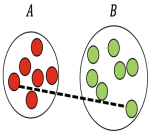
Single-link
- The distance between the nearest points

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



Complete-link
- The distance between the furthest points

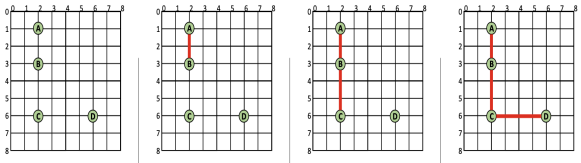
$$d(A, B) = \max_{x_a \in A, x_b \in B} d(x_a, x_b)$$



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				-

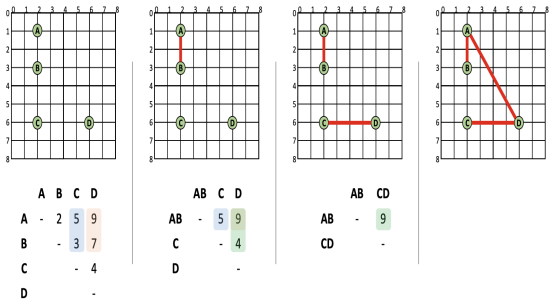
	AB	C	D
AB	-	3	7
C		-	4
D			-

	ABC	D
ABC	-	4
D		-

Complete-Link Method Example (L1 distance)

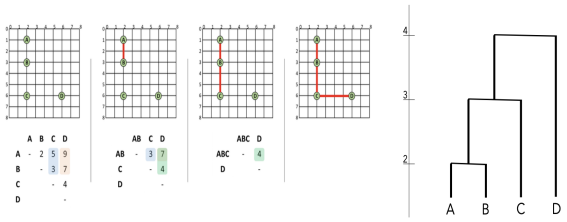
Complete-link
- The distance between the furthest points

$$d(A,B) = \max_{x_i \in A, x_j \in B} d(x_i, x_j)$$

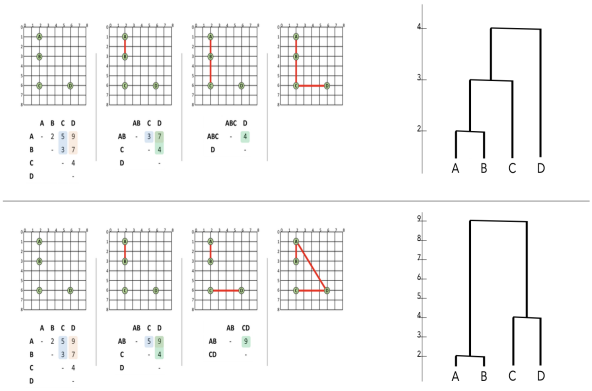


Visualizing Hierarchical Clusterings: Dendrogram

- 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)



Visualizing Hierarchical Clusterings: Dendrogram

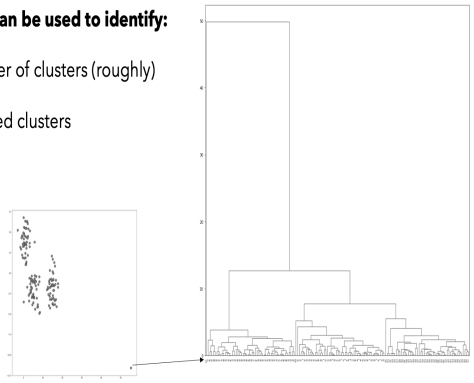


 Slide adapted from Xiaoli Fern

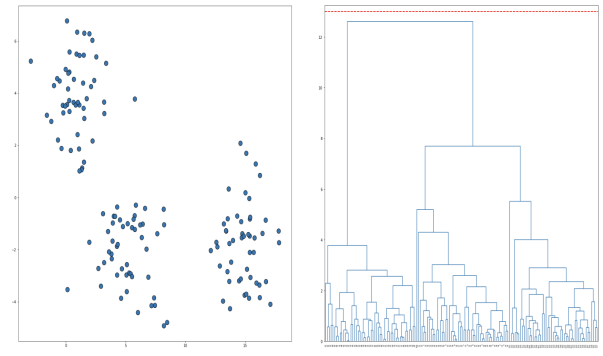
Interpreting Dendrograms

Dendrograms can be used to identify:

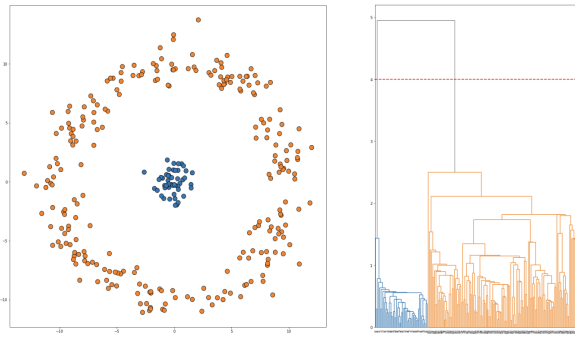
- The number of clusters (roughly)
- Well-formed clusters
- Outliers



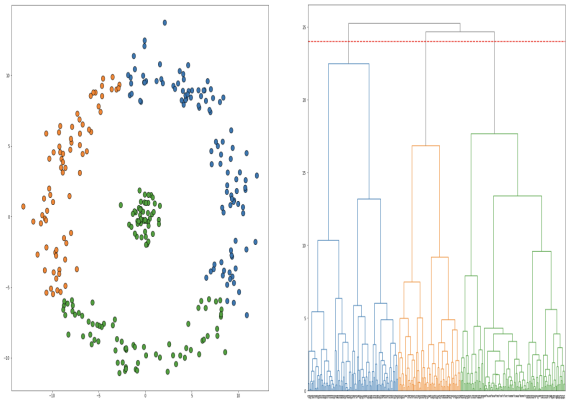
Visualizing Hierarchical Clusterings: Dendrogram



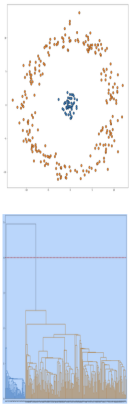
Single Link (Distance to Closest Point)



Complete Link (Distance to Furthest 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

Summarizing Hierarchical Agglomerative Clustering

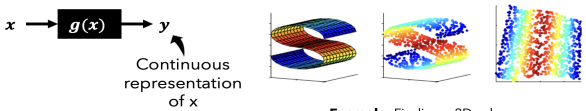
- HAC is a convenient tool that often provides interesting views of a dataset
- Primarily HAC can be viewed as an intuitively appealing clustering procedure for data analysis/exploration
- We can create clusterings of different granularity by stopping at different levels of the dendrogram
- HAC often 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

- 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



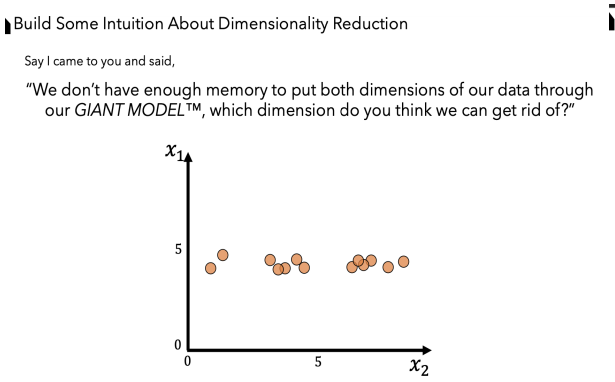
Example: Finding a 2D subspace in a 3D feature space

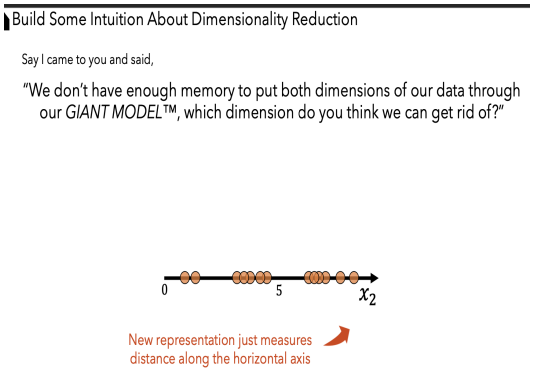
New Topic: Dimensionality Reduction

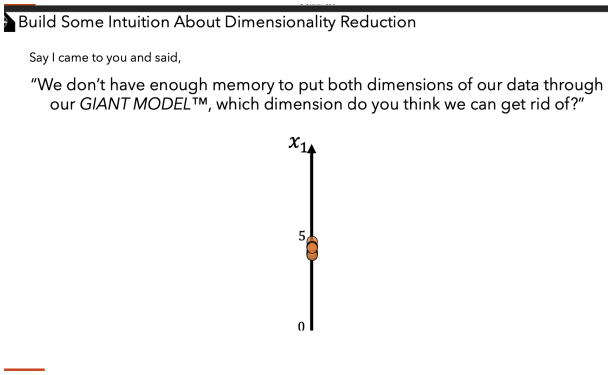
Dimensionality reduction tries to find a more compact representation of the data –- creating new features with lower dimensionality that still represents the data well.

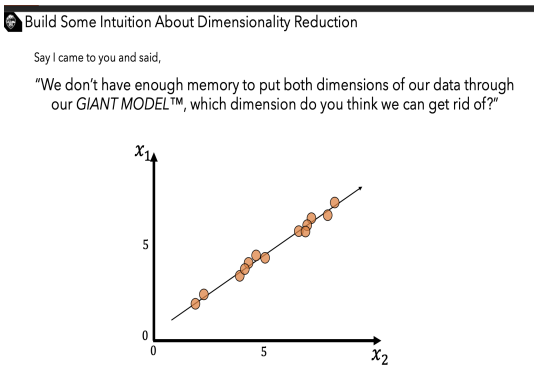
Why would we want to do this?

- 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.





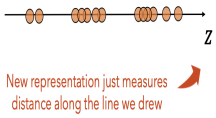




Build Some Intuition About Dimensionality Reduction

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?"

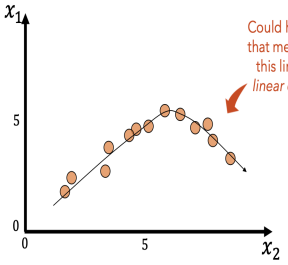


New representation just measures distance along the line we drew

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.*

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?

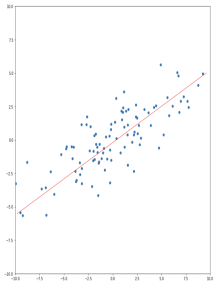
• Suppose two features dim-0 and dim-1, but can only keep one:
  • For dim-0, most examples have similar values (small variance)
  • For dim-1, most examples differ (large variance)

• Keep the one with more variance! It explains more about the difference between items

Principal Component Analysis (PCA) -- Conceptually

Consider doing the following:

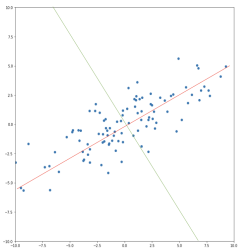
- Find a line such that most of the variance of the data follows along that line
- Or equivalently, a line where the dataset projected onto that line maintains as much variance.
- Call this line your first principal component.



Principal Component Analysis (PCA) -- Conceptually

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)

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

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:

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

→ For Eigenvector w and Eigenvalue λ_1 of Σ_x

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

→ By the constraint

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

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.

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 $\frac{1}{n} X^T X$
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.

What have we just done?

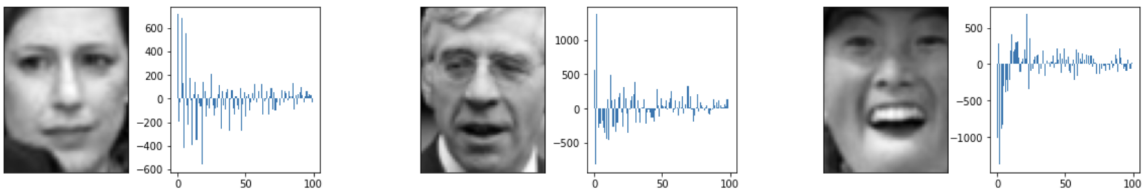
We wanted to find a line to project our data onto that would preserve as much variance as possible. So to find that linear transform we:

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.

Classic Example – Eigenfaces

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

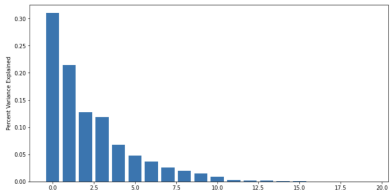


Choosing How Far To Reduce Dimensionality

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

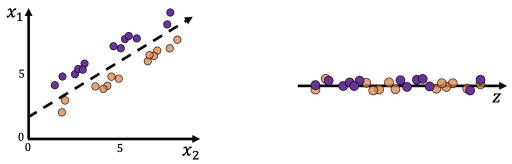
- Use fraction of variance explained to judge how representative the visualization is

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

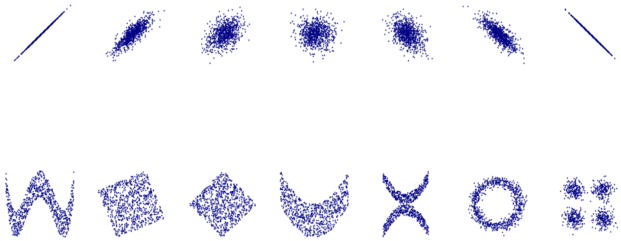
Direction of maximum variance may not be useful for classification:



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

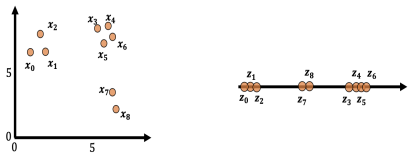
Only works at finding linear subspace...



Non-Linear Dimensionality Reduction: t-SNE

t-Distributed Stochastic Neighbor Embedding (t-SNE)

Idea: For each **d-dimensional** input vector 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.



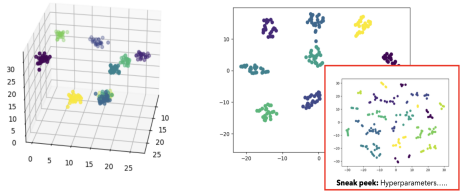
Important Note: We aren't learning a function $z_i = f(x_i)$, we are learning the z vectors directly!

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 xi, directly optimize a k-dimensional vector zG 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 **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 z_i that make the distribution $q_{i|*}$ match $p_{i|*}$ then we would have the same structure in this neighborly sense.

t-Distributed Stochastic Neighbor Embedding (t-SNE)

Hyperpamameters:

- The number of lower dimensions **k**
- The sigma's for each datapoint:

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

  - 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

•

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)

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

Hyperparameters really matter.

Distance between clusters might not be meaningful.

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

Shapes may be distorted.