EE1083/EEN1085
Data analysis and machine learning

Ali Intizar

Ollscoil Chathair Bhaile Átha Cliath
Dublin City University

Unsupervised Learning

Ollscoil Chathair Bhaile Átha Cliath
Dublin City University

Overview

Motivation and goals of unsupervised learning

Clustering

- K-Means
- Agglomerative clustering

Motivation

- Vast amounts of unlabelled data
- Most data has structure; we would like to discover hidden structure
- Modelling the **probability density** of the data P(X)
- Fighting the **curse of dimensionality**
- **Visualizing** high-dimensional data
- Supervised learning tasks: learning from fewer training examples

Assumptions

It is necessary to make some assumptions to learn structure from data.

“You can’t do inference without making assumptions”

-- David MacKay, Information Theory, Inference, and Learning Algorithms

Typical assumptions:

- Smoothness assumption
  - Points which are close to each other are more likely to share semantics.
- Cluster assumption
  - The data form discrete clusters; points in the same cluster are likely to share semantics
- Manifold assumption
  - The data lie approximately on a manifold of much lower dimension than the input space.

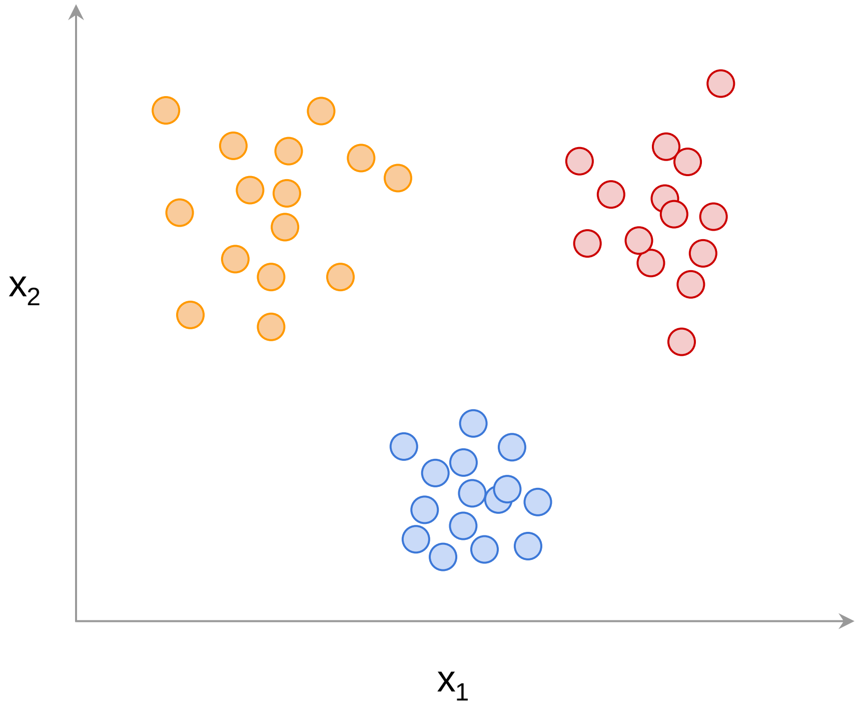
Clustering

Cluster assumption: data form discrete clusters.

We would like to come up with an algorithm to automatically discover these clusters from data.

Will look at two approaches:

1. **K-Means**: formulate as an optimization problem. Find approximate solution using iterative algorithm.
2. **Agglomerative clustering**: iterative greedy bottom-up algorithms that produce a hierarchical clustering



K-Means

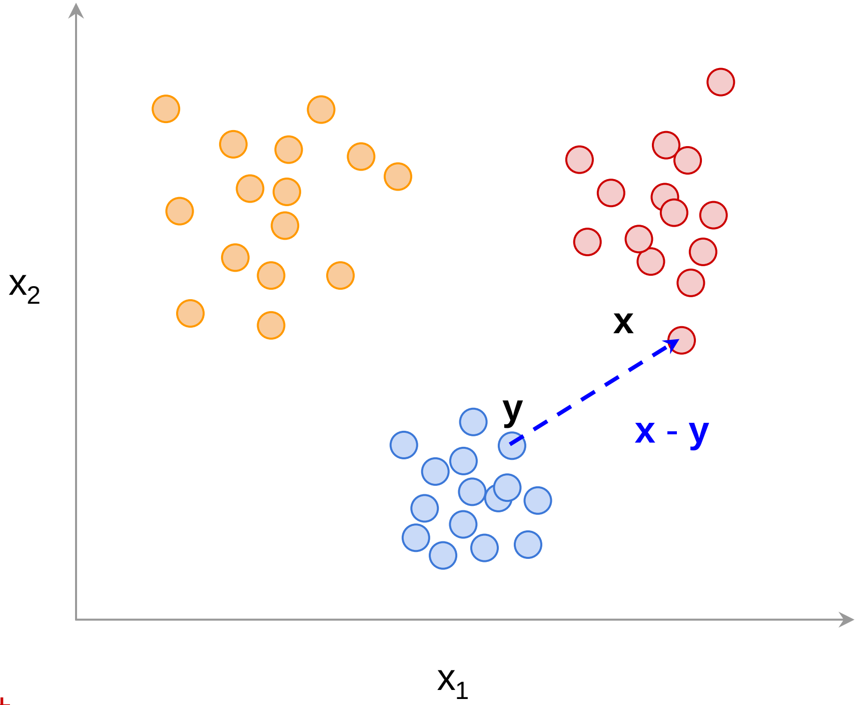
Simple but very popular clustering algorithm

Aims to find a fixed number (*k*) of discrete clusters such that the average distance from a point to the center of its cluster is minimized.

Distance is taken to be the square Euclidean distance.

$$d(\mathbf{x}, \mathbf{y}) = ||\mathbf{x} - \mathbf{y}||_2^2$$
$$= (\mathbf{x} - \mathbf{y})^T (\mathbf{x} - \mathbf{y})$$

$$= (x_1 - y_1)^2 + (x_2 - y_2)^2 + \dots$$



K-Means

k-means objective:

$$\underset{\mathbf{S}}{\arg\min} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2$$

where:

- $\boldsymbol{S} = \{\boldsymbol{S}_1, \boldsymbol{S}_2, \ldots, \boldsymbol{S}_k\}$ is the set of non-overlapping clusters assignments
- $\boldsymbol{S}_i$ is set of all points in cluster *i,*
- $\boldsymbol{\mu}_i$ is centroid of cluster *i* (mean of all points in $\boldsymbol{S}_i$)

Discrete optimization problem: objective function is non-smooth and **non-convex.**

Objective is **NP-hard** even in 2D: impossible to solve in polynomial time.

K-Means algorithms attempt to find an **approximate solution** (local minimum of the objective function) in polynomial time.

Lloyd's algorithm

Iterative approach for finding a local minimum of the *k*-means objective.

Idea: start with randomly cluster centres. At each iteration, move them to reduce the cost.

Algorithm
Start with *k* random chosen cluster centers

While not converged:
1. **Assign** each point (descriptor) to nearest cluster center
2. **Update** cluster center to centroid of all points assigned to it

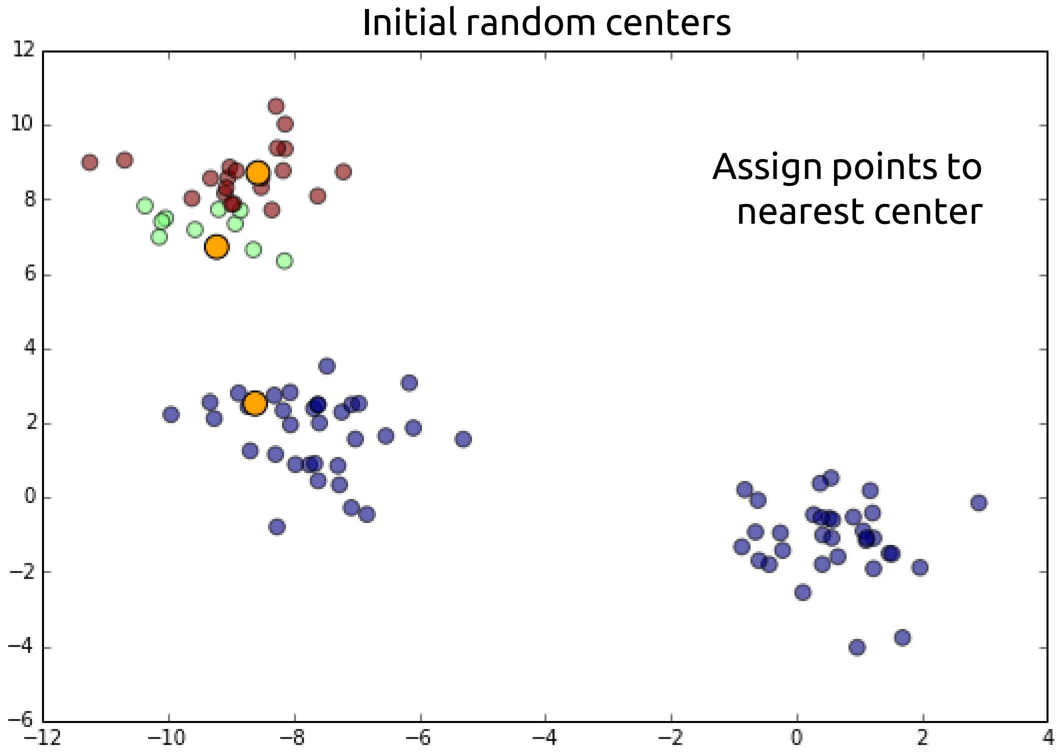
Convergence is when assignments don't change or change in position of cluster centers is not significant

Example of a two-step **coordinate descent** algorithm:

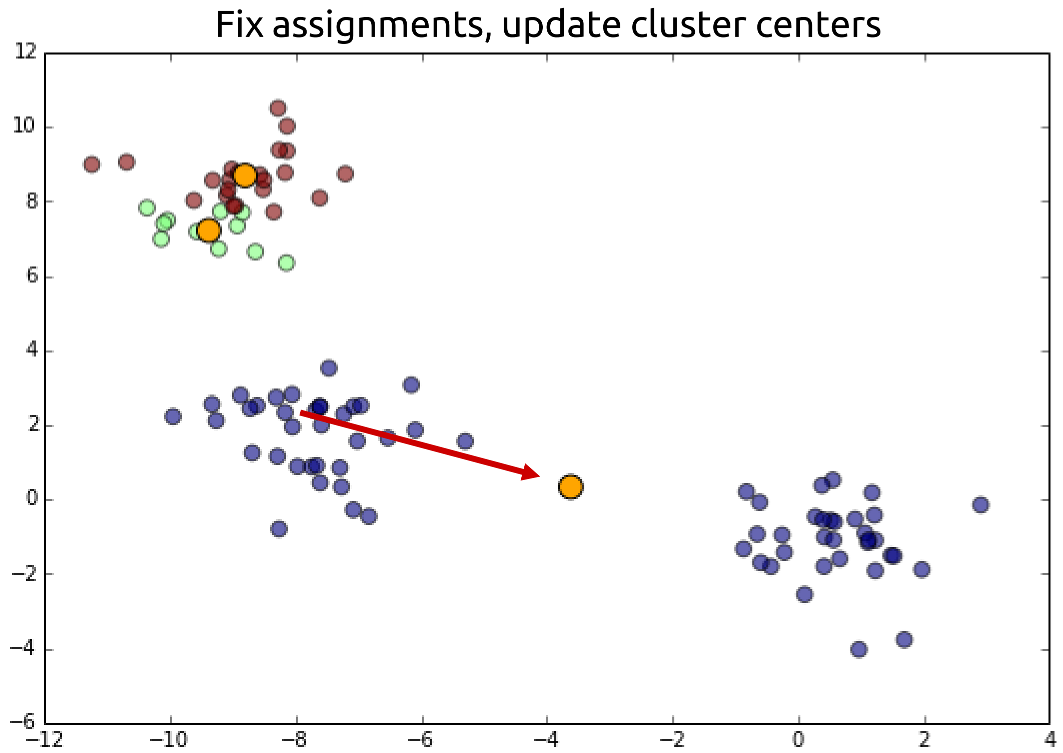
- **Step 1**: fix the cluster centers, find the optimal assignments
- **Step 2**: fix the assignments, find the optimal cluster centers

Coordinate descent: fix A, minimize B, fix B, minimize A, ...

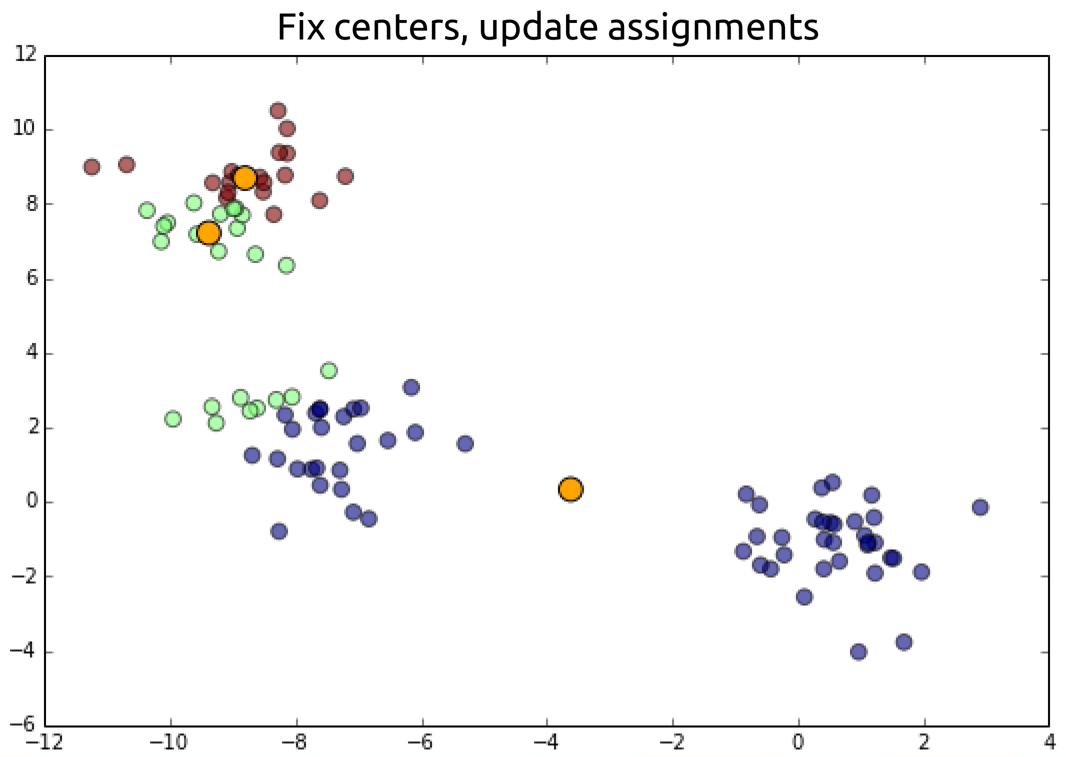
Demo of Lloyd's algorithm



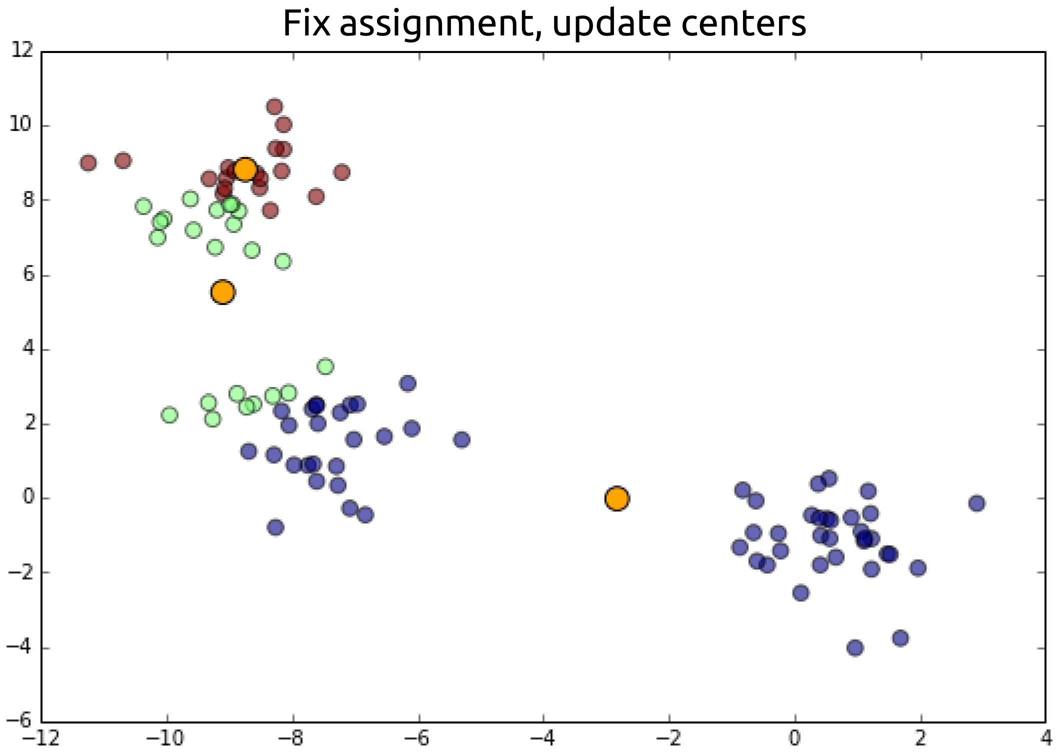
Demo of Lloyd's algorithm



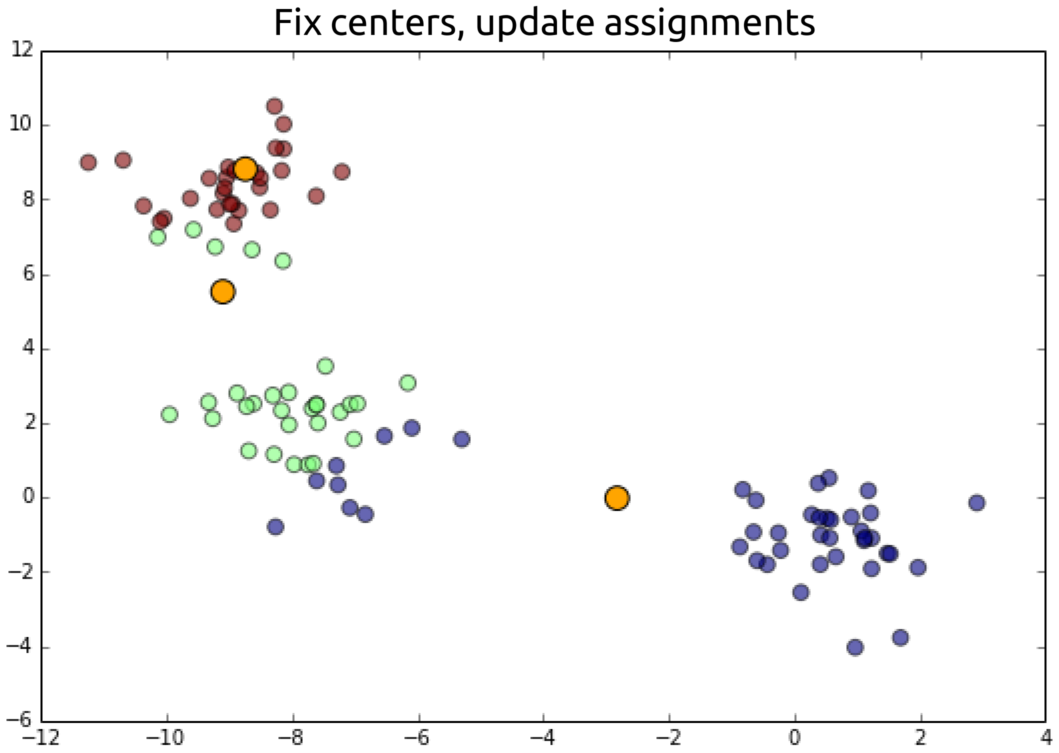
Demo of Lloyd's algorithm



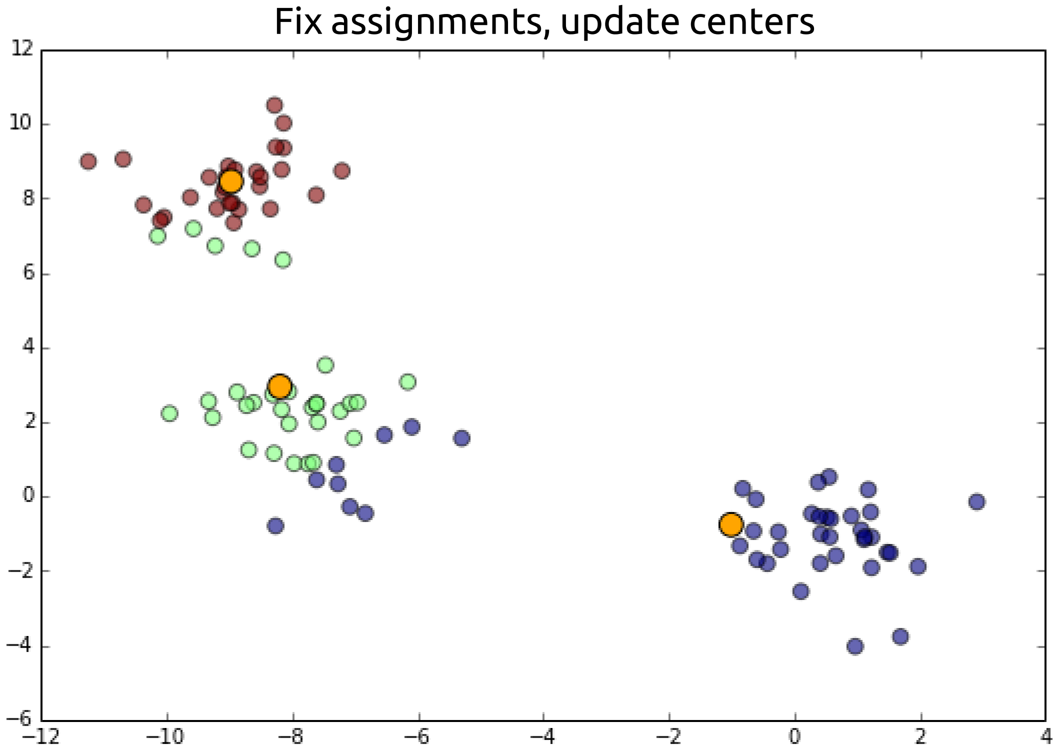
Demo of Lloyd's algorithm



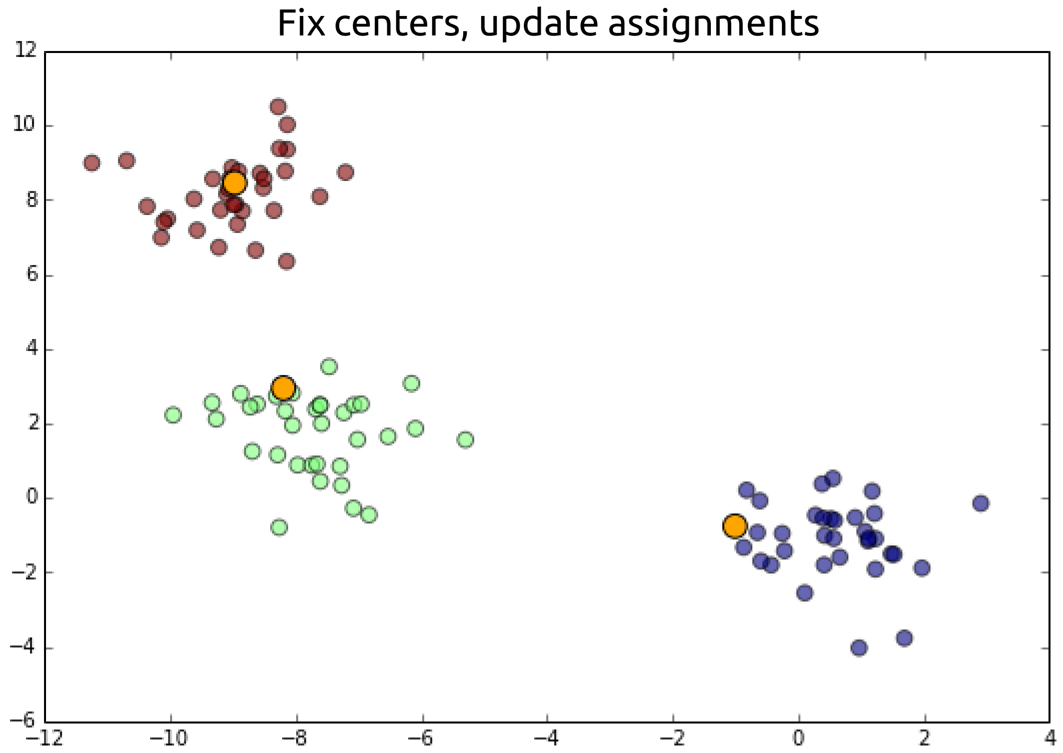
Demo of Lloyd’s algorithm



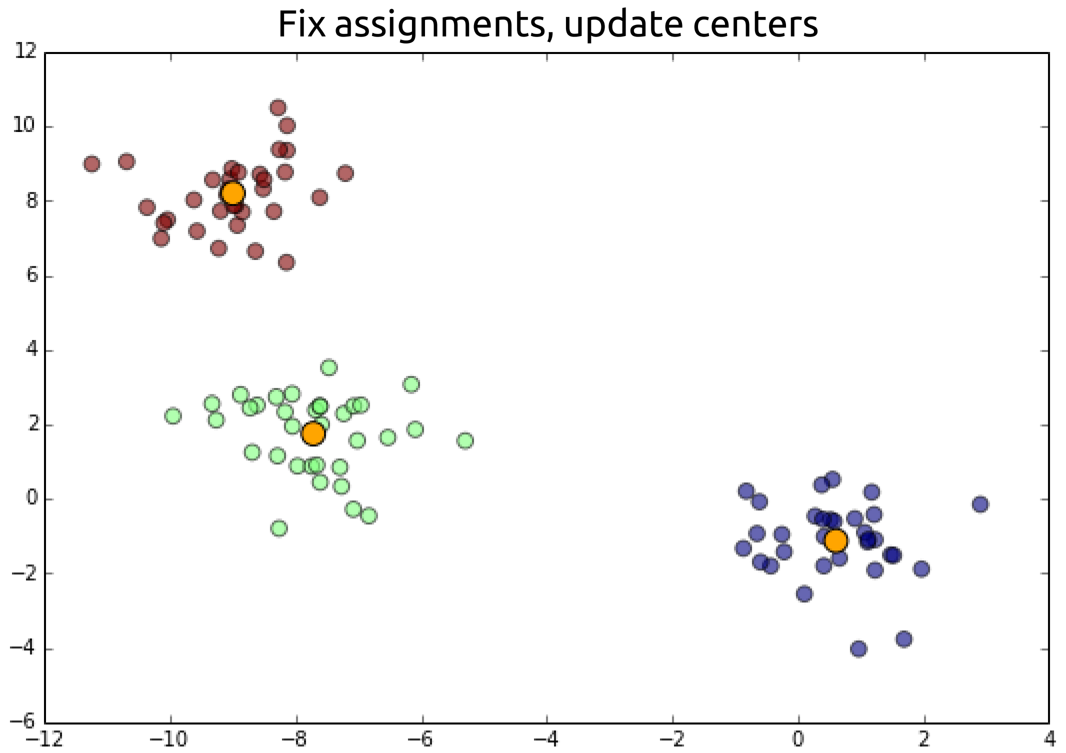
Demo of Lloyd's algorithm



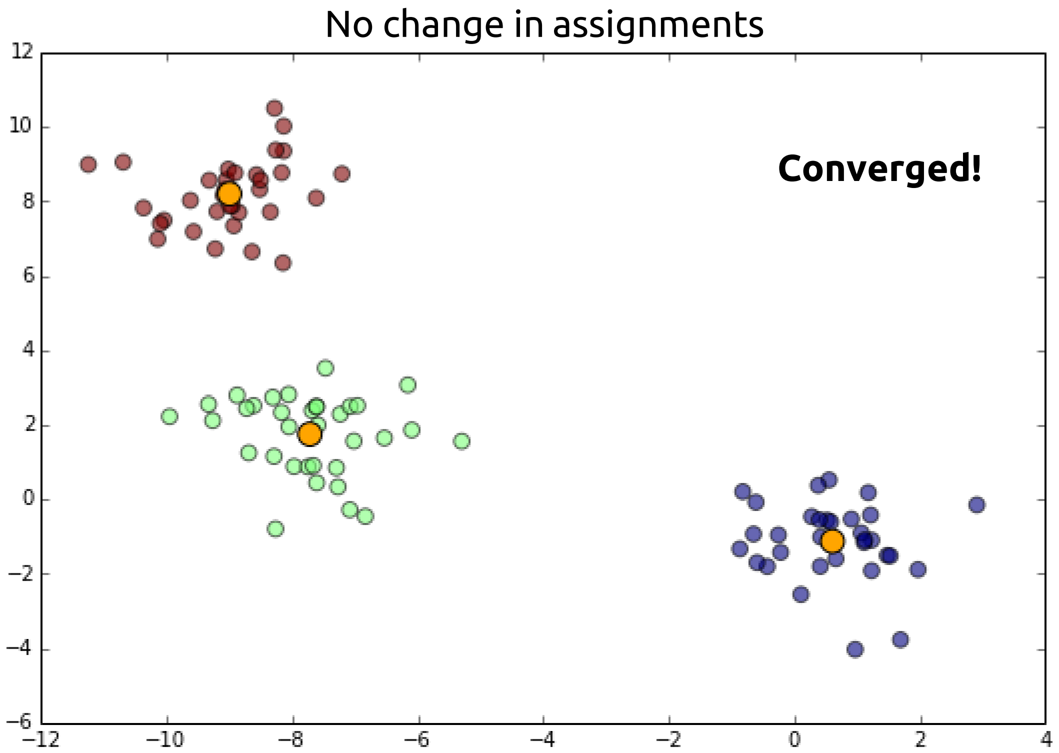
Demo of Lloyd's algorithm



Demo of Lloyd’s algorithm



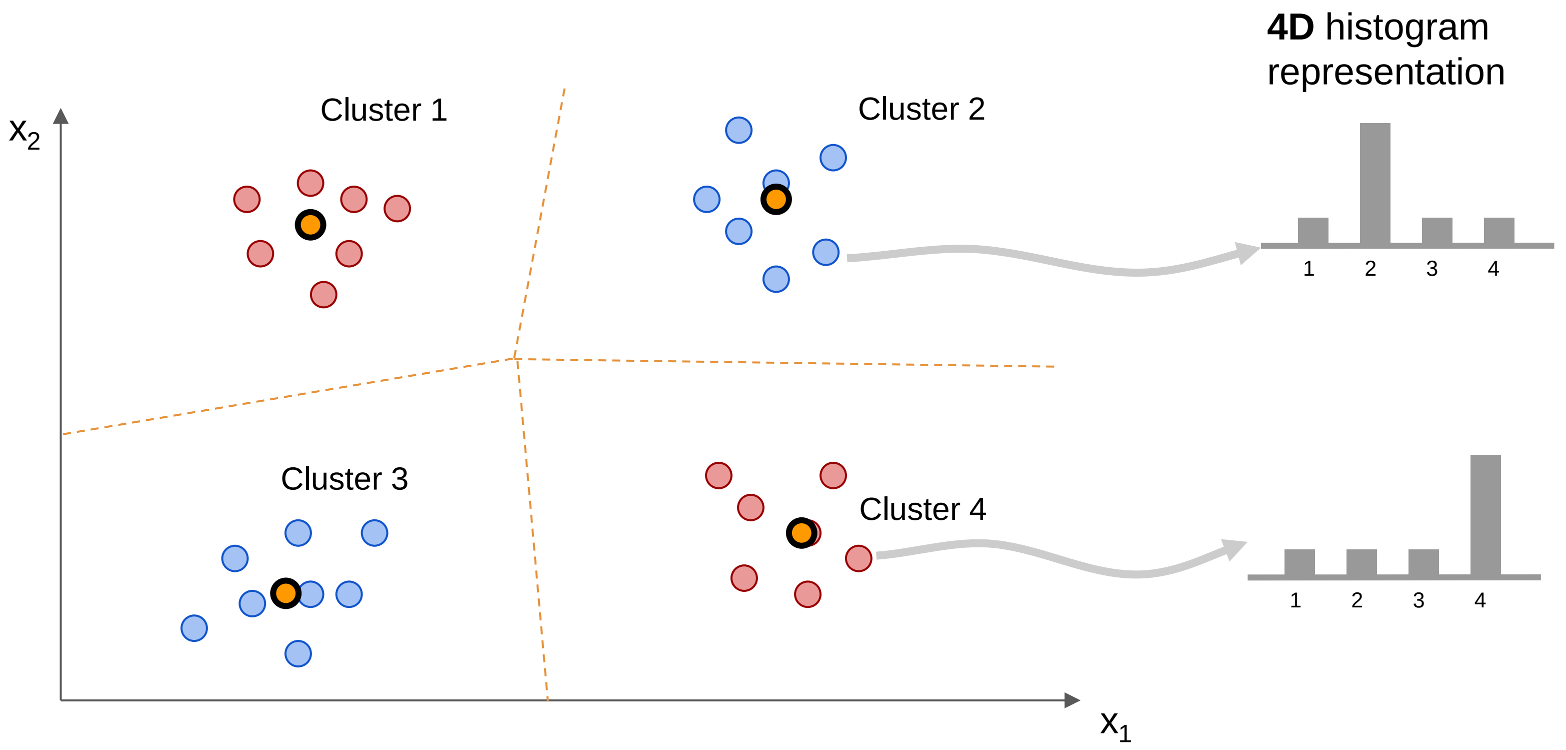
Demo of Lloyd's algorithm



Codebook representation

- Possible now to represent a data point using the index of nearest cluster center. This is called **vector quantization**.
- Distances to nearest cluster center induces a **Voronoi tessellation** of the space.
- In this context, the set of cluster centers is often called the **codebook**.
- Easy to compute a histogram of the counts of points assigned to each cluster center.

Vector quantization



Speeding up distance computations

At each step in *k*-means we need to compute the distance from every data point to every cluster center.

$$d(\mathbf{x}_i, \mu_j) = (\mathbf{x}_i - \mu_j)^T(\mathbf{x}_i - \mu_j)$$

Can be computed very quickly for all centers and data points using a matrix multiplication

Possible to use binomial expansion to speed this up:

$$d(\mathbf{x}_i, \mu_j) = \mathbf{x}_i^T\mathbf{x}_i + \mu_j^T\mu_j - 2\mathbf{x}_i^T\mu_j$$

Doesn't change when centers change: compute once for all data points at start

$$X = \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix}, \quad C = \begin{bmatrix} \mu_1 & \cdots & \mu_K \end{bmatrix}, \quad \langle \mathbf{x}_i, \mu_j \rangle = (XC)_{ij}.$$

Quick to compute when number of cluster centers is relatively small

$$\left[2\mathbf{x}_i^T\mu_j\right] = 2XC^T$$

K-Means code

```python
def kmeans(data, k=3, max_iters=100):

    # precompute norms for fast distance computations
    data_norms = np.sum(data**2, axis=1)

    def get_distances_to_centers(centers):

        # binomial trick for fast distances
        center_norms = np.sum(centers ** 2, axis=1)
        dists = -2 * np.dot(data, centers.T)
        dists += center_norms[np.newaxis,:]
        dists += data_norms[:,np.newaxis]
        return dists

    def get_assignments(centers):
        dists = get_distances_to_centers(centers)
        assignments = np.argmin(dists, 1)
        return dists, assignments

    def get_updated_centers(assignments):
        centers = [data[assignments==i,:].mean(axis=0)
                   for i in xrange(k)]
        return np.array(centers)
```

```python
    def get_initial_centers():
        N = data.shape[0]
        indices = np.random.randint(0, N, k)
        return data[indices, :]

    # start off with initial random centers
    centers = get_initial_centers()
    prev_assignments = None
    for i in xrange(max_iters):

        # fix centers and update assignments
        dists, assignments = get_assignments(centers)

        # fix assignments and update centers
        centers = get_updated_centers(assignments)

        # if nothing changes, we have converged
        if prev_assignments is not None:
            if np.sum(assignments != prev_assignments) == 0:
                break

        prev_assignments = assignments

    return assignments, centers
```

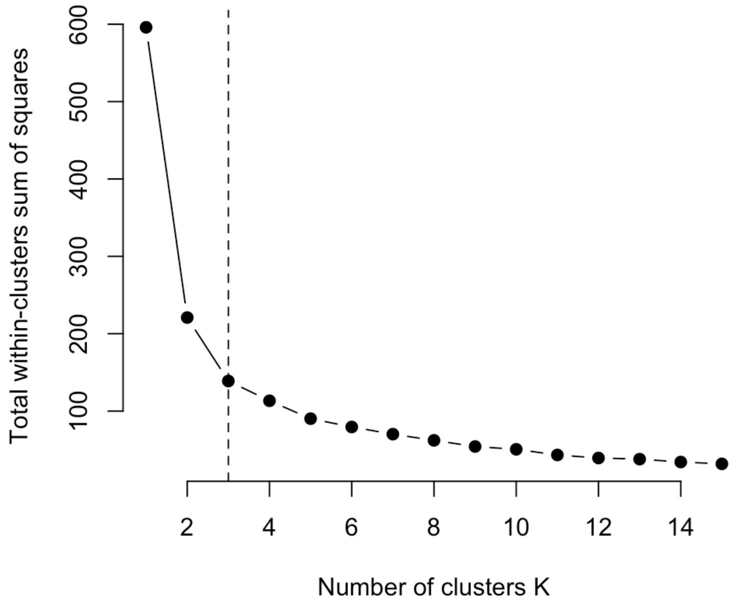
Choosing *k*

Typical heuristic:

- Plot within cluster sum of squares for different values of *k*
- Choose *k* at the “elbow” in the plot

The gap statistic

- More formal way of finding the elbow algorithmically
- **Idea**: generate random datasets and compare within cluster sum of squares of clusterings on random data with real data as *k* increases.



The gap statistic algorithm

1. Generate B randomly distributed datasets that are uniformly distributed over the ranges of the original attributes.
2. For increasing values of k:
   - 2.1 Using k centers, cluster original and reference datasets.
   - 2.2 Calculate the log point scatter $\log(W_k)$ for clusterings on the original dataset
   - 2.3 Calculate the log point scatter $\log(W^*_{kb})$ for clusterings on the reference datasets
   - 2.4 Calculate the gap statistic: $G_k = \mathbf{E}_B[\log(W^*_{kb})] - \log(W_k)$
3. Choose $\hat{k} =$ smallest k such that $G_k \geq G_{k+1} - s_{k+1}$, where $s_k = \sigma_k \sqrt{1 + 1/B}$, with σ_k being the standard deviation of the log point scatter on the reference datasets for k clusters.

Robert Tibshirani, Guenther Walther, Trevor Hastie, *Estimating the number of clusters in a data set via the gap statistic*, Journal of the Royal Statistical Society: Series B (Statistical Methodology), vol. 63 (2), pp. 411-423, 2001

K-Means notes

Sensitive to initialization: different initializations can produce different clusterings

- Can run several times with different initializations
- Use clustering with minimum cost function value

Alternative initialization strategies: kmeans++

1. Choose first seed at random
2. Find distance to all other points
3. Choose next seed randomly with probability proportional to distance
4. Repeat step 2 and 3 until all seeds chosen

Approximate k-means: possible to make *k*-means practical for very large datasets (#bigdata) by using a **fast approximate nearest neighbour algorithm (e.g. FAISS)**

- When computing cluster assignments we need to find the nearest cluster centre (nearest neighbour)

Overview

Motivation and goals of unsupervised learning

Clustering

- K-Means
- Agglomerative clustering

Agglomerative clustering

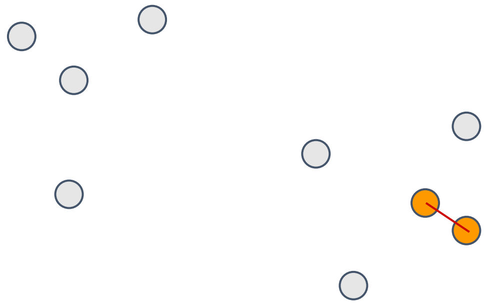
Bottom-up **recursive hierarchical clustering** of points.

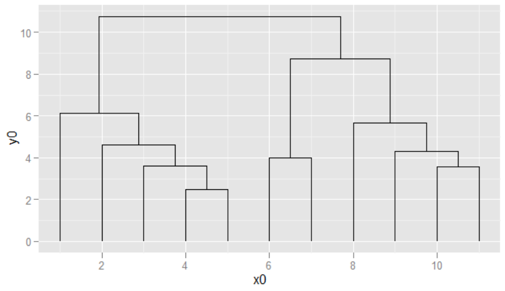
Need to specify two things:

1. Distance measure (e.g. Euclidean)
2. Linkage type

Linkage types

1. Single linkage (min)
2. Complete linkage (max)
3. Average linkage (mean)
4. Centroid linkage
5. Ward linkage





Dendrogram

Linkage

Single linkage: distance between two clusters is the **minimum** distance between any single data point in the first cluster and any single data point in the second cluster

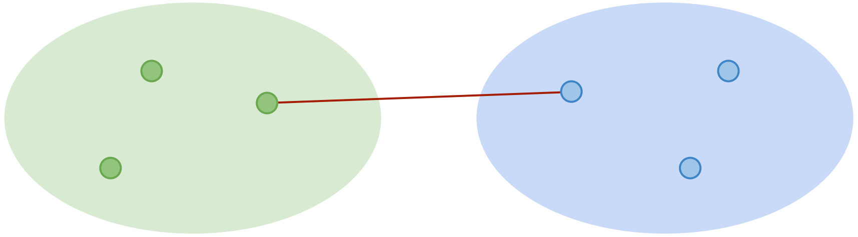
Complete linkage: the distance between two clusters is the **maximum** distance between any single data point in the first cluster and any single data point in the second cluster

Average linkage: the distance between two clusters is the **average** distance between data points in the first cluster and data points in the second cluster.

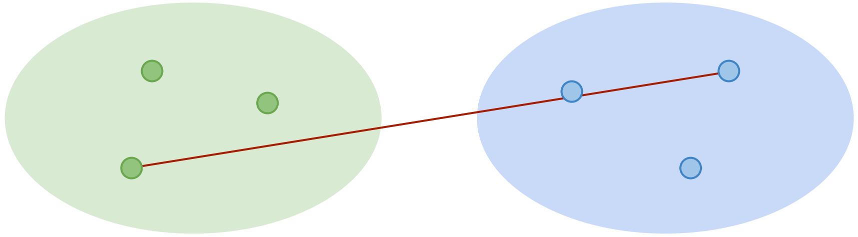
Centroid method: the distance between two clusters is the distance between the two mean vectors (centroids) of the clusters.

Ward's method: merge the clusters that create the smallest increase in cluster variance.

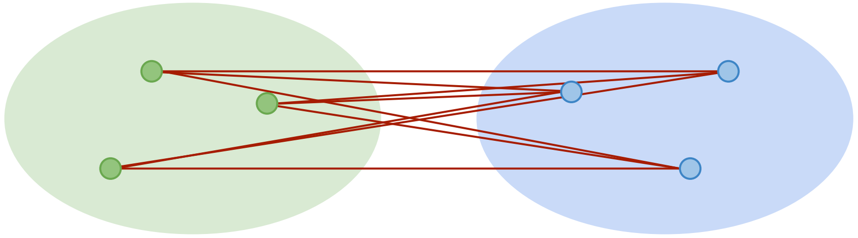
Single linkage



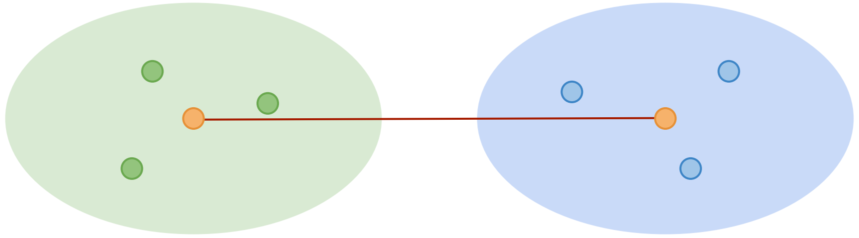
Complete linkage



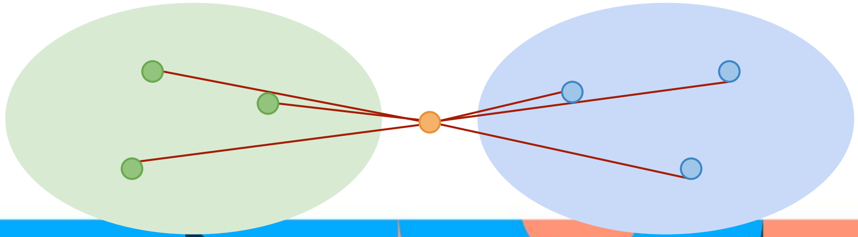
Average linkage



Centroid linkage



Ward linkage



Overview

Motivation and goals of unsupervised learning

Clustering

- K-Means
- Agglomerative clustering

Dimensionality reduction

- PCA
- t-SNE
- Other methods

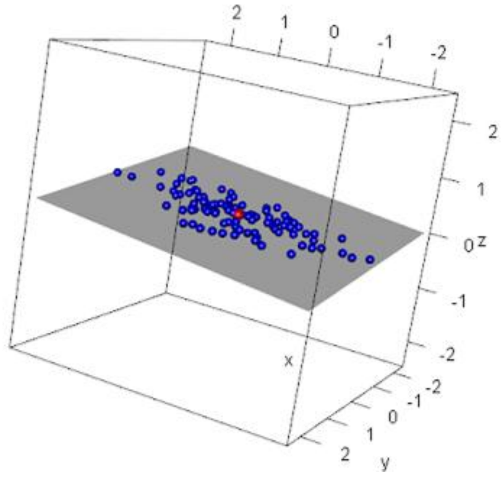
Dimensionality Reduction

Assumption: the data lie close to a lower dimensional manifold embedded in a high dimensional space.

Linear dimensionality reduction: further assume that this manifold is a hyperplane.

- 1D: a line
- 2D: a plane
- N-D: hyperplane

We would like to recover the coordinates of the data on this manifold.



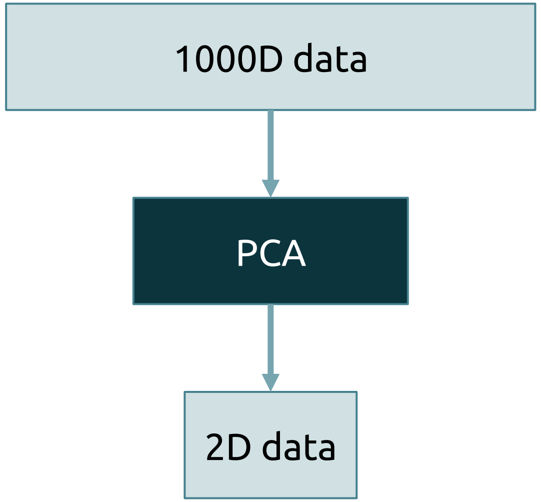
Why?
- Visualization
- Compression
- Summarization
- Supervised learning

Principal component analysis (PCA)

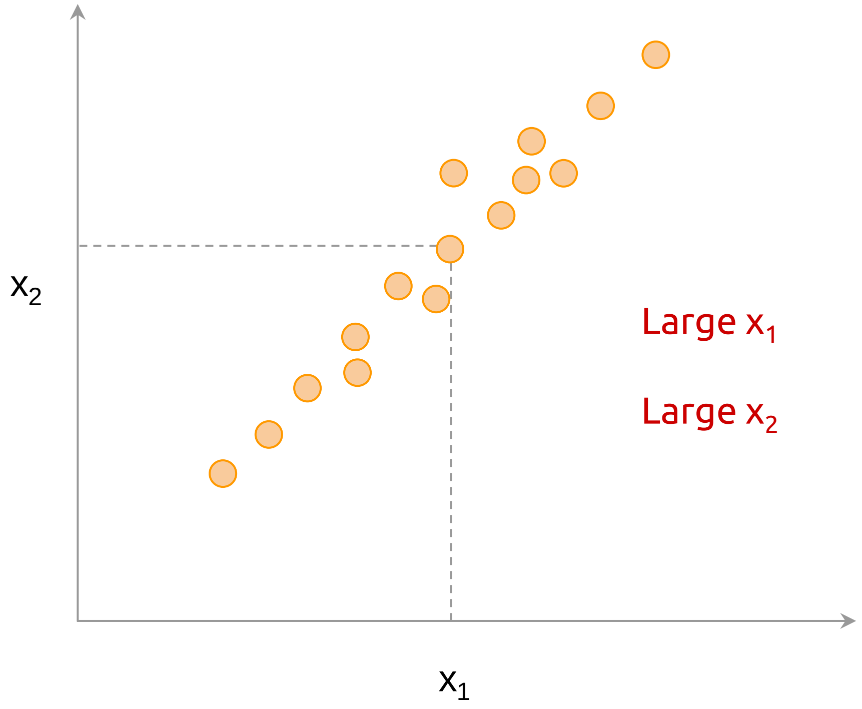
Reduce the dimensionality of the data by **projecting it onto a linear subspace** and **discarding axes of least variation**.

Very widely used:

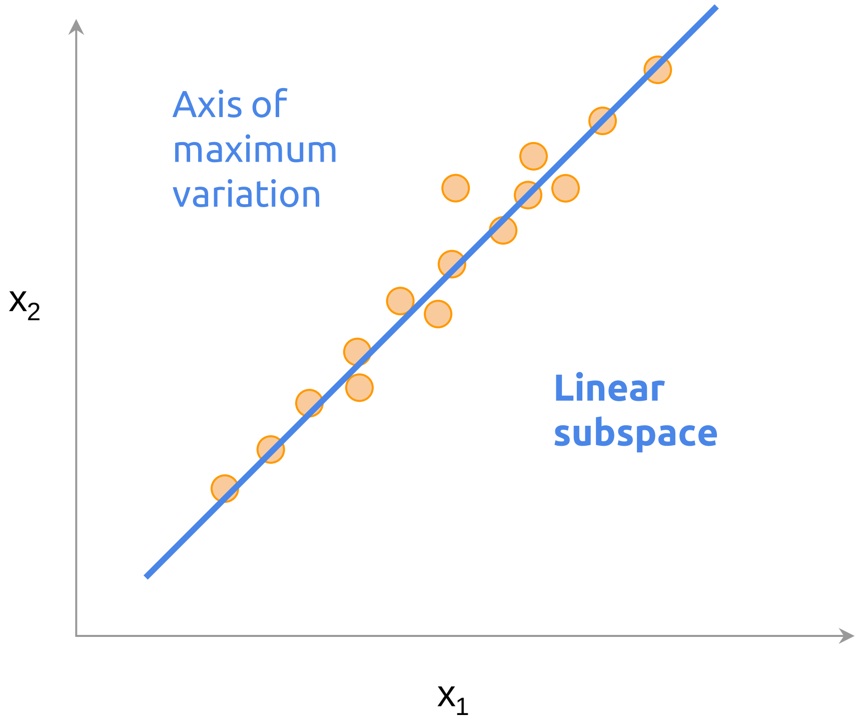
- 2D/3D visualization of high-dimensional data
- Compression
- Decorrelating data, **whitening**
- Eliminating noise
- Supervised learning



Intuition

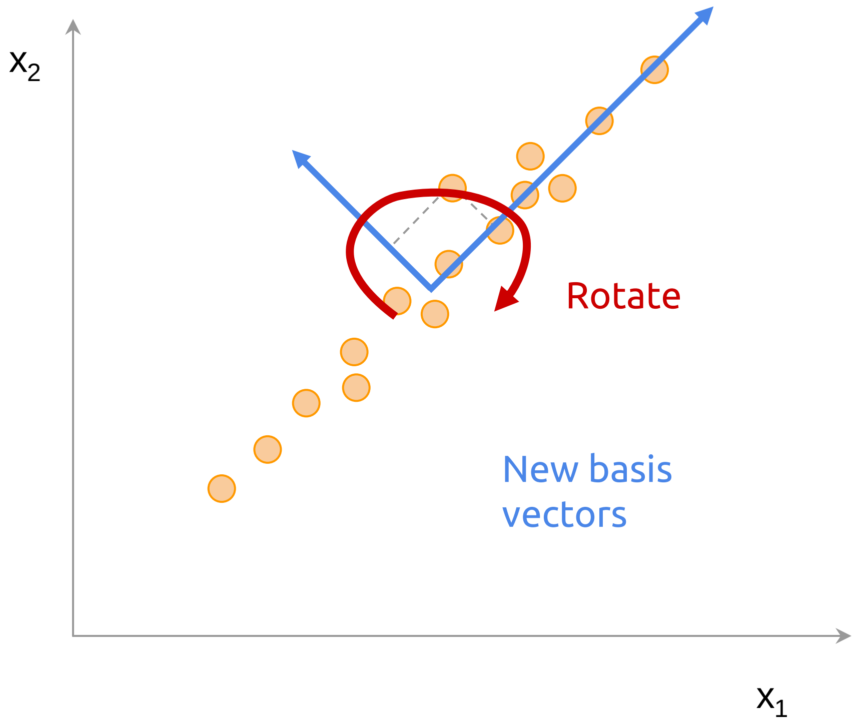


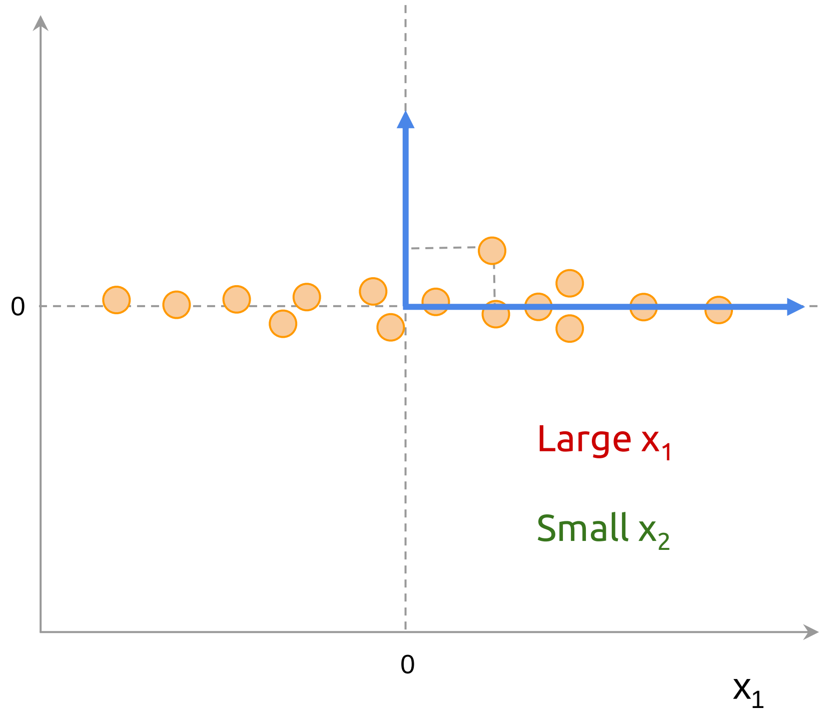
x_2
Large x_1
Large x_2
x_1



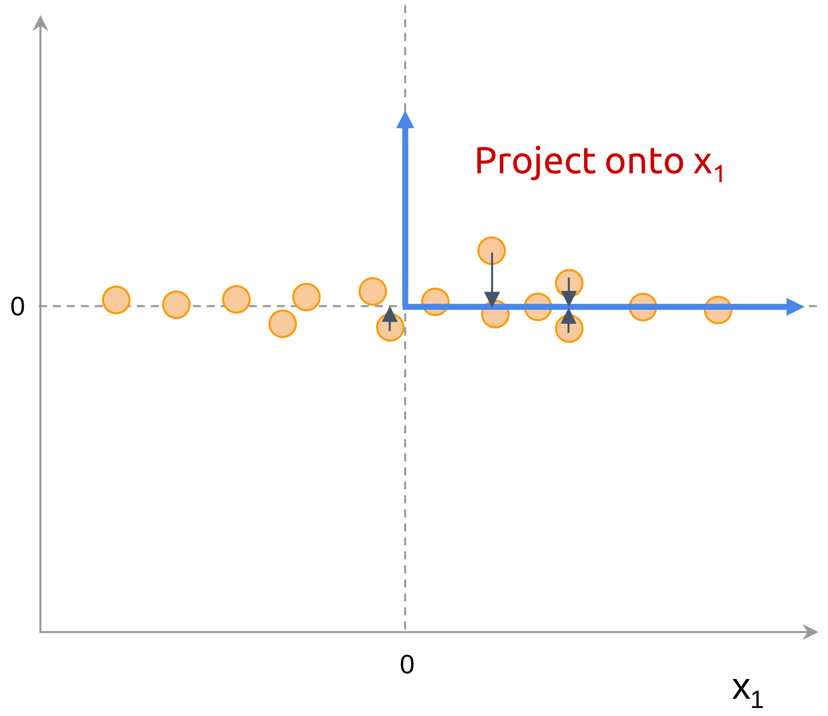
Axis of maximum variation
x_2
Linear subspace
x_1

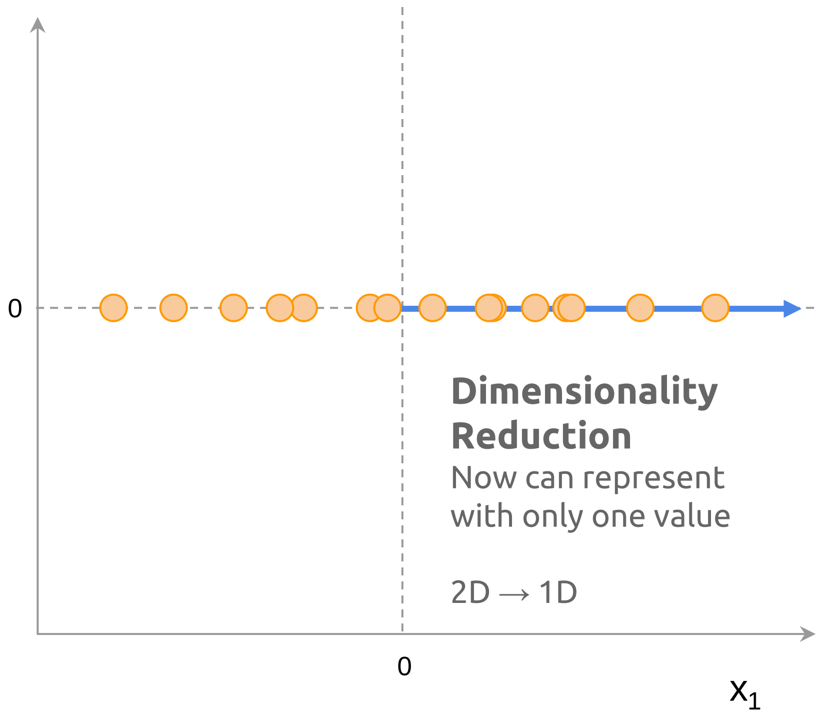
Intuition





Intuition

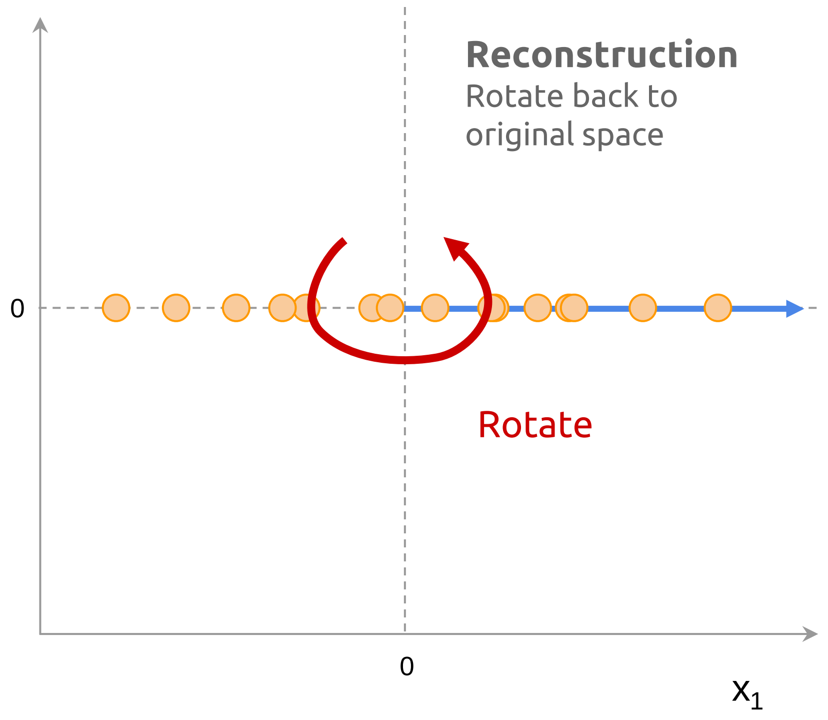


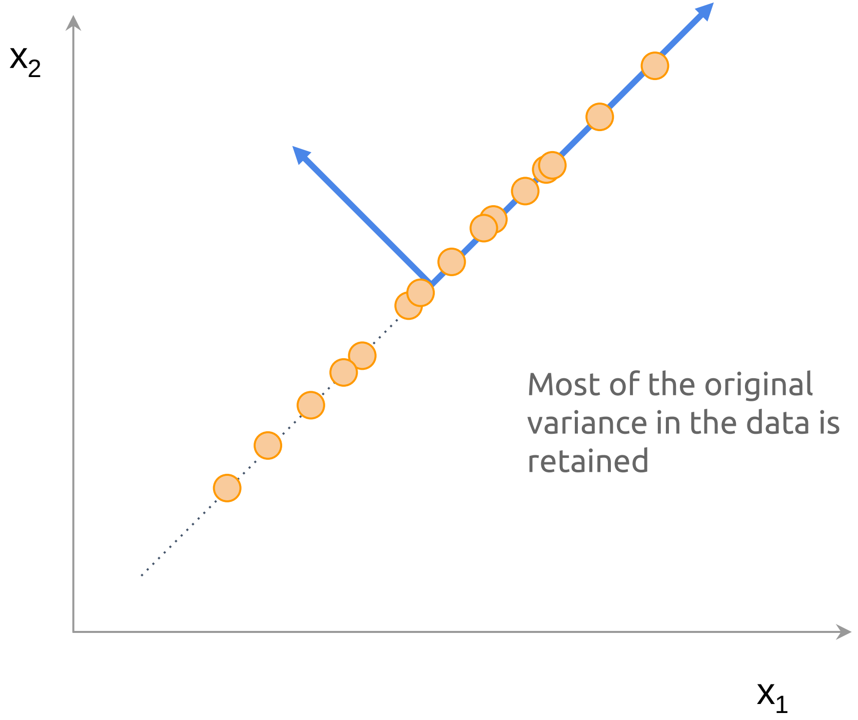


Dimensionality Reduction
Now can represent with only one value

2D → 1D

Intuition





Most of the original variance in the data is retained

PCA

PCA finds a **linear transform** (matrix) that **rotates** the data so that the directions of maximum variation in the data are aligned with the (x_1, x_2, …, x_n) axes.

The largest direction of variation is aligned with the x_1 axis, the second largest with the x_2 axis, and so on.

Truncating the dimension of the data after this rotation eliminates directions small variation and **reduces the dimensionality** of the data.

Reconstruction can be done by rotating the dimensionally reduced data back to its original coordinate system (basis).

Method
The PCA algorithm is based on Eigenvalue decomposition of the empirical covariance matrix of the data.

This matrix captures how the different dimensions of the data (e.g. x_1 and x_2) covary

PCA in code

```python
class PCA(object):
    def __init__(self, n=1):
        self.n = n

    def fit(self, X):

        # compute mean of the data and store
        self.mean = X.mean(axis=0)

        # subtract the mean
        X = X - self.mean

        # compute the empirical covariance matrix
        covariance = np.dot(X.T, X)

        # compute eigenvalues and eigenvectors
        vals, vecs = np.linalg.eig(covariance)
```

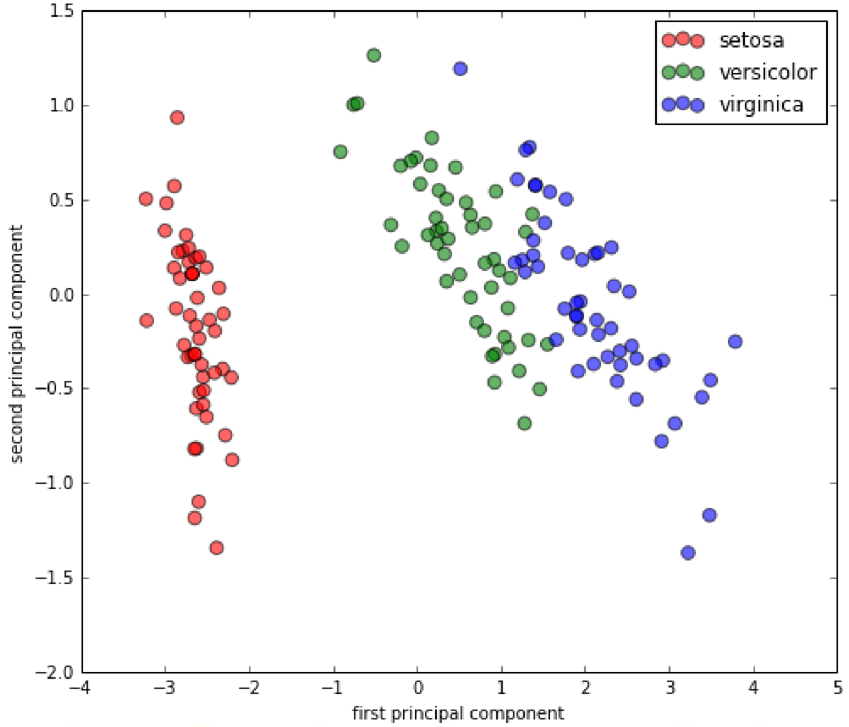
```python
        # sort eigenvalues and vectors by eigenvalue
        indices = np.argsort(-vals)
        vals = vals[indices]
        vecs = vecs[:, indices]

        # store eigenvalues and principal components
        self.eigenvalues = vals
        self.components = vecs[:, :self.n]
        return self

    def transform(self, X):
        return np.dot(X - self.mean, self.components)

    def inverse_transform(self, X):
        return np.dot(X, self.components.T) + self.mean
```

Example: visualization of 4D data

The famous Iris plants dataset (Fisher, 1950)

150 samples, 4 variables, 3 classes of plant

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	class
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
5	5.4	3.9	1.7	0.4	setosa
6	4.6	3.4	1.4	0.3	setosa



Application of PCA

Eigenfaces (1987)
Recognition of identities from images of people's faces

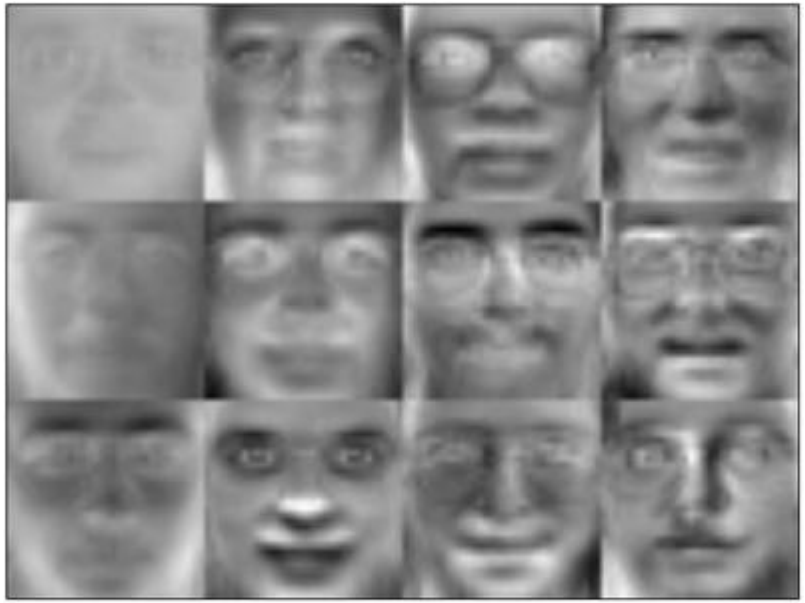
Images are very high dimensional data!

- 100 x 100 image = 10,000D

We would like a low-dimensional representation that captures most of the variation: PCA

We can then compare different images (using distance or similarity metric) in this low-dimensional space

(Much better ways to do this nowadays)



Latent semantic analysis (LSA)

PCA-like algorithms applied to document analysis

Documents can be represented as **bags of words**. Histograms of word counts in the document. These are very high-dimensional representations (1 dimension per word in vocabulary).

Applying PCA to this representation gives a lower-dimensional representation for a document that retains the main directions of variation

Assumption: documents about similar topics have similar terms (covariance).

LSA is used to create a low-dimensional **latent** representation that captures the main topics.

Can be used with k-means or agglomerative clustering to cluster documents with similar topics.

How many principal components?

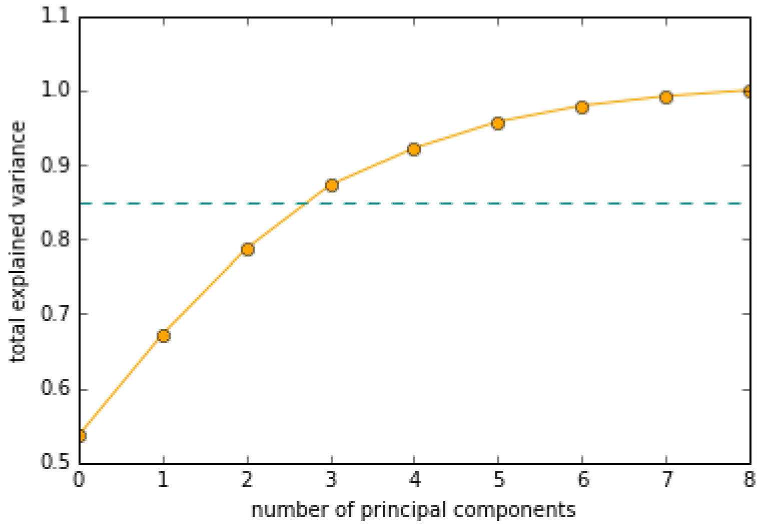
Depends on the task

- 2D visualization: 2 (obviously)

Typical heuristic:

- Take first *k* components that explain 85-90% of variance

More components: more variation explained, but possibly more noise



Overview

Motivation and goals of unsupervised learning

Clustering

- K-Means
- Agglomerative clustering

Dimensionality reduction

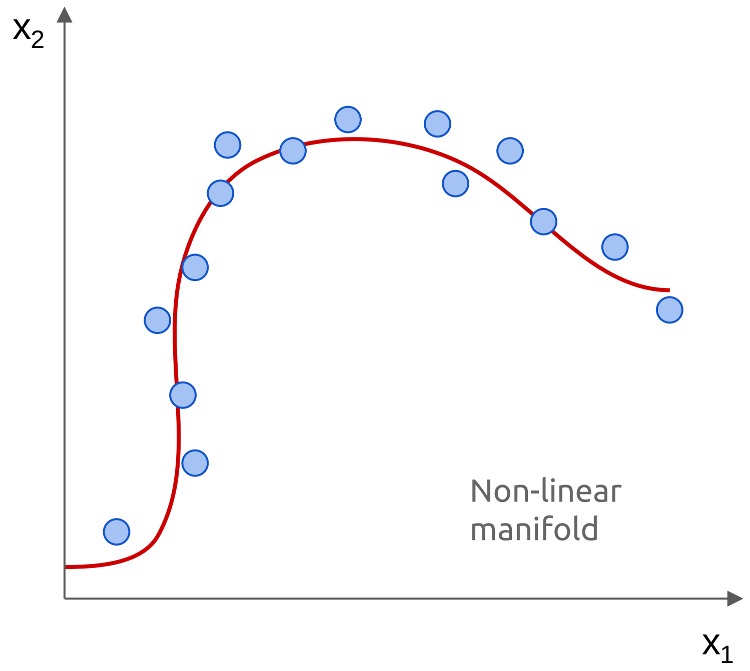
- PCA
- t-SNE
- Other methods

Nonlinear dimensionality reduction

PCA assumes that the data is near a linear manifold (hyperplane)

There are also techniques called **manifold learning** algorithms that can model nonlinear manifolds.

- Nonlinear autoencoders
- t-SNE
- Isomap
- Multidimensional scaling (MDS)



t-SNE

t-distributed stochastic neighbor embedding

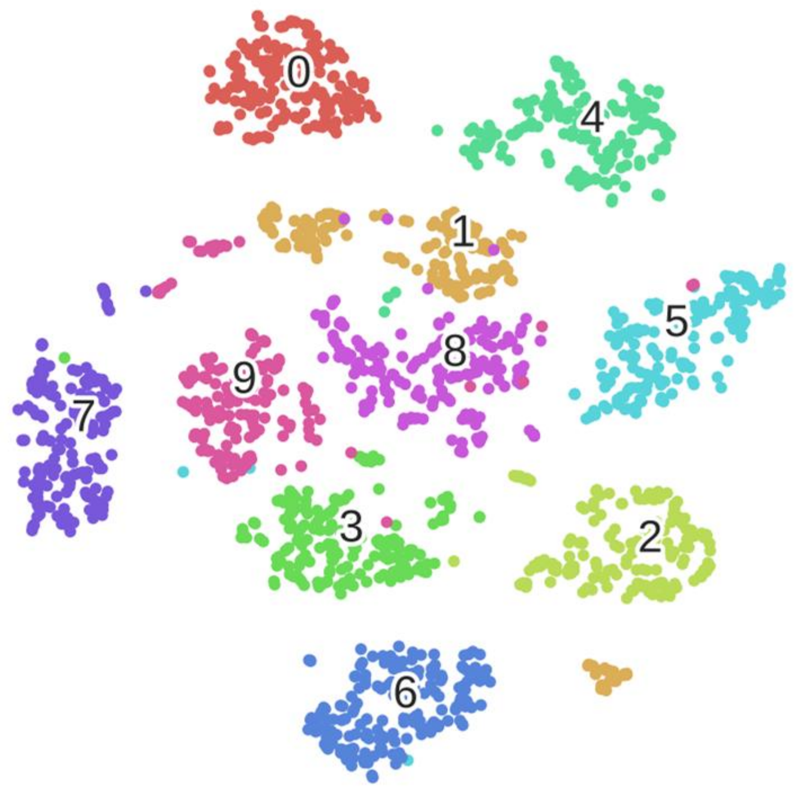
Mainly used as a data visualization technique.

Takes points in D dimensions and **embeds** them in a 2D or 3D space (map points).

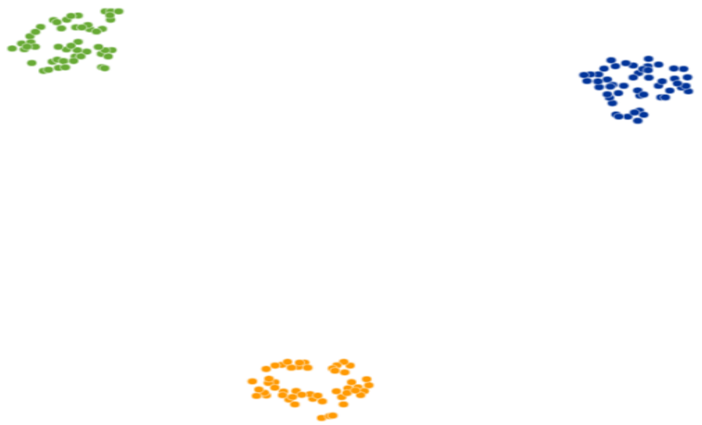
Starts off with a random (**stochastic**) embedding and iteratively improves it.

Attempts to preserve distances between nearby **neighbors**.

Uses the **Student's t distribution** to measure the similarity between points



t-SNE



t-SNE algorithm

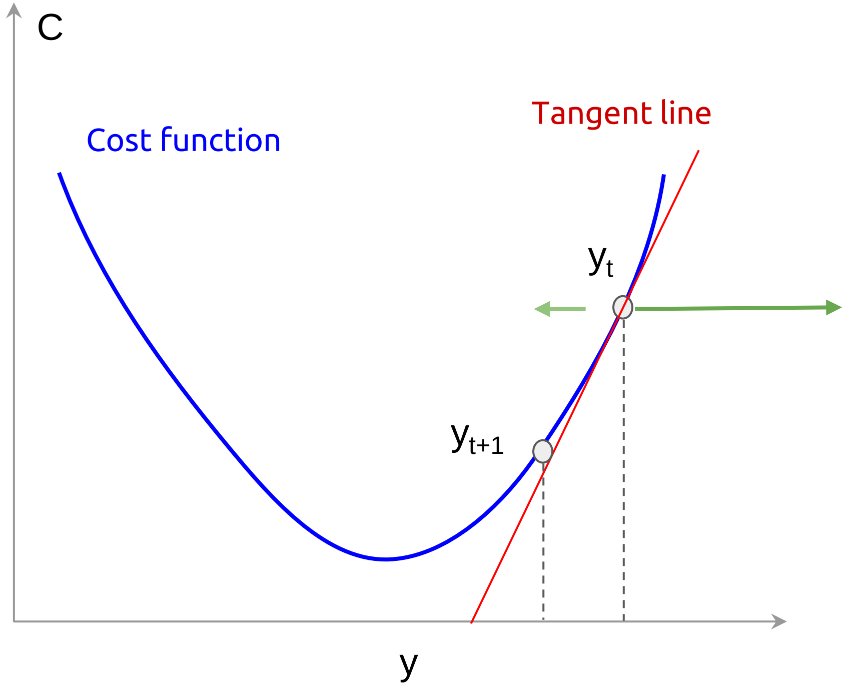
Method: **gradient descent**

Start with randomly chosen *y_i*

Iteratively improve *y_i* by computing gradient of cost function wrt. *y_i* and updating y by taking a step in the direction opposite to the gradient.

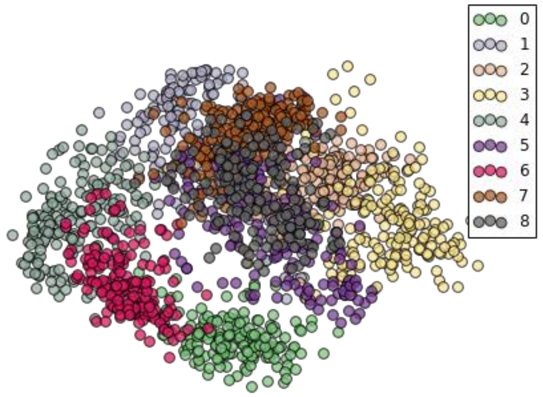
$$\frac{\partial C}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + ||y_i - y_j||^2)^{-1}$$

$$\mathbf{y}_{t+1} = \mathbf{y}_t - \alpha \nabla_{\mathbf{y}} C$$

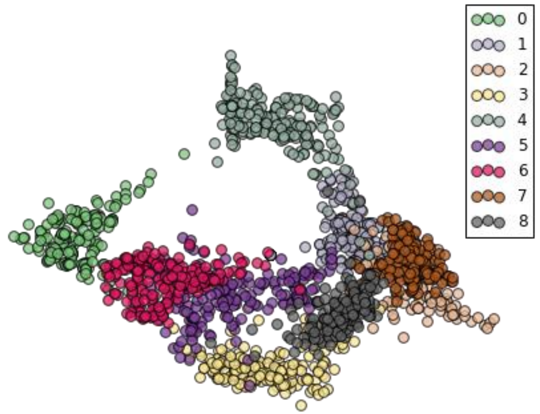


Visualization

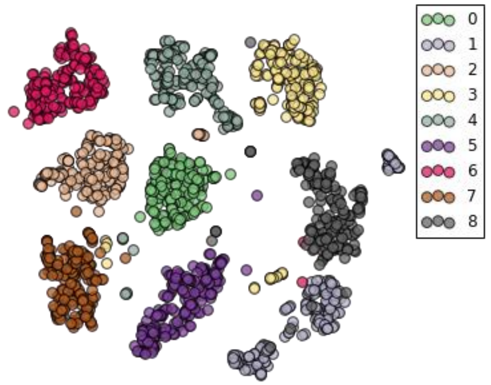
PCA



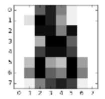
Isomap



t-SNE



Manifold learning for 8x8 (64D) images of handwritten digits (1800 samples)



t-SNE notes

Tends to work better than PCA for data visualization.

t-SNE is **stochastic**: you’ll get a different result every time you run it (unless you seed the random number generator)

Only transforms given data: cannot be used to transform new unseen data (unlike K-means and PCA). Mostly used as a visualization tool.

Standard algorithm can be computationally expensive. Faster approximations like Barnes-Hut exist.

Overview

Motivation and goals of unsupervised learning

Clustering

- K-Means
- Agglomerative clustering

Dimensionality reduction

- PCA
- t-SNE
- Other methods

Other approaches

Multidimensional scaling (MDS)

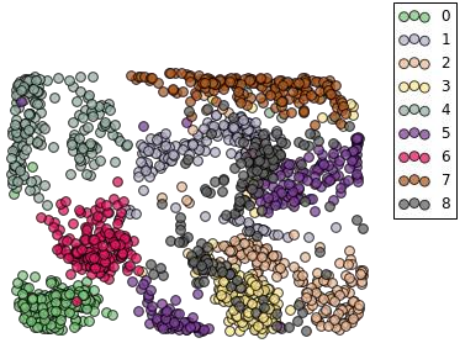
- Given **pairwise distance matrix**, find an embedding that preserves distances
- Based on Eigenvalue analysis of the **Gram matrix** (XX^T). Closely related to PCA

Isomap

- Extension of MDS: use neighbourhood graph and geodesic distances to create pairwise distance matrix.
- Solve with MDS

Autoencoders

- Minimize reconstruction error using gradient descent
- Linear autoencoders ~ PCA
- Lots of variants (sparse, denoising, deep)
- Unlike MDS/Isomap/TSNE, can be used to transform unseen data.
- Will see more about these in lecture on deep learning



Unsupervised learning with scikit-learn

Classes

- `sklearn.cluster.`**`KMeans`**
- `sklearn.cluster.`**`AgglomerativeClustering`**
- `sklearn.decomposition.`**`PCA`**
- `sklearn.manifold.`**`TSNE`**
- `sklearn.manifold.`**`MDS`**
- `sklearn.manifold.`**`Isomap`**
- `sklearn.mixture.`**`GaussianMixture`**

Methods

- **`fit`**`(X)`
- **`fit_transform`**`(X)`
- **`transform`**`(X)`
- **`predict`**`(X)`

```
from sklearn.datasets import load_digits
from sklearn.manifold import TSNE

digits = load_digits()
X, y = digits.data, digits.target

tsne = TSNE()
X_tsne = tsne.fit_transform(X)
```

Not all algorithms implement all these methods!

Further reading

The Elements of Statistical Learning, Chapter 14

- Introduction: 14.1
- Clustering: 14.3
  - K-Means: 14.3.6
  - Gap statistic: 14.3.11
  - Agglomerative clustering: 14.3.12
- PCA: 14.5.1