Statistical Natural Language Processing

Unsupervised machine learning

Çağrı Çöltekin

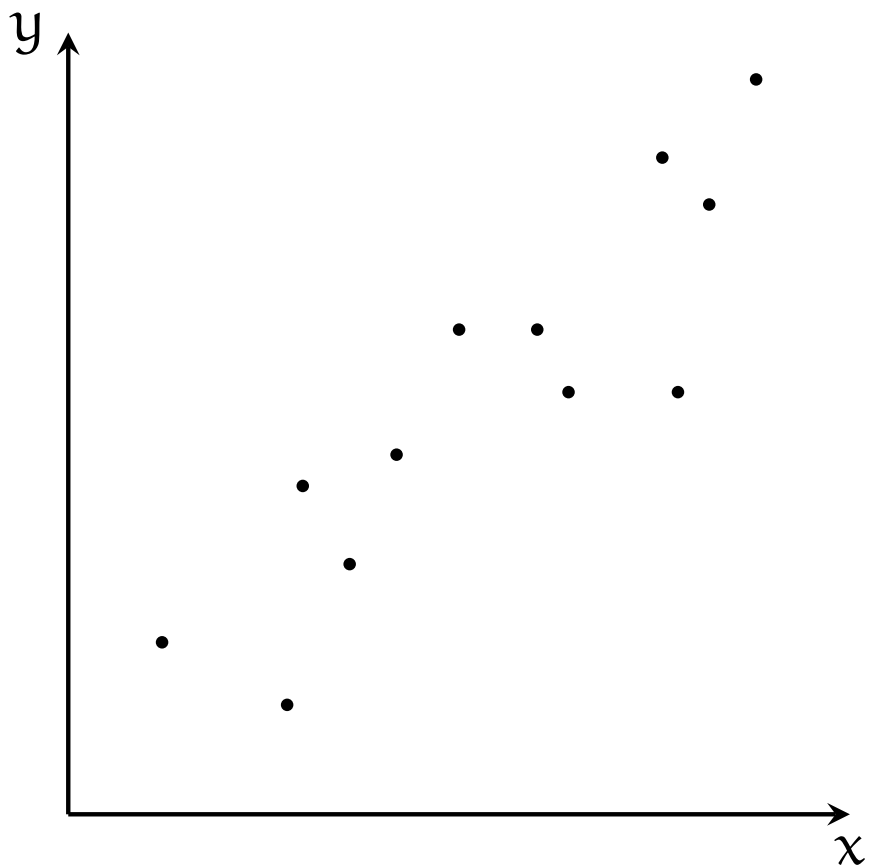
University of Tübingen
Seminar für Sprachwissenschaft

Summer Semester 2017

Supervised learning

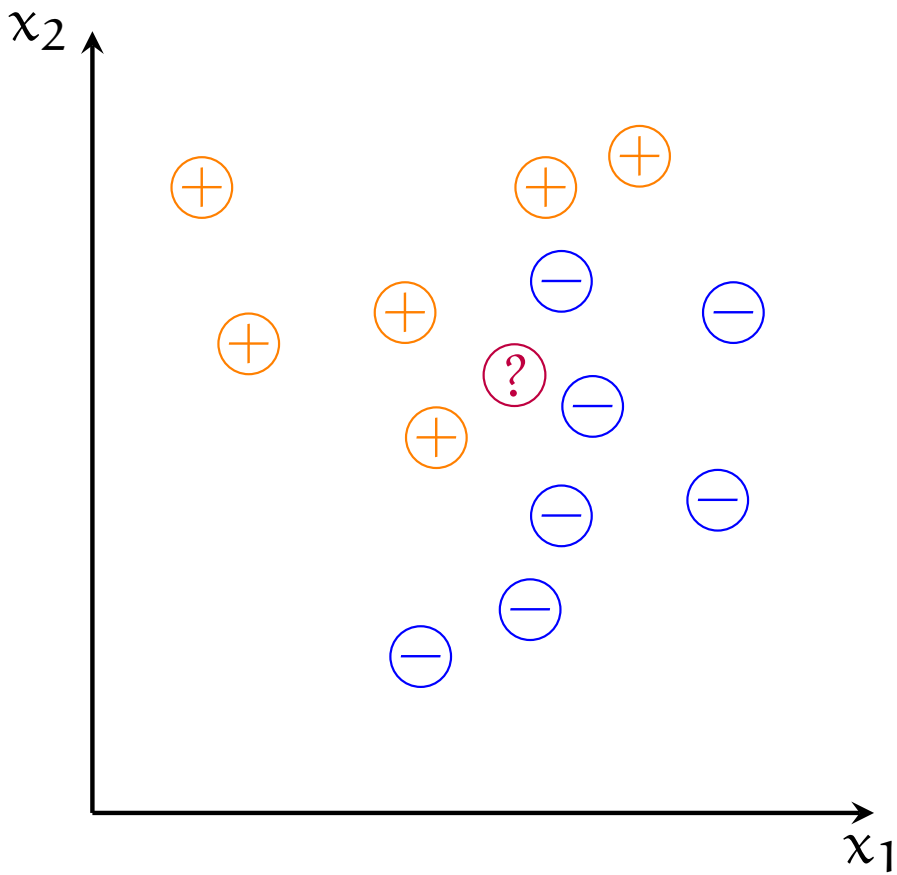
- The methods we studied so far are instances of supervised learning
- In supervised learning, we have a set of predictors $\mathbf{x}$, and want to predict a response or outcome variable $\mathbf{y}$
- During training, we have both input and output variables
- Training consist of estimating parameters $\boldsymbol{w}$ of a model
- During prediction, we are given $\mathbf{x}$ and make predictions based on model we learned

Supervised learning: regression



- The response (outcome) variable (y) is a quantitative variable.
- Given the features ($\mathbf{x}$) we want to predict the value of y

Supervised learning: classification



- The response (outcome) is a label. In the example: positive ⊕ or negative ⊖
- Given the features (x_1 and x_2), we want to predict the label of an unknown instance ?

Supervised learning: estimating parameters

- Most models/methods estimate a set of parameters $\boldsymbol{w}$ during training
- Often we find the parameters that minimize a loss function
  - For least-squares regression

$$J(w) = \sum_i (\hat{y}_i - y_i)^2 + \|\boldsymbol{w}\|$$

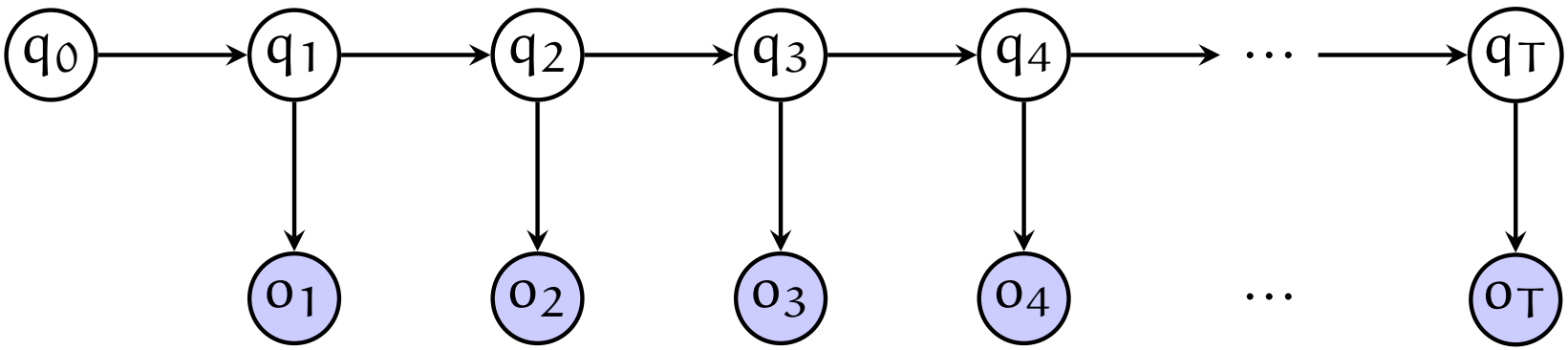
  - For logistic regression, the negative log likelihood

$$J(w) = -\log \mathcal{L}(w) + \|\boldsymbol{w}\|$$

- If the loss function is *convex*, we can find a *global* minimum using analytic solutions, otherwise use search methods such as *gradient descent*

Models with hidden variables

Hidden Markov models



- HMMs, or other models with hidden variables, can be learned without labels
- Unsupervised learning is essentially learning the hidden variables

Unsupervised learning

- In unsupervised learning, we do not have labels
- Our aim is to find useful patterns/structure in the data
- Typical unsupervised methods include
  - *Clustering*: find related groups of instances
  - *Density estimation*: find a probability distribution that explains the data
  - *Dimensionality reduction*: find an accurate/useful lower dimensional representation of the data
- All can be cast as graphical models with hidden variables
- Evaluation is difficult: we do not have 'true' labels/values

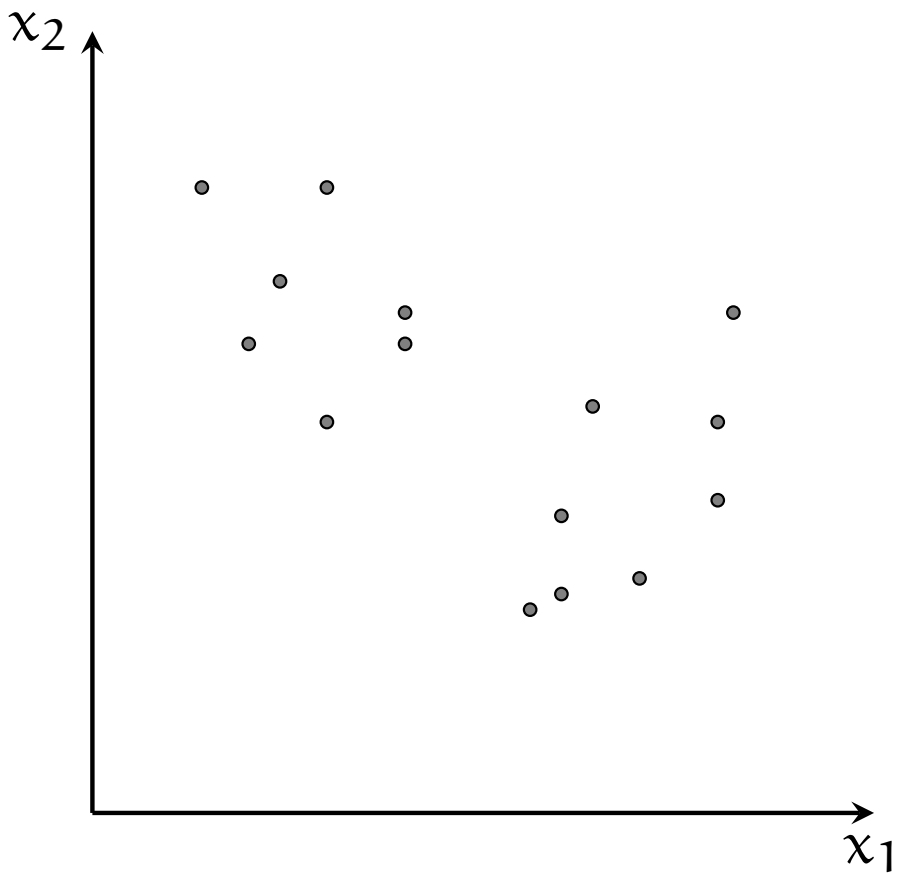
Clustering: why do we do it?

- The aim is to find groups of instances/items that are similar to each other
- Applications include
  - Clustering languages, dialects for determining their relations
  - Clustering (literary) texts, for e.g., authorship attribution
  - Clustering words for e.g., better parsing
  - Clustering documents, e.g., news into topics
  - …

Clustering

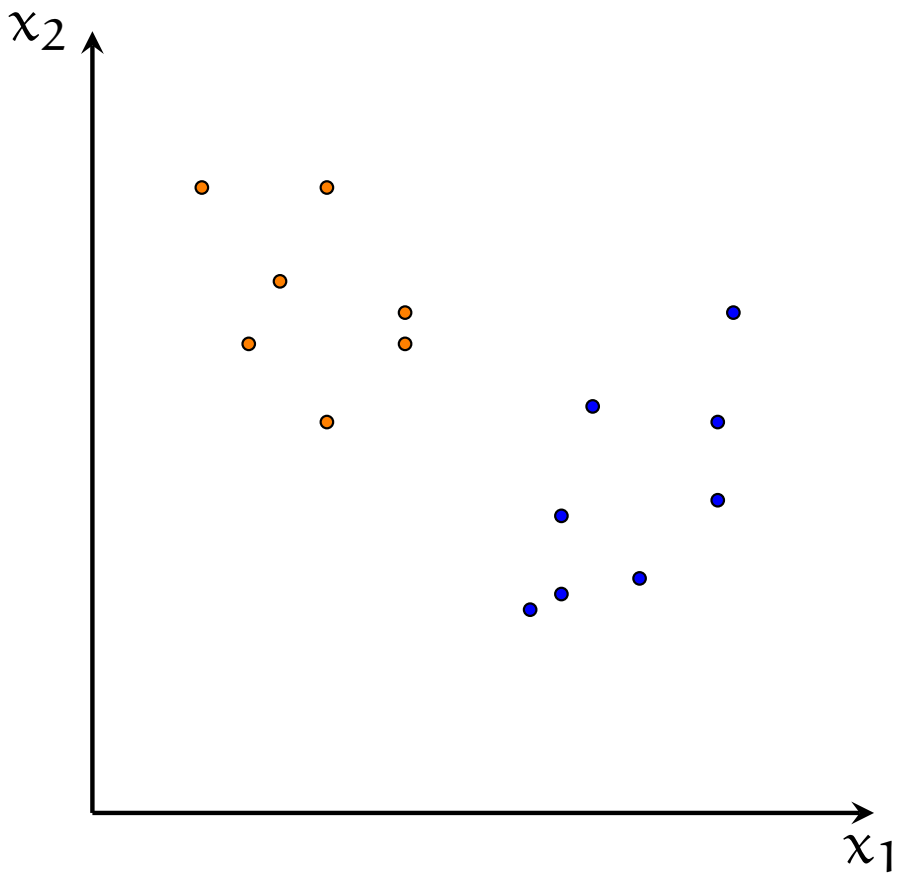
- Clustering can be *hierarchical* or non-hierarchical
- Clustering can be *bottom-up* (agglomerative) or top-down (divisive)
- For most (useful) problems we cannot find globally optimum solutions, we often rely on greedy algorithms that find a local minimum
- The measure of distance or similarity between the items is important

Clustering in two dimensional space



- Unlike classification, we do not have labels

Clustering in two dimensional space



- Unlike classification, we do not have labels
- We want to find 'natural' groups in the data
- Intuitively, similar or closer data points are grouped together

Similarity and distance

- The notion of distance (similarity) is important in clustering. A distance measure D,
  - is symmetric: $D(a, b) = D(b, a)$
  - non-negative: $D(a, b) \geqslant 0$
    for all a, b, and it $D(a, b) = 0$ iff $a = b$
  - obeys triangle inequality: $D(a, b) + D(b, c) \geqslant D(a, c)$
- The choice of distance is application specific
- We will often face with defining distance measures between linguistic units (letters, words, sentences, documents, ...)

Distance measures in Euclidean space

- Euclidean distance:

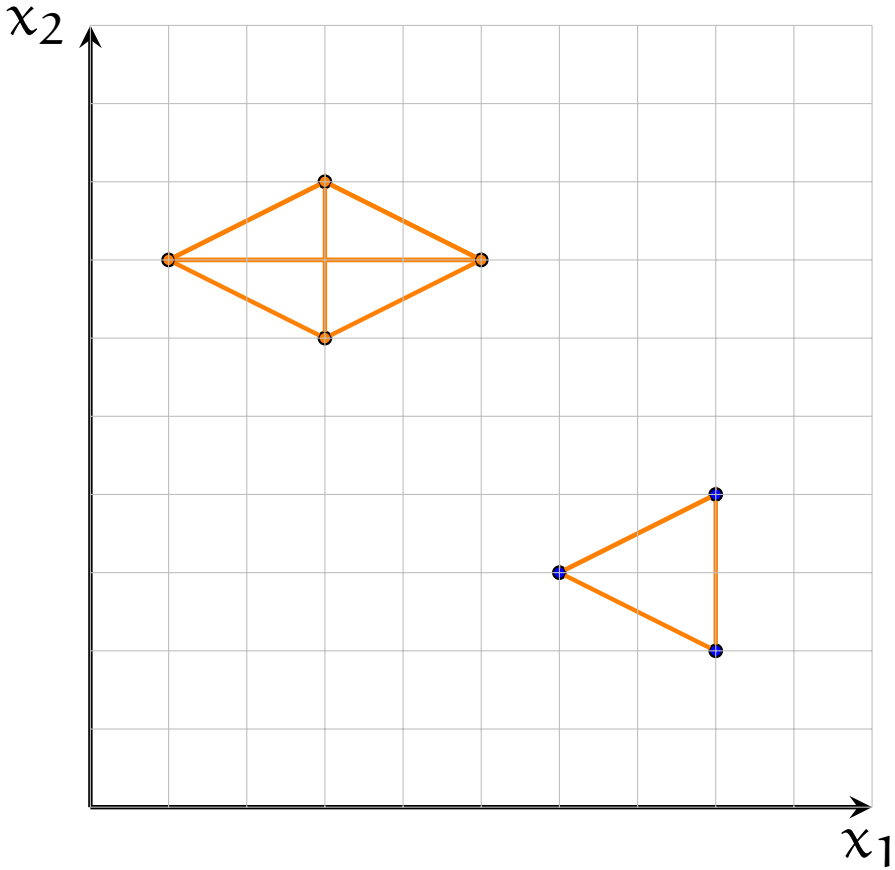
$$\|\mathbf{a} - \mathbf{b}\| = \sqrt{\sum_{j=1}^{k} (a_j - b_j)^2}$$

- Manhattan distance:

$$\|\mathbf{a} - \mathbf{b}\|_1 = \sum_{j=1}^{k} |a_j - b_j|$$

How to do clustering

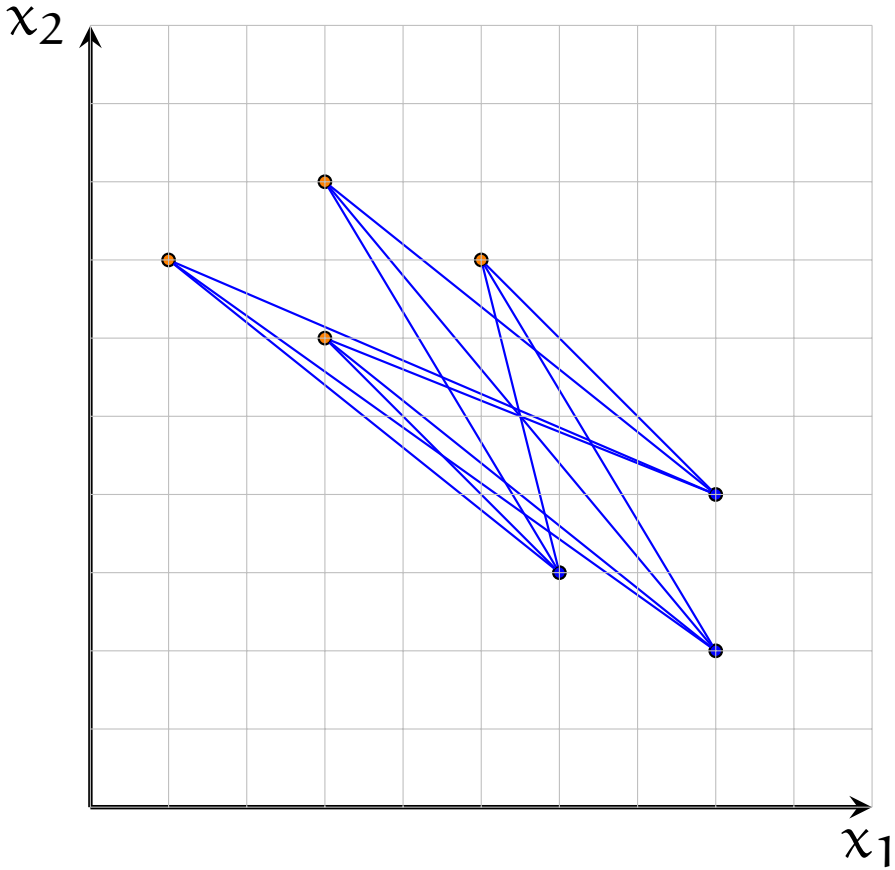
Most clustering algorithms try to minimize the scatter within each cluster. Which is equivalent to maximizing the scatter between clusters



$$\sum_{k=1}^{K} \sum_{C(a)=k} \sum_{C(b)=k} d(a,b)$$

How to do clustering

Most clustering algorithms try to minimize the scatter within each cluster. Which is equivalent to maximizing the scatter between clusters

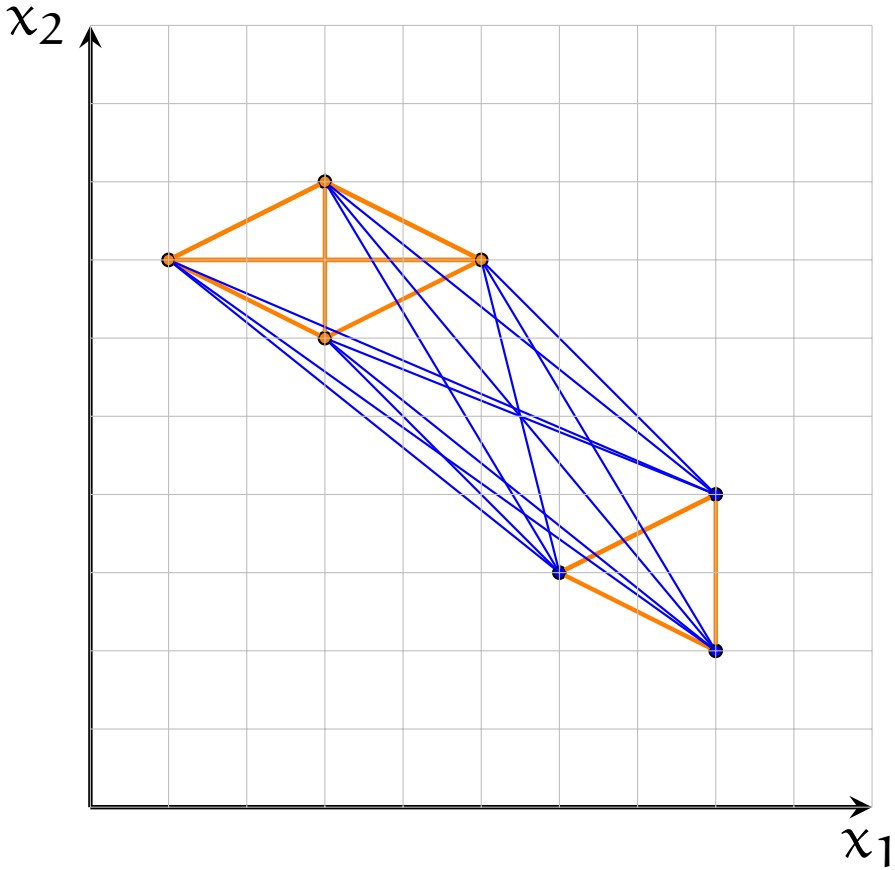


$$\sum_{k=1}^{K} \sum_{C(a)=k} \sum_{C(b)=k} d(a,b)$$

$$\sum_{k=1}^{K} \sum_{C(a)=k} \sum_{C(b)\neq k} d(a,b)$$

How to do clustering

Most clustering algorithms try to minimize the scatter within each cluster. Which is equivalent to maximizing the scatter between clusters



$$\sum_{k=1}^{K} \sum_{C(a)=k} \sum_{C(b)=k} d(a,b)$$

$$\sum_{k=1}^{K} \sum_{C(a)=k} \sum_{C(b)\neq k} d(a,b)$$

K-means clustering

K-means is a popular method for clustering.

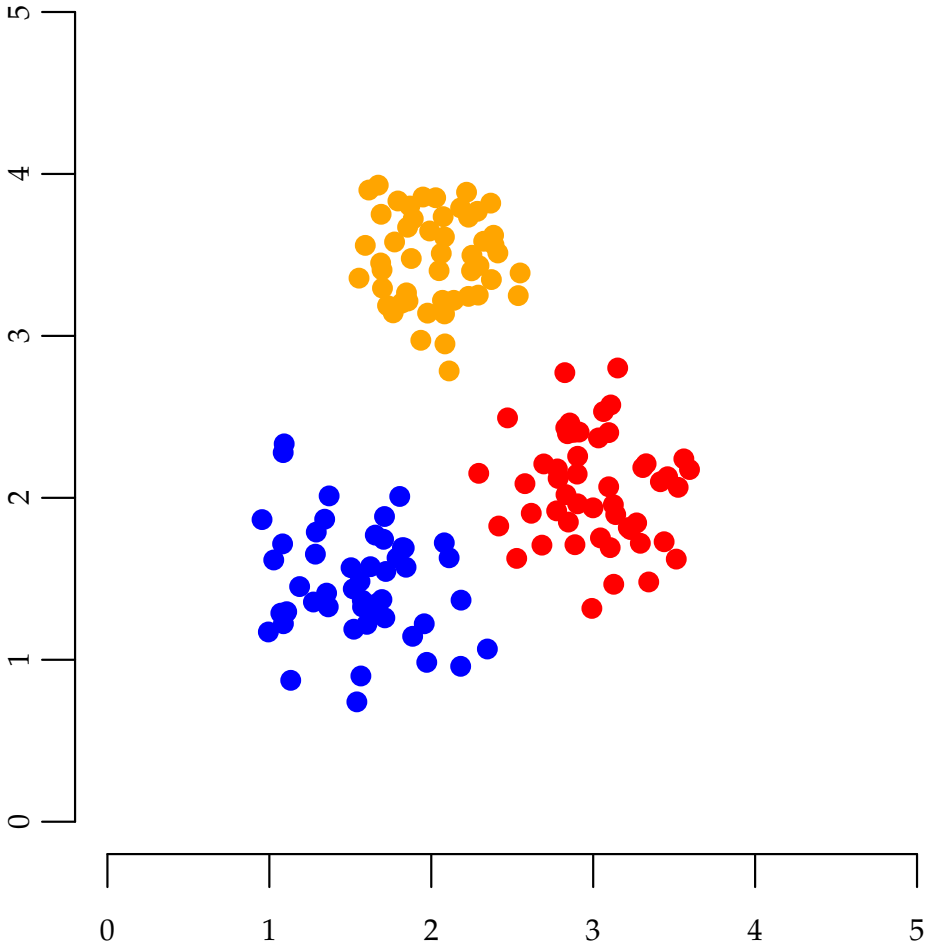
1. Randomly choose *centroids*, $m_1, \ldots, m_K$, representing K clusters
2. Repeat until convergence
   - Assign each data point to the cluster of the nearest centroid
   - Re-calculate the centroid locations based on the assignments

Effectively, we are finding a *local minimum* of the sum of squared Euclidean distance within each cluster

$$\frac{1}{2} \sum_{k=1}^{K} \sum_{C(a)=k} \sum_{C(b)=k} \|a - b\|^2$$

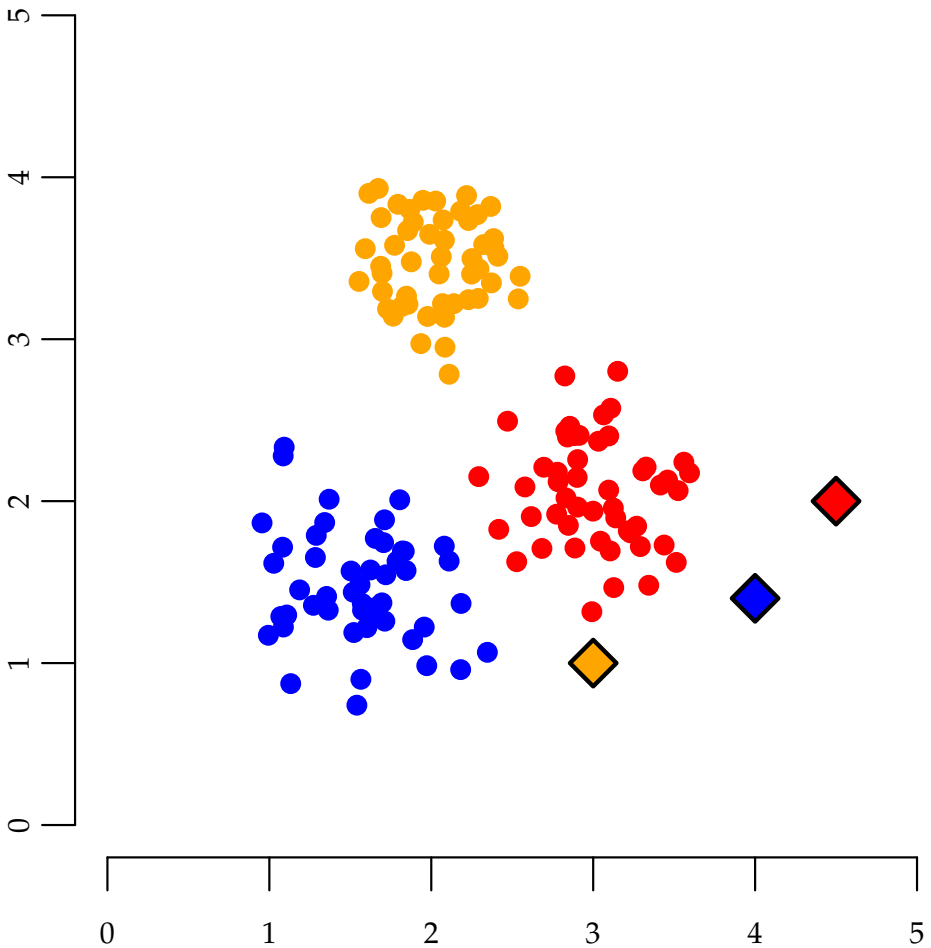
* Note the similarity with the EM algorithm

K-means clustering: visualization



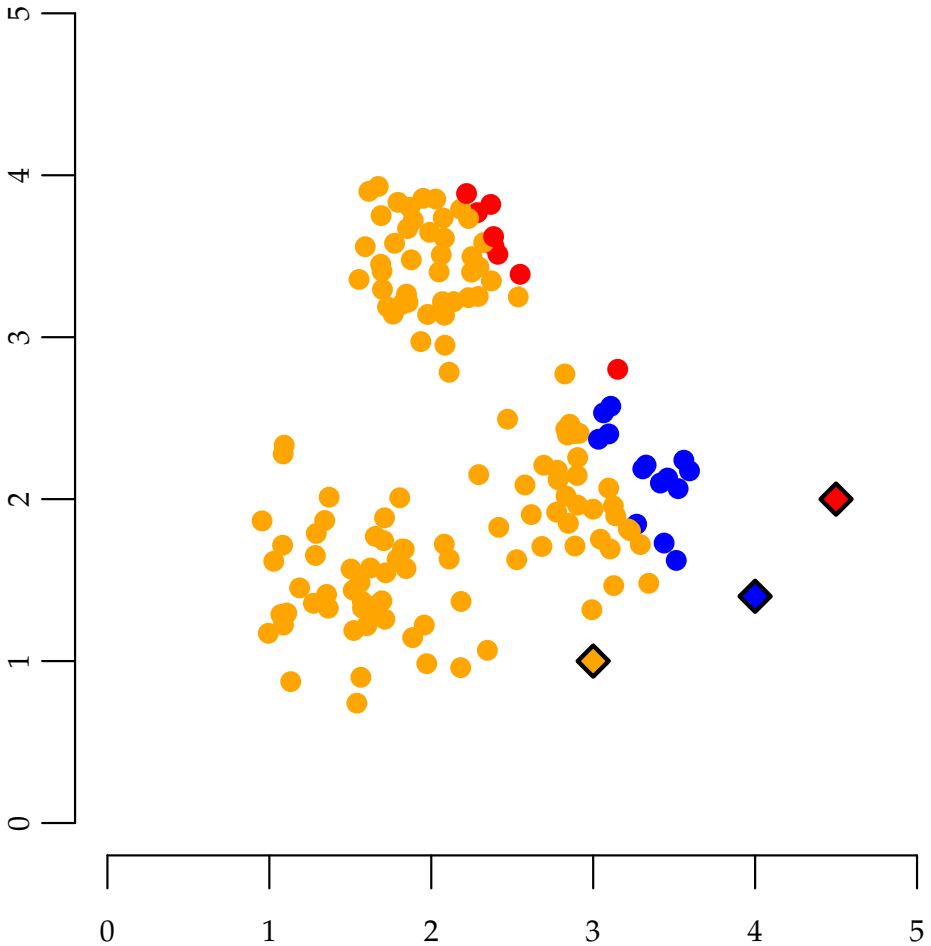
- The data
- Set cluster centroids randomly
- Assign data points to the closest centroid
- Recalculate the centroids

K-means clustering: visualization



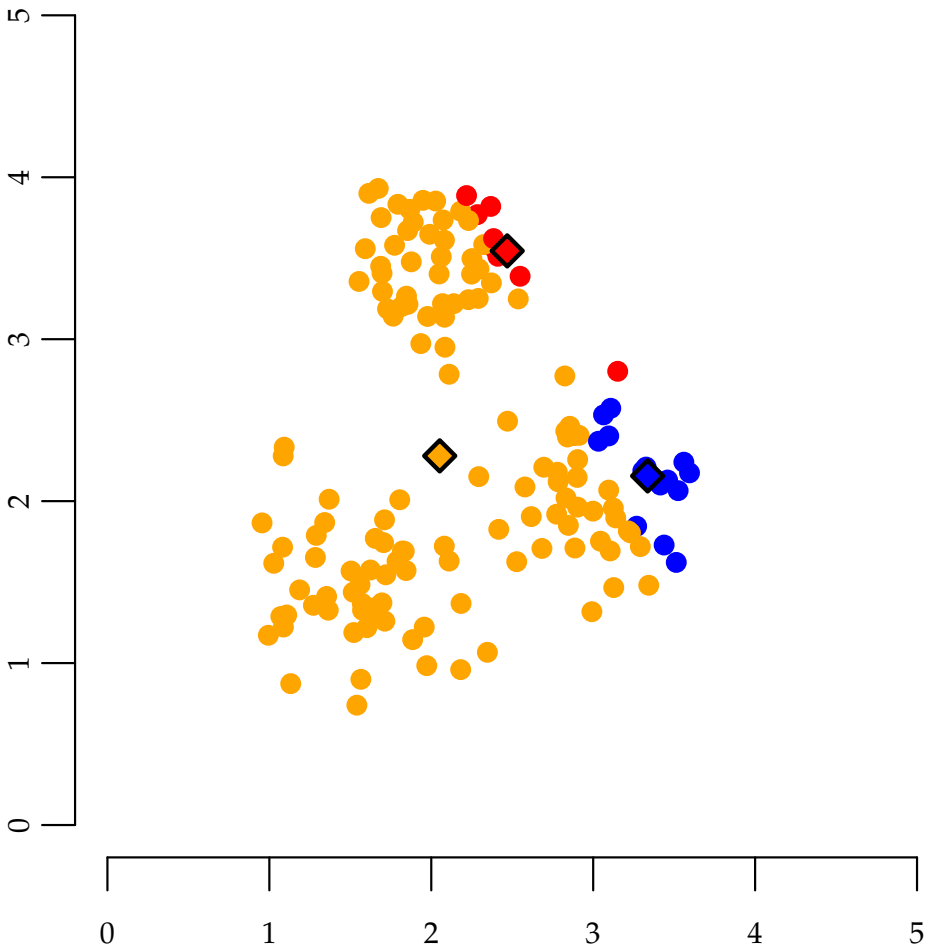
- The data
- Set cluster centroids randomly
- Assign data points to the closest centroid
- Recalculate the centroids

K-means clustering: visualization



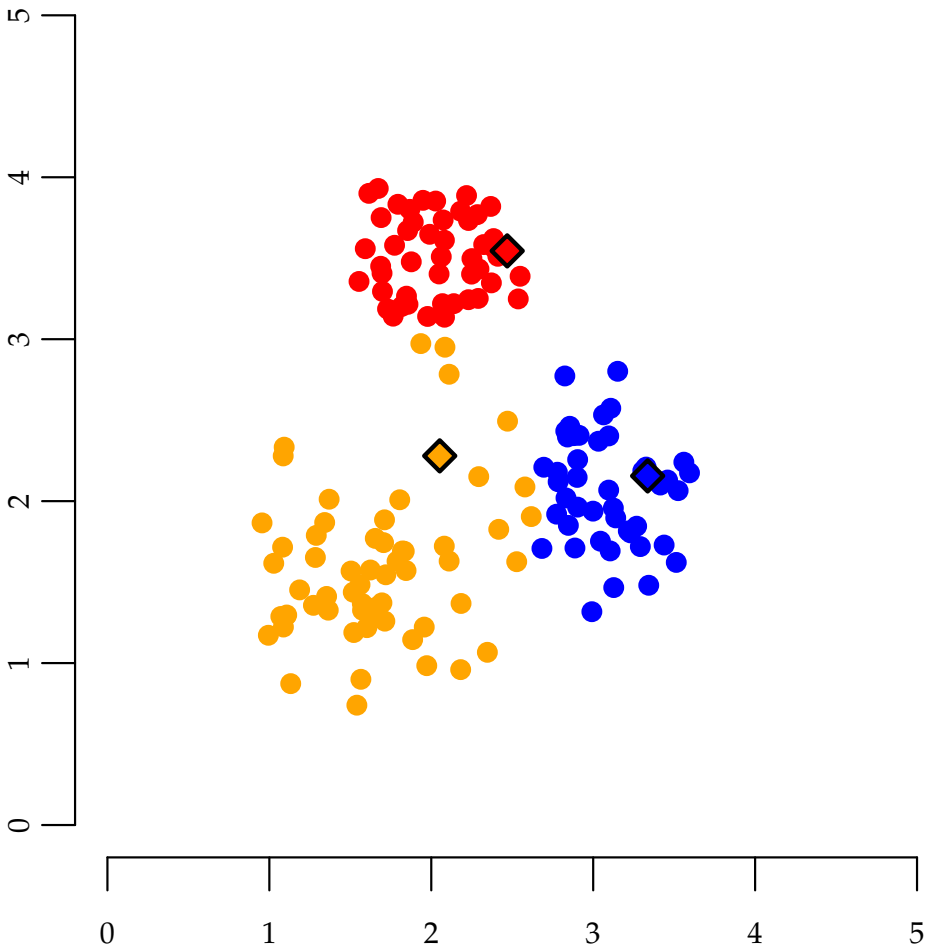
- The data
- Set cluster centroids randomly
- Assign data points to the closest centroid
- Recalculate the centroids

K-means clustering: visualization



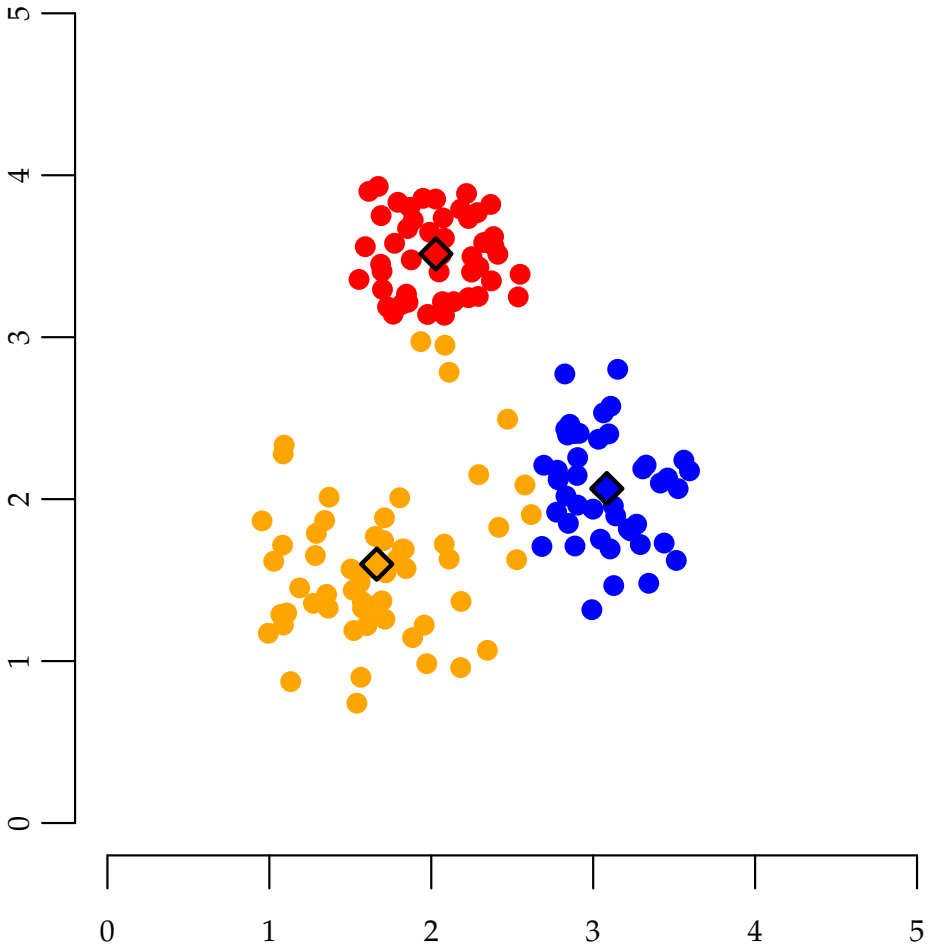
- The data
- Set cluster centroids randomly
- Assign data points to the closest centroid
- Recalculate the centroids

K-means clustering: visualization



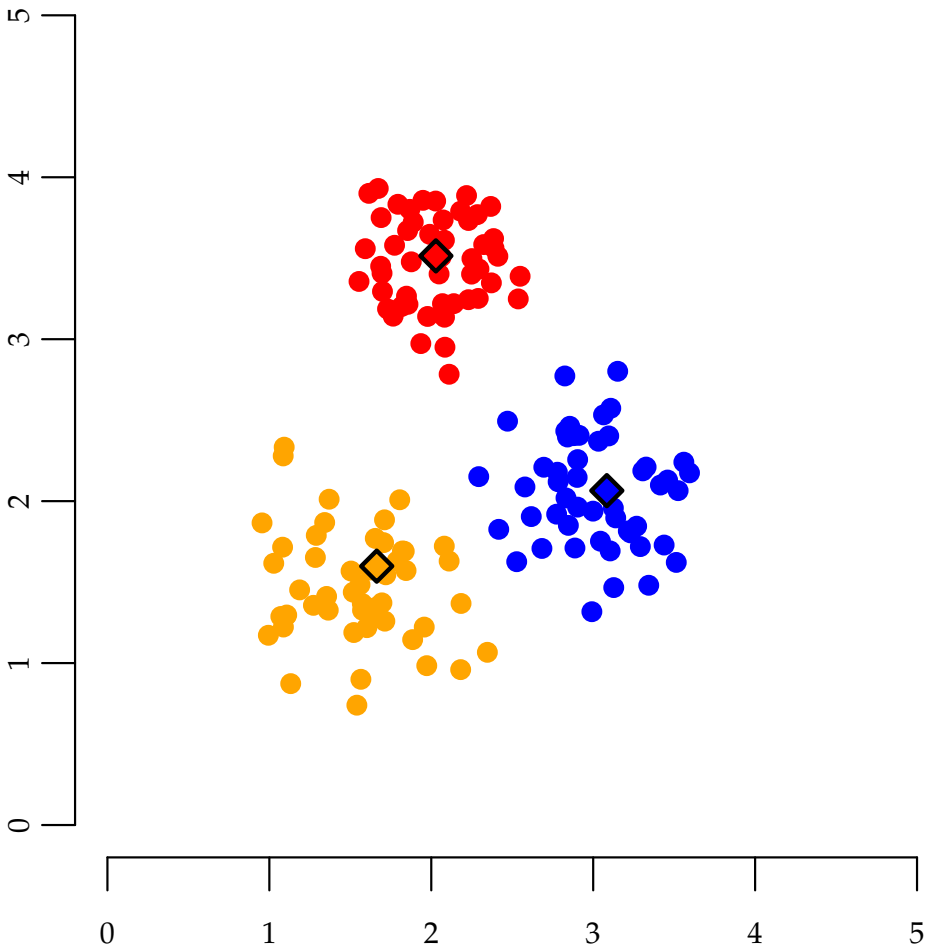
- The data
- Set cluster centroids randomly
- Assign data points to the closest centroid
- Recalculate the centroids

K-means clustering: visualization



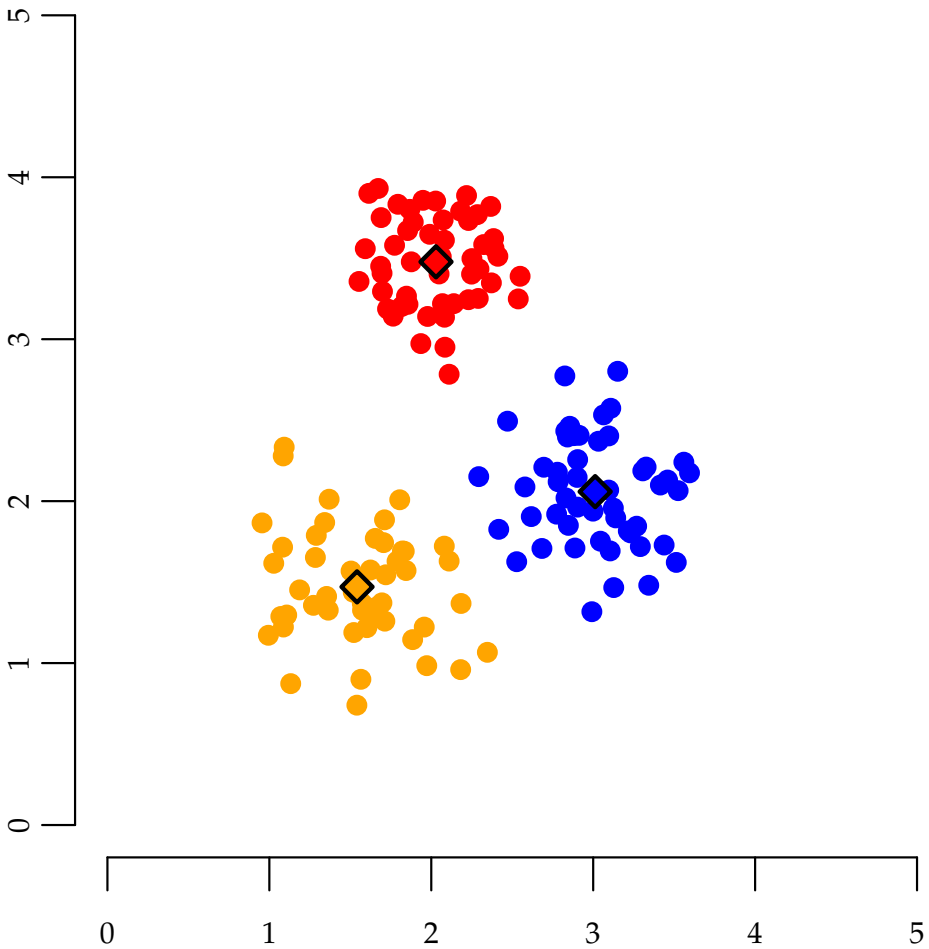
- The data
- Set cluster centroids randomly
- Assign data points to the closest centroid
- Recalculate the centroids

K-means clustering: visualization



- The data
- Set cluster centroids randomly
- Assign data points to the closest centroid
- Recalculate the centroids

K-means clustering: visualization



- The data
- Set cluster centroids randomly
- Assign data points to the closest centroid
- Recalculate the centroids

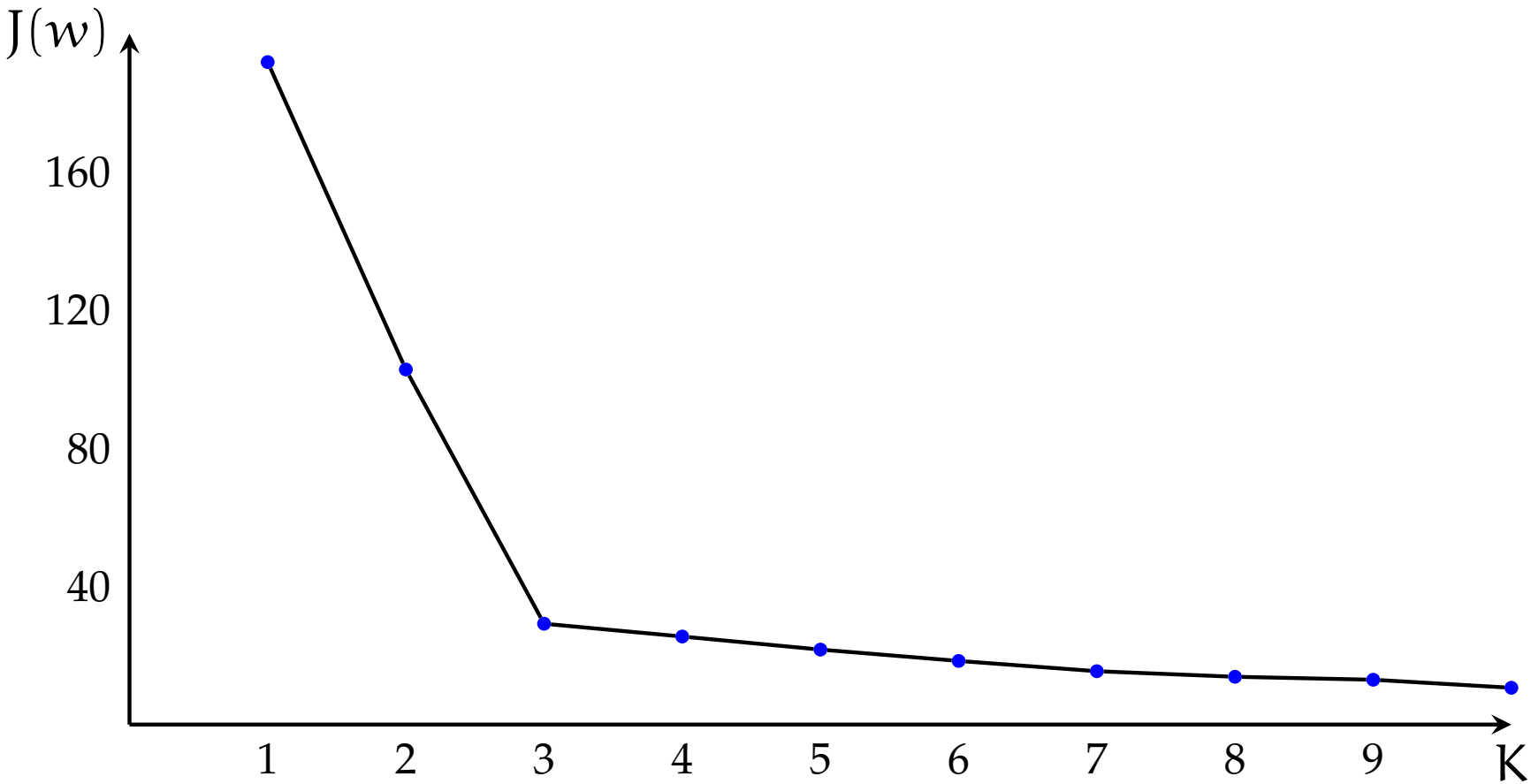
K-means: issues

- K-means requires the data to be in an Euclidean space
- K-means is sensitive to outliers
- The results are sensitive to initialization
  - There are some smarter ways to select initial points
  - One can do multiple initializations, and pick the best (with lowest within-group squares)
- It works well with approximately equal-size round-shaped clusters
- We need to specify number of clusters in advance

How many clusters?

- The number of clusters is defined for some problems, e.g., classifying news into a fixed set of topics/interests
- For others, there is no clear way to select the best number of clusters
- The error (within cluster scatter) always decreases with increasing number of clusters, using a test set or cross validation is not useful either
- A common approach is clustering for multiple K values, and picking where there is an 'elbow' in the graph against the error function

How many clusters?



This plot is sometimes called a *scree plot*.

K-medoids

- K-medoids algorithm is an alternation of K-means
- Instead of calculating centroids, we try to find most typical data point at each iteration
- K-medoids can work with distances, does not need feature vectors to be in an Euclidean space
- It is less sensitive to outliers
- It is computationally more expensive than K-means

Density estimation

- K-means treats all data points in a cluster equally
- A 'soft' version of K-means is density estimation for Gaussian mixtures, where
  - We assume the data comes from a mixture of K Gaussian distributions
  - We try to find the parameters of each distribution (instead of centroids) that maximizes the likelihood of the data
- Unlike K-means, mixture of Gaussians assigns probabilities for each data point belonging to one of the clusters
- It is typically estimated using the expectation-maximization (EM) algorithm

Density estimation using the EM algorithm

- The EM algorithm (or its variations) is used in learning models with latent/hidden variables
- It is closely related to the K-means algorithm

1. Initialize the parameters (e.g., randomly) of K multivariate normal distributions $(\boldsymbol{\mu}, \boldsymbol{\Sigma})$
2. Iterate until convergence:

E-step Given the parameters, compute the membership 'weights', the probability of each data point belonging to each distribution

M-step Re-estimate the mixture density parameters using the calculated membership weights in the E-step

Hierarchical clustering

- Instead of flat division to clusters as in K-means, hierarchical clustering builds a hierarchy based on similarity of the data points
- There are two main 'modes of operation':

Bottom-up or *agglomerative* clustering

- starts with individual data points,
- merges the clusters until all data is in a single cluster

Top-down or *divisive* clustering

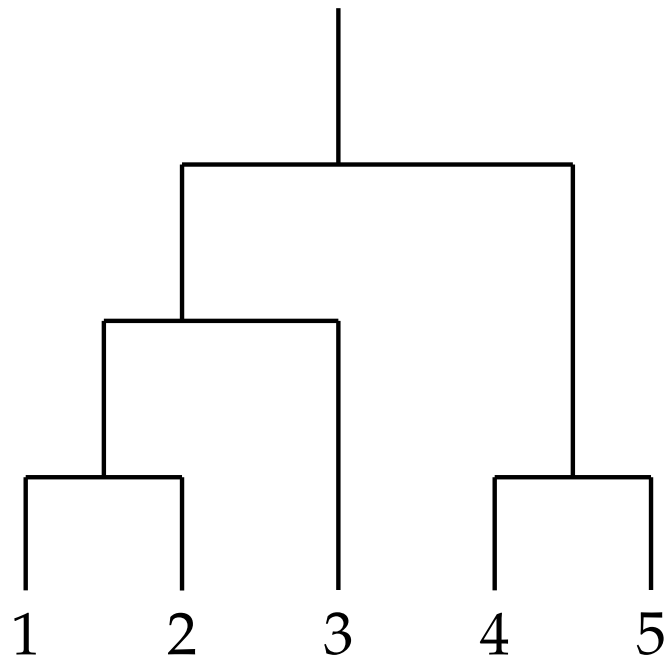
- starts with a single cluster,
- and splits until all leaves are single data points

Hierarchical clustering

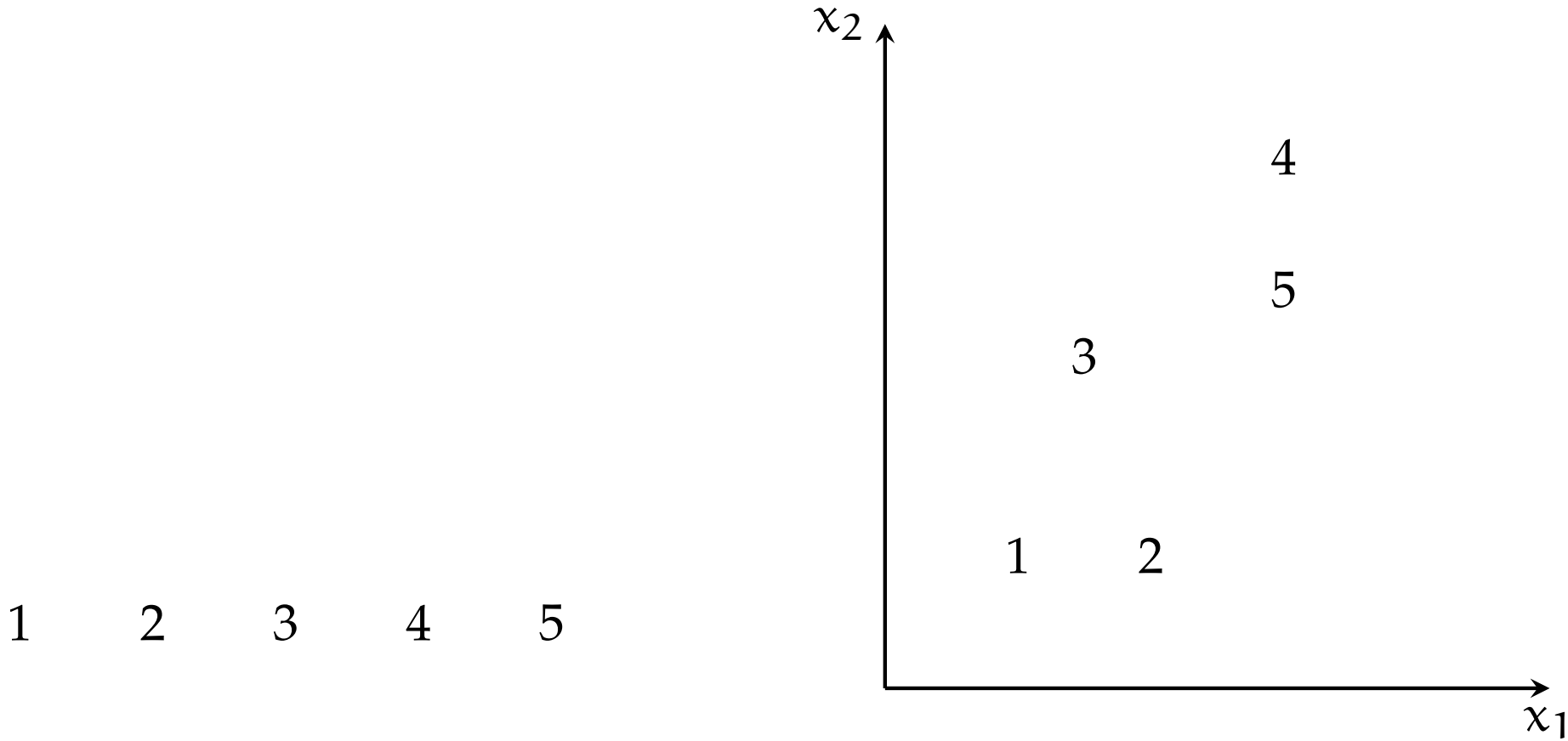
- Hierarchical clustering operates on differences
- The result is a binary tree called *dendrogram*
- Dendrograms are easy to interpret (especially if data is hierarchical)
- The algorithm does not commit to the number of clusters K from the start, the dendrogram can be 'cut' at any height for for determining the clusters

Agglomerative clustering

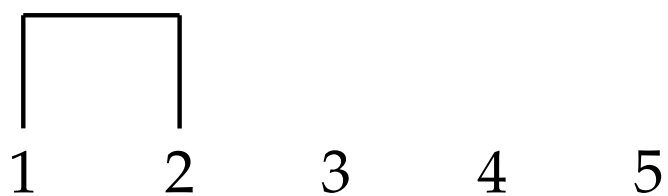
1. Compute the similarity/distance matrix
2. Assign each data point to its own cluster
3. Repeat until no clusters left to merge
   - Pick two clusters that are most similar to each other
   - Merge them into a single cluster

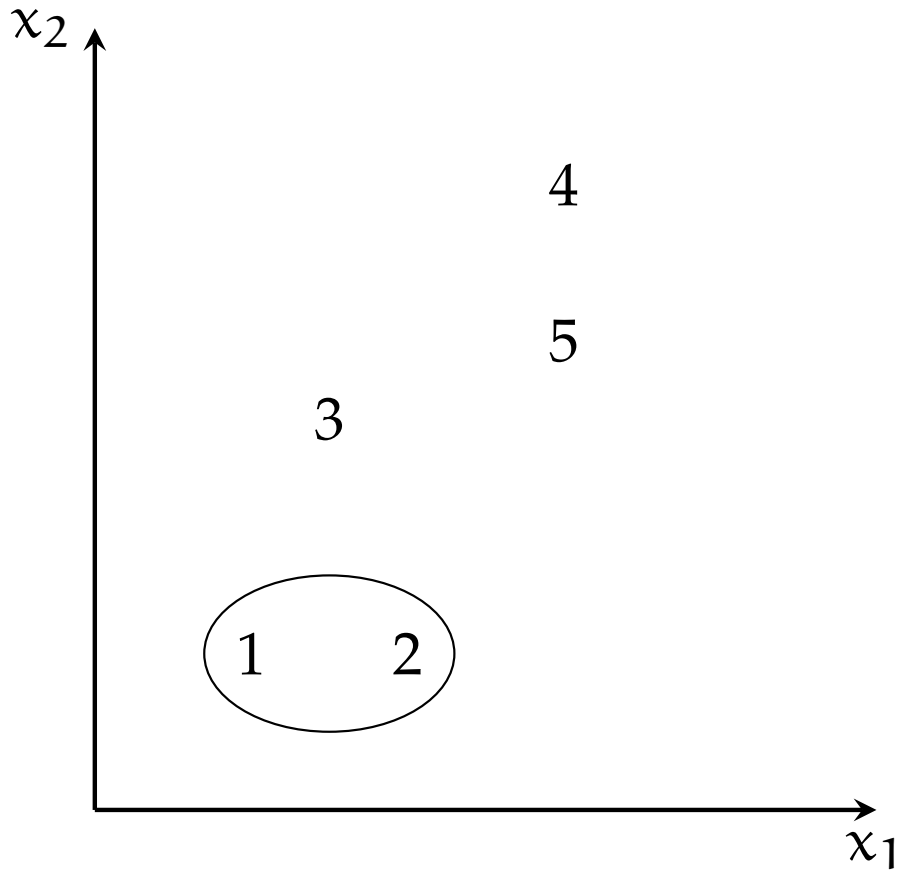


Agglomerative clustering demonstration

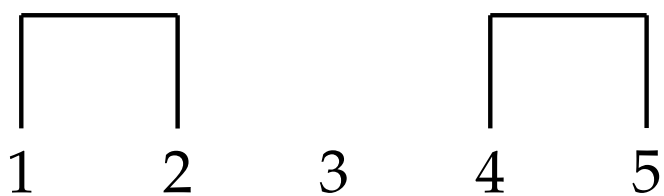


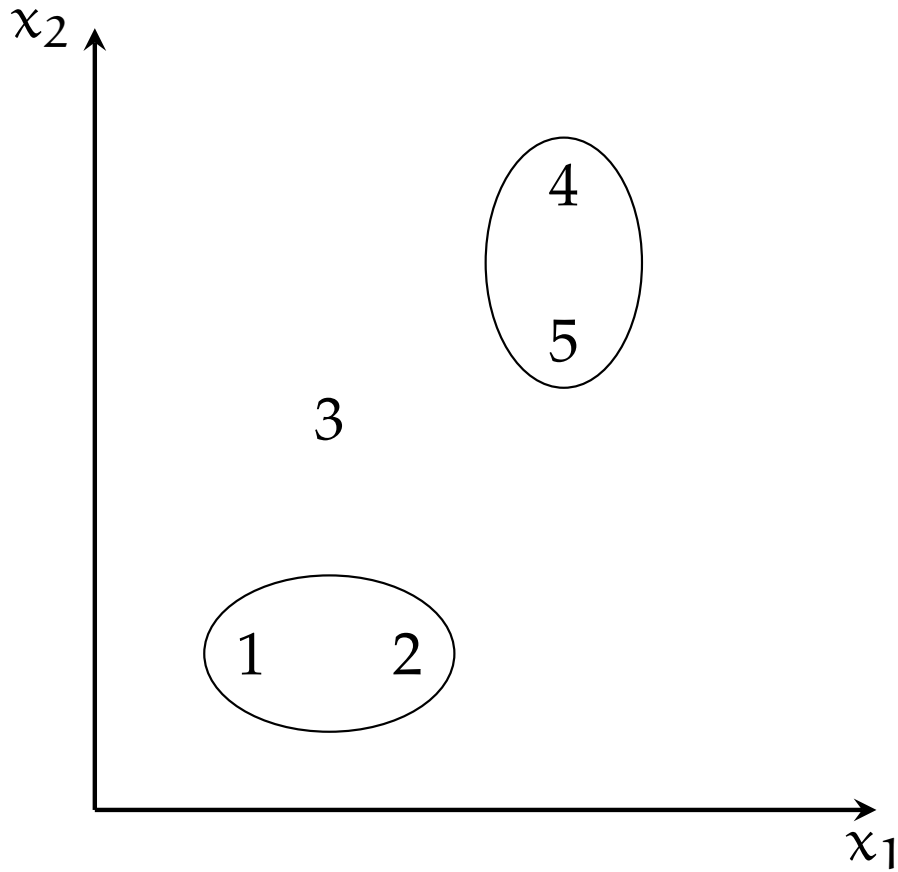
Agglomerative clustering demonstration



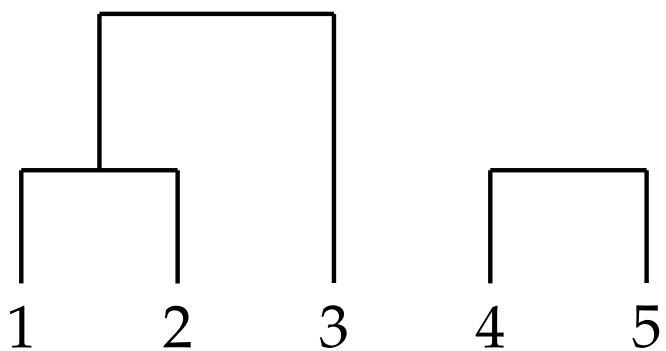


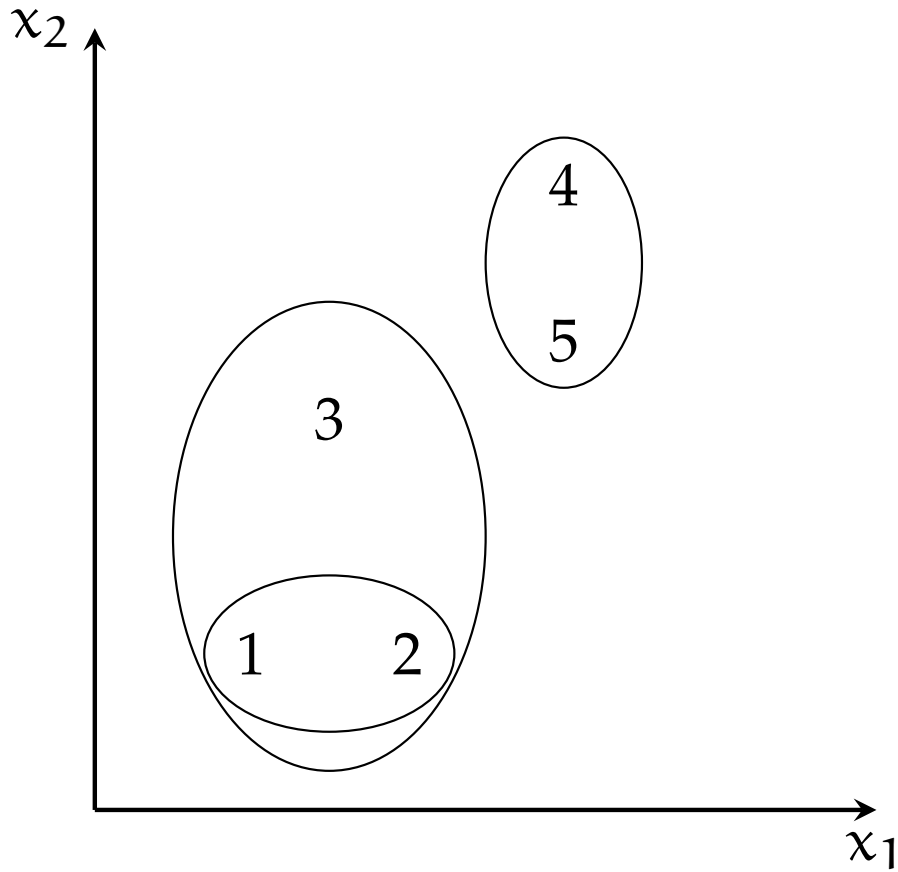
Agglomerative clustering demonstration



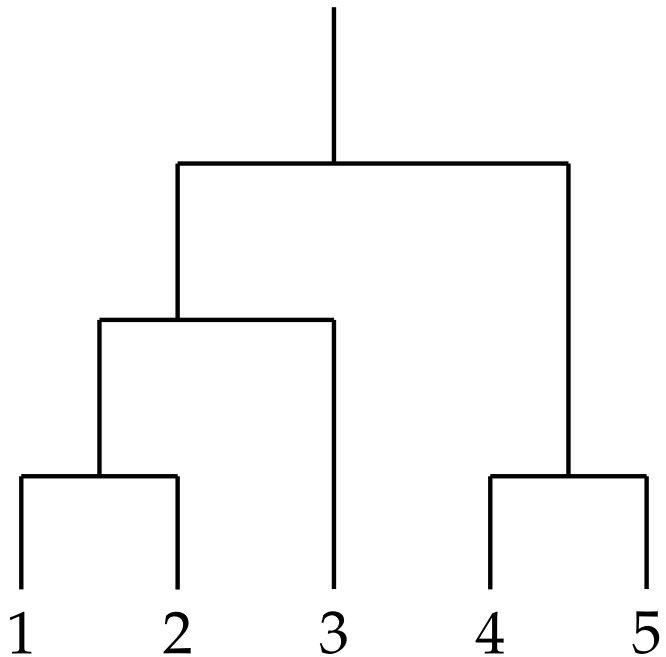


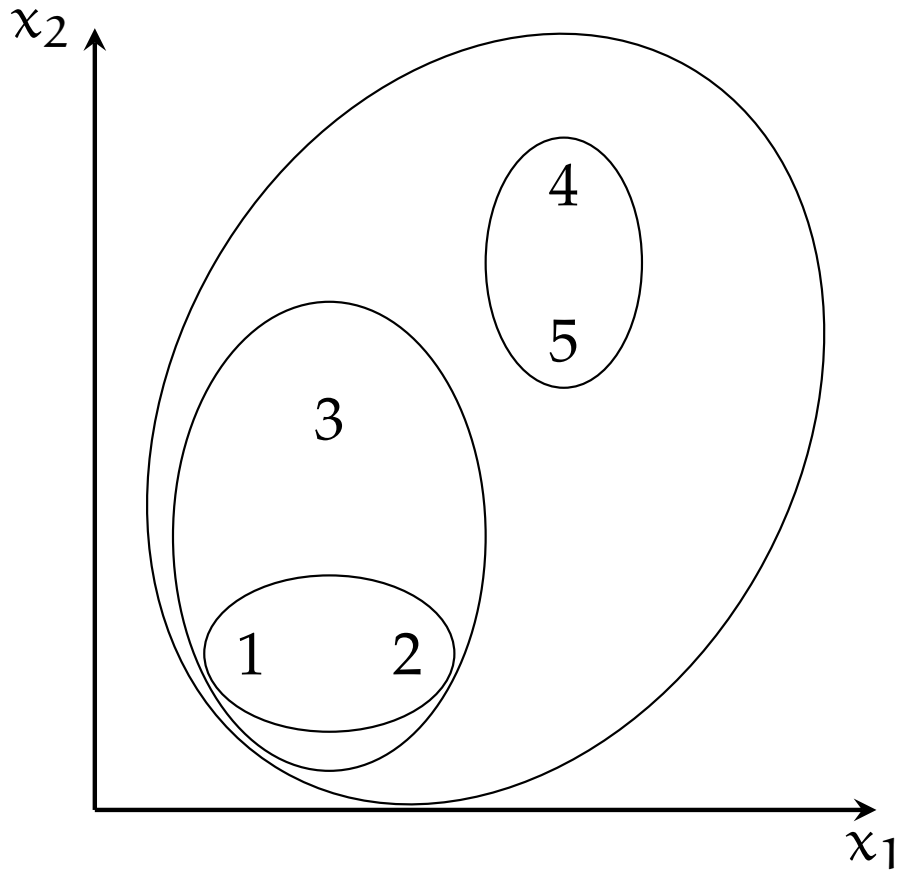
Agglomerative clustering demonstration





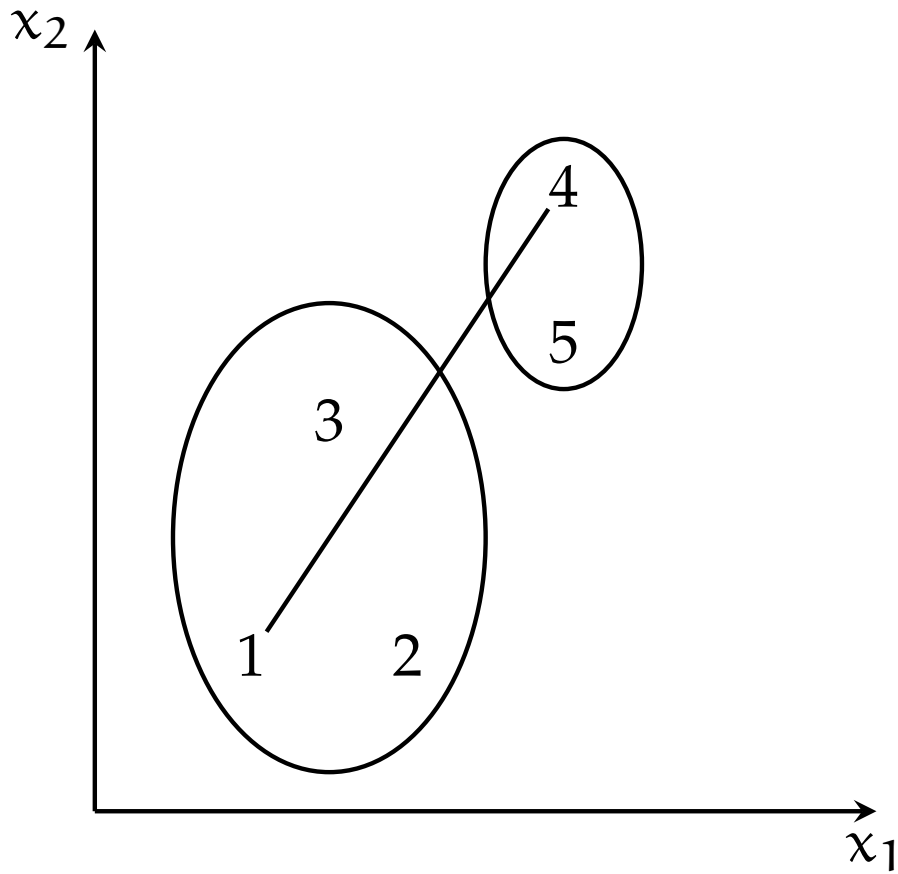
Agglomerative clustering demonstration





How to calculate between cluster distances

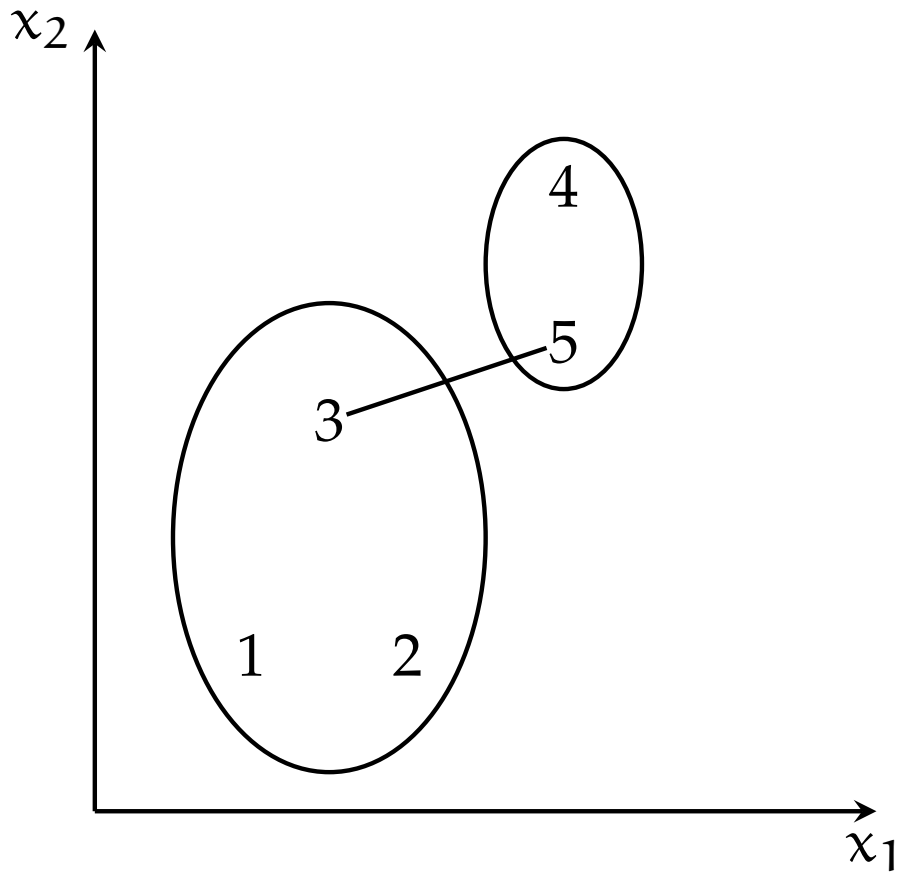
Complete maximal
inter-cluster distance



How to calculate between cluster distances

Complete maximal inter-cluster distance

Single minimal inter-cluster distance

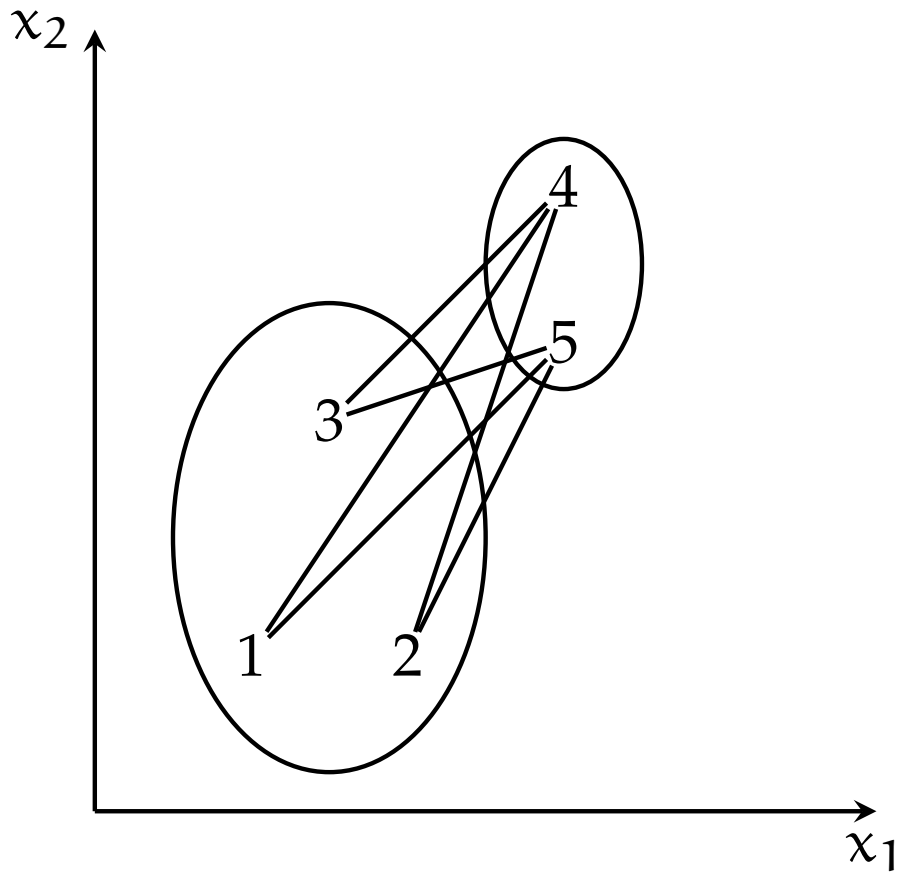


How to calculate between cluster distances

Complete maximal inter-cluster distance

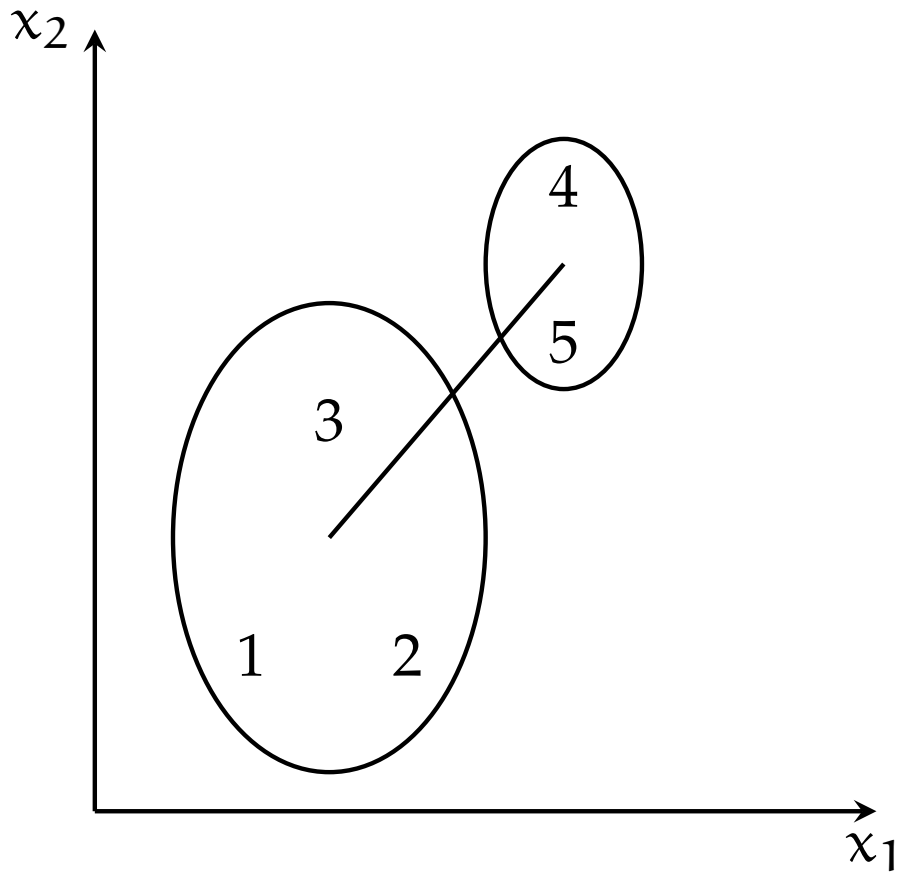
Single minimal inter-cluster distance

Average mean inter-cluster distance



How to calculate between cluster distances

Complete	maximal inter-cluster distance
Single	minimal inter-cluster distance
Average	mean inter-cluster distance
Centroid	distance between the centroids



Note: we only need distances, (feature) vectors are not necessary

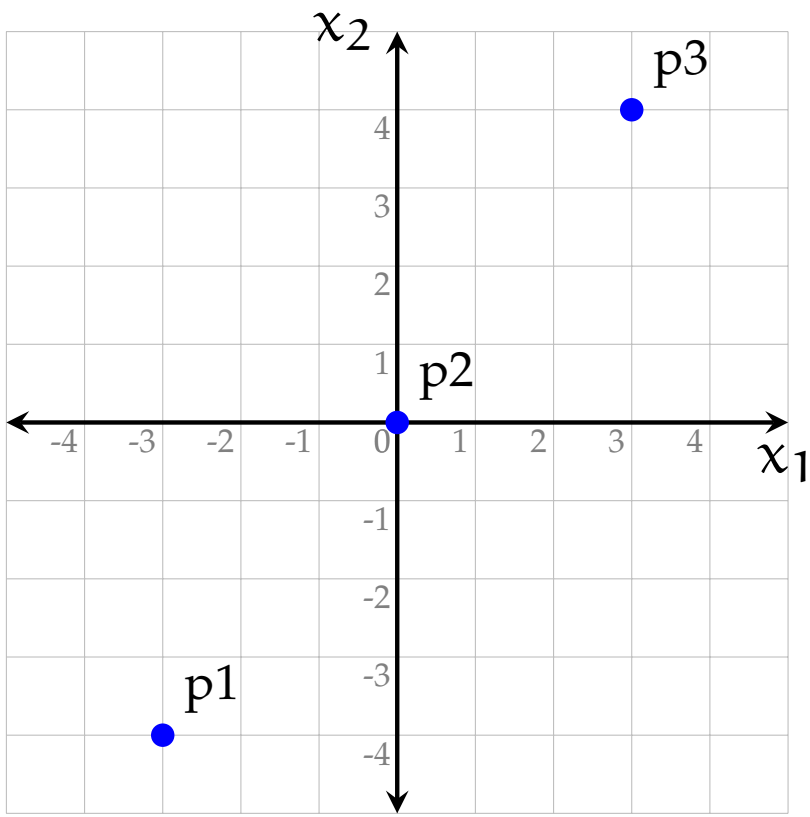
Clustering: some closing notes

- We do not have proper evaluation procedures for clustering results (for unsupervised learning in general)
- Clustering is typically unstable, slight changes in the data or parameter choices may change the results drastically
- Approaches against instability include some validation methods, or producing 'probabilistic' dendrograms by running clustering with different options

Principal component Analysis

- Principal component analysis (PCA) is a method of *dimensionality reduction*
- PCA maps the original data into a lower dimensional space by a linear transformation (rotation)
- The transformed variables retain most of the variation (=information) in the input
- PCA can be used for
  - visualization
  - data compression
  - reducing dimensionality for use in supervised methods
  - eliminating noise

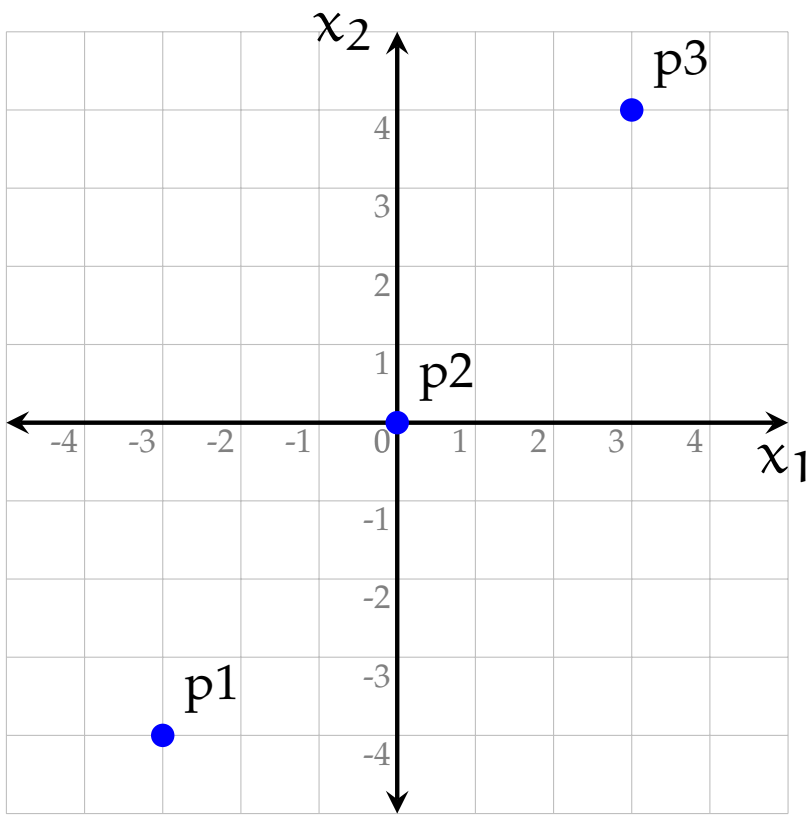
PCA: a toy example



Questions:

- How many dimensions do we have?
- How many dimensions do we need?

PCA: a toy example

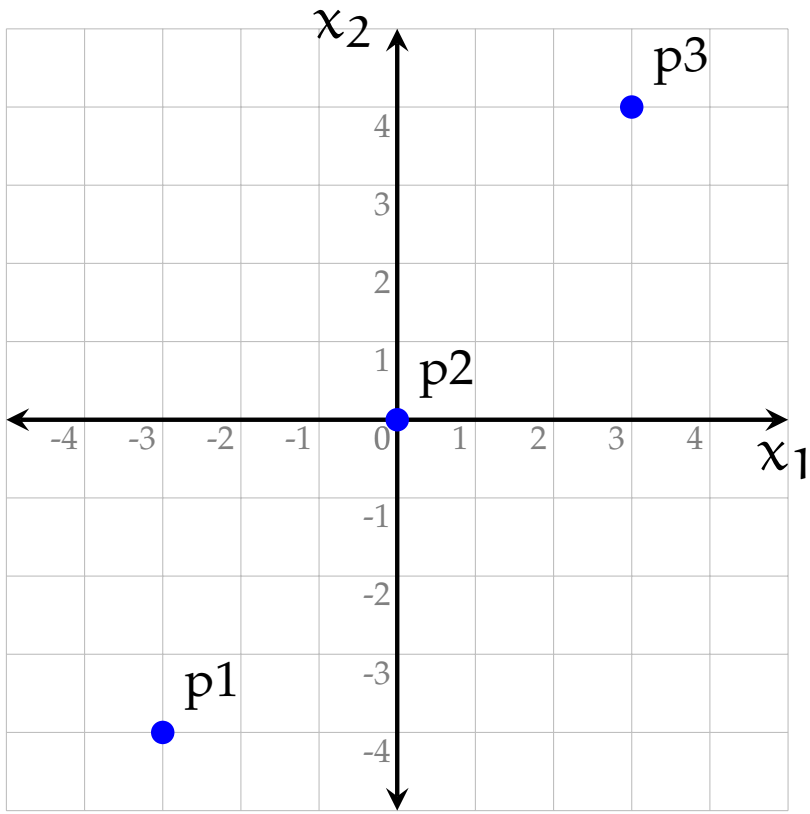


Questions:

- How many dimensions do we have?
- How many dimensions do we need?
- Short divergence: calculate the covariance matrix

$$\Sigma = \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix}$$

PCA: a toy example

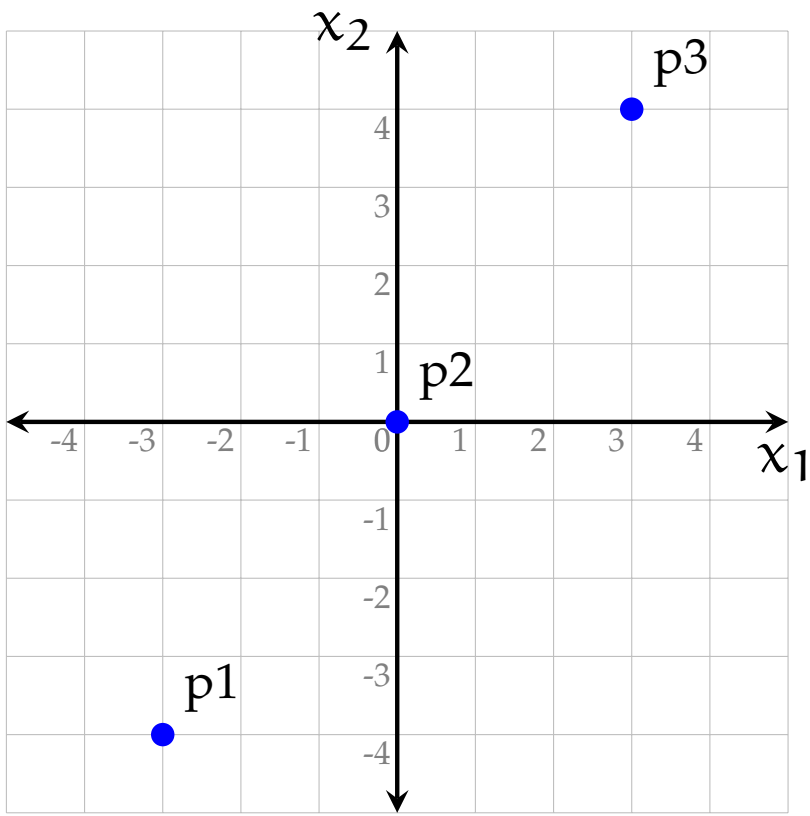


Questions:

- How many dimensions do we have?
- How many dimensions do we need?
- Short divergence: calculate the covariance matrix

$$\Sigma = \begin{bmatrix} \sigma^2_{x_1} & \sigma_{x_2,x_1} \\ \sigma_{x_1,x_2} & \sigma^2_{x_2} \end{bmatrix}$$

PCA: a toy example

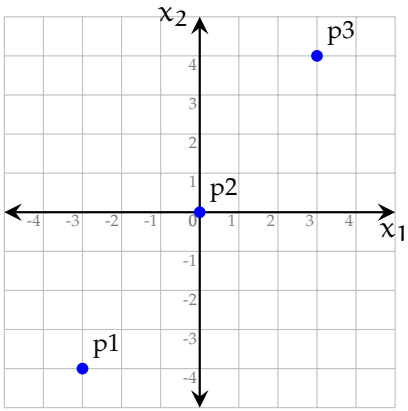


Questions:

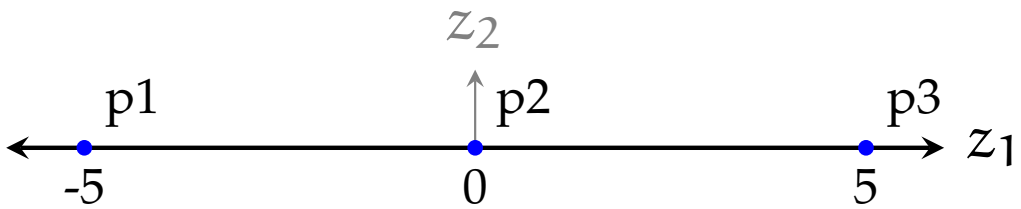
- How many dimensions do we have?
- How many dimensions do we need?
- Short divergence: calculate the covariance matrix

$$\Sigma = \begin{bmatrix} \frac{18}{3} & 8 \\ 8 & \frac{32}{3} \end{bmatrix}$$

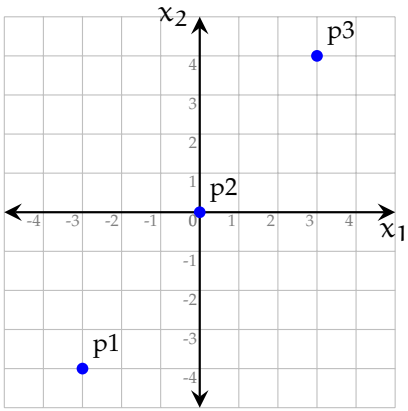
PCA: A toy example (2)



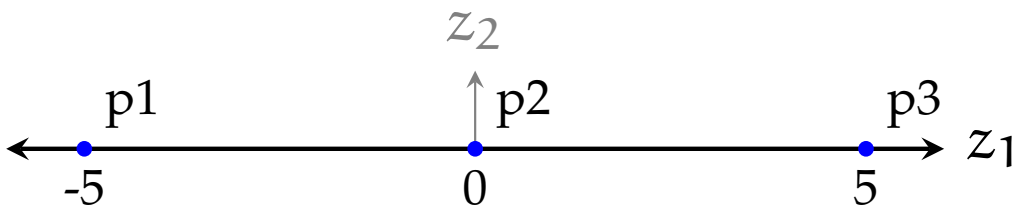
What if we reduce the data to:



PCA: A toy example (2)



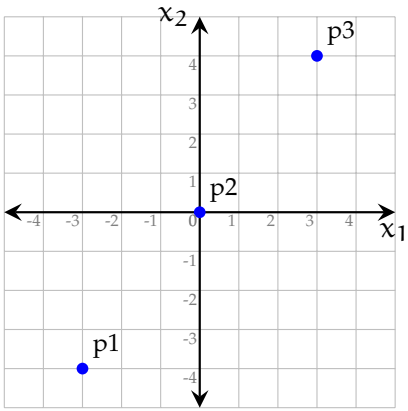
What if we reduce the data to:



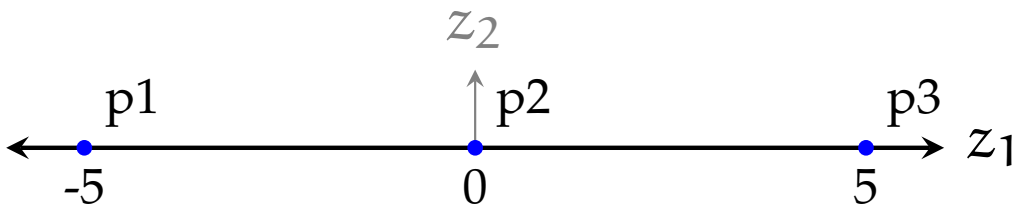
Going back to the original coordinates is easy, rotate using:

$$A = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} = \begin{bmatrix} \frac{3}{5} & -\frac{4}{5} \\ \frac{4}{5} & \frac{3}{5} \end{bmatrix}$$

PCA: A toy example (2)



What if we reduce the data to:

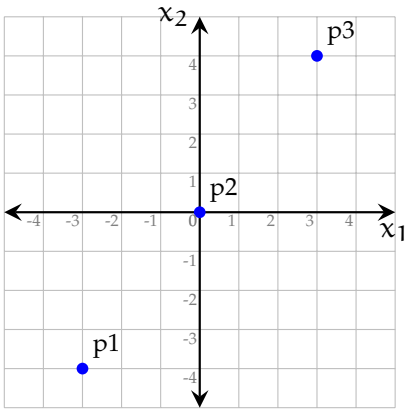


Going back to the original coordinates is easy, rotate using:

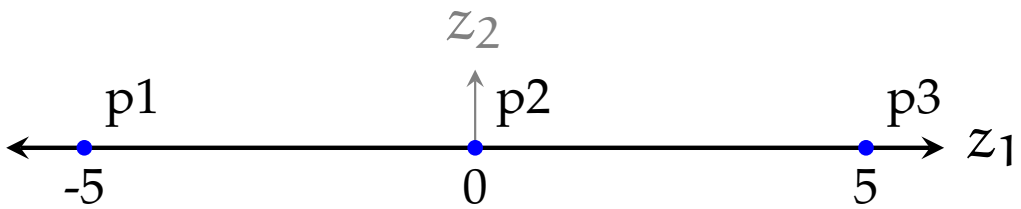
$$A = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} = \begin{bmatrix} \frac{3}{5} & -\frac{4}{5} \\ \frac{4}{5} & \frac{3}{5} \end{bmatrix}$$

$$p1 = A \times \begin{bmatrix} -5 \\ 0 \end{bmatrix} = \begin{bmatrix} -3 \\ -4 \end{bmatrix} \quad p1 = A \times \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad p1 = A \times \begin{bmatrix} 5 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

PCA: A toy example (2)



What if we reduce the data to:



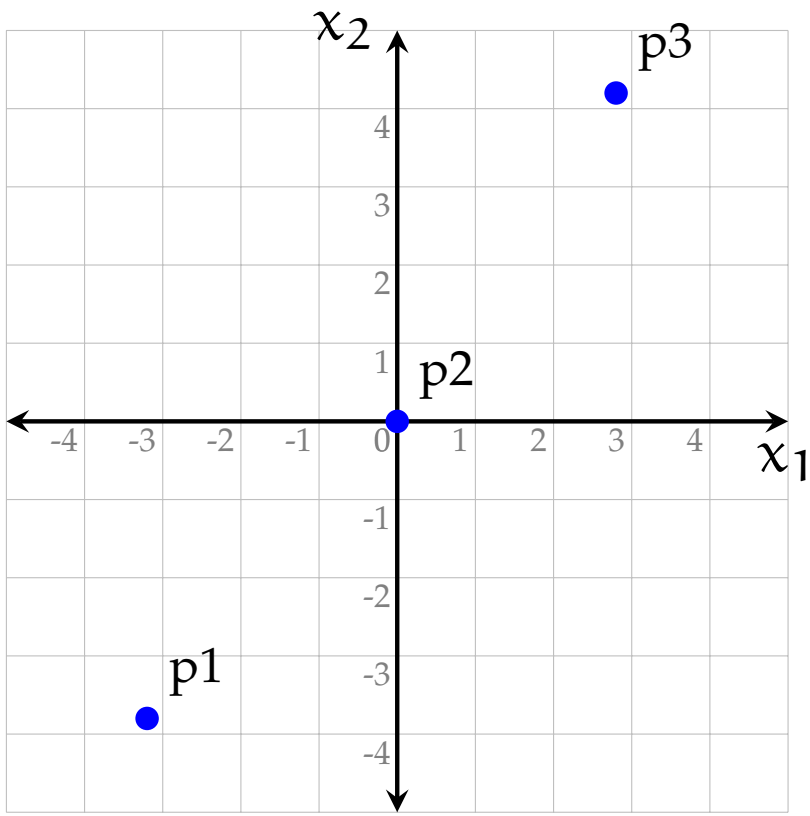
Going back to the original coordinates is easy, rotate using:

$$A = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} = \begin{bmatrix} \frac{3}{5} & -\frac{4}{5} \\ \frac{4}{5} & \frac{3}{5} \end{bmatrix}$$

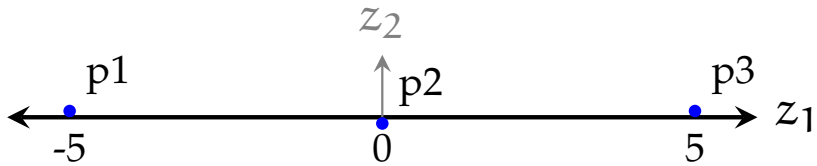
$$p1 = A \times \begin{bmatrix} -5 \\ 0 \end{bmatrix} = \begin{bmatrix} -3 \\ -4 \end{bmatrix} \quad p1 = A \times \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad p1 = A \times \begin{bmatrix} 5 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

We can recover the original points perfectly. In this example the inherent dimensionality of the data is only 1.

PCA: A toy example (3)



- What if the variables were not perfectly but strongly correlated?
- We could still do a similar transformation:



- Discarding z_2 results in a small reconstruction error:

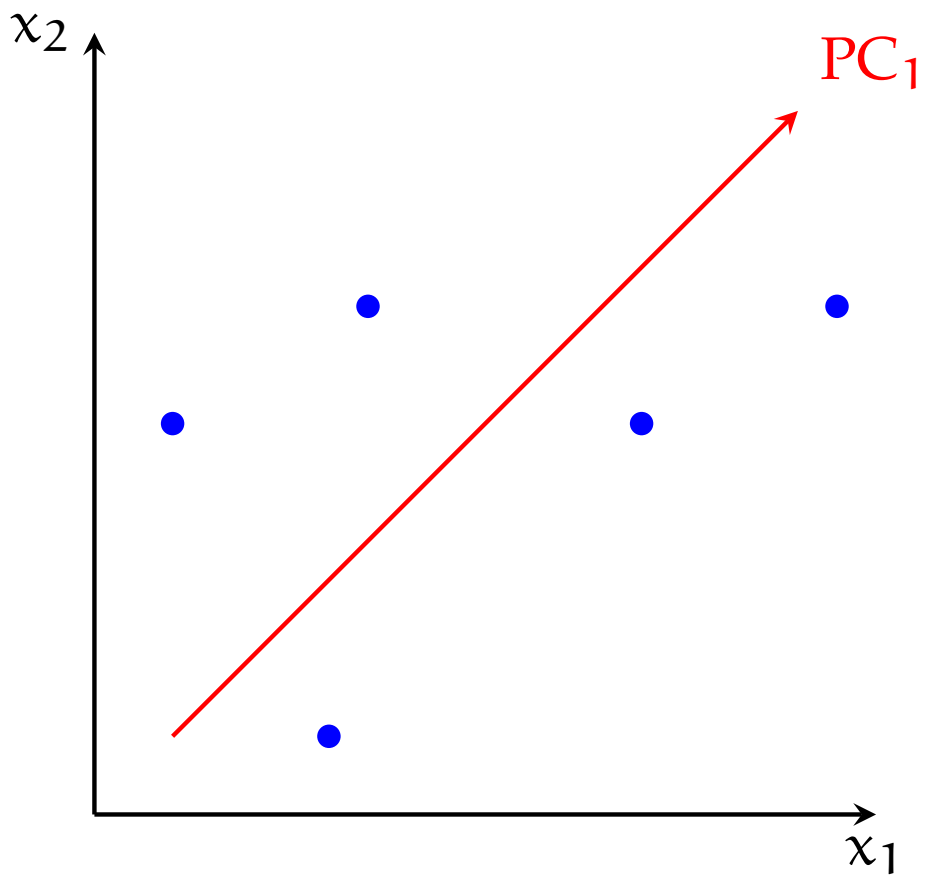
$$\text{p1} = A \times \begin{bmatrix} -5 \\ 0 \end{bmatrix} = \begin{bmatrix} -3 \\ -4 \end{bmatrix}$$

- Note: z_1 (also z_2) is a linear combination of original variables

Why do we want to reduce the dimensionality

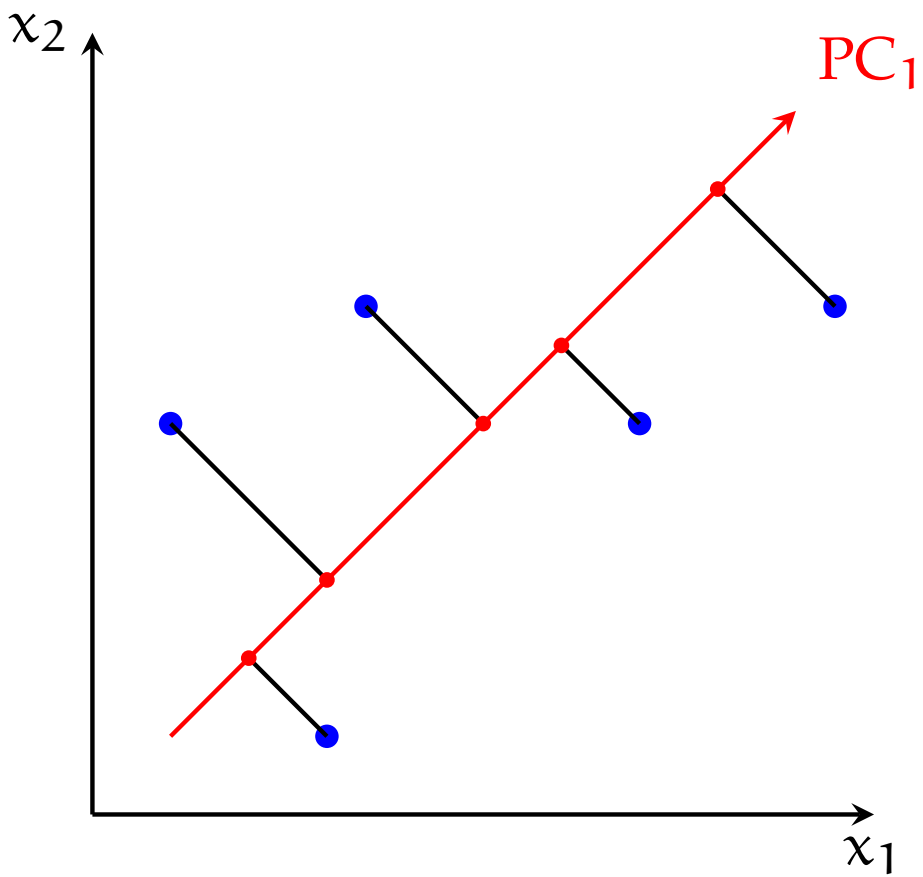
- Visualizing high-dimensional data becomes possible
- If we use the data for supervised learning, we avoid 'the curse of dimensionality'
- Decorrelation is useful in some applications
- We compress the data (in a lossy way)
- We eliminate noise (assuming a high signal to noise ratio)

Different views on PCA



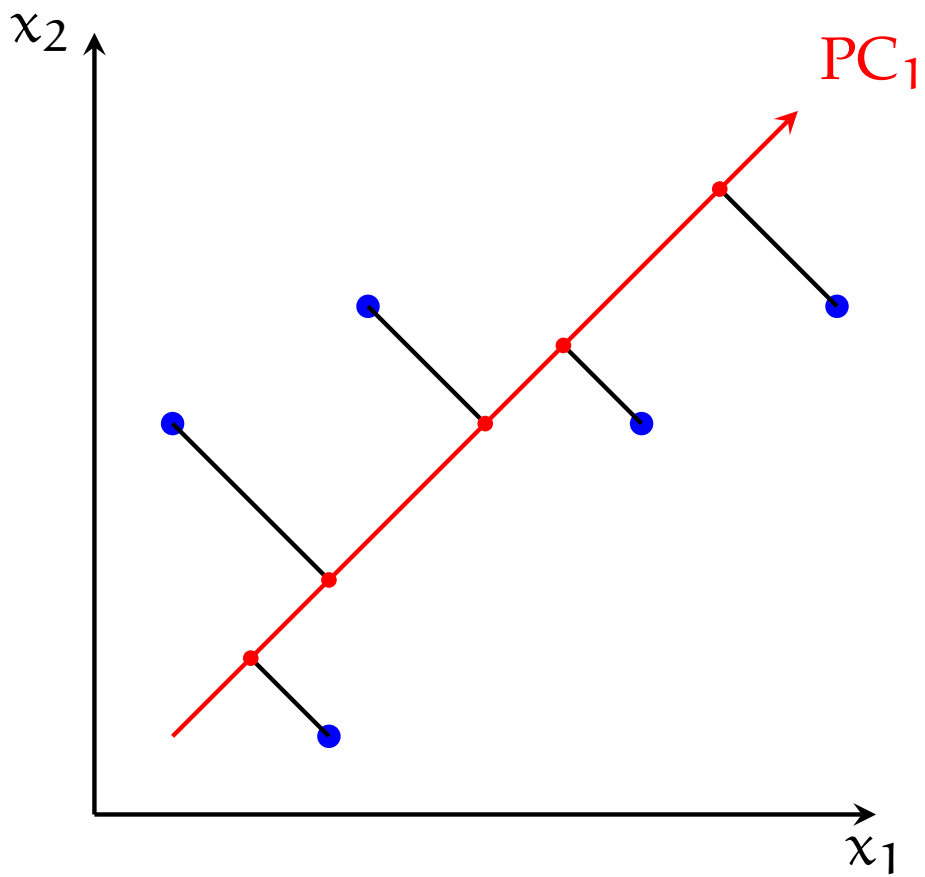
- Find the direction of the largest variance

Different views on PCA



- Find the direction of the largest variance
- Find the projection with the least reconstruction error

Different views on PCA



- Find the direction of the largest variance
- Find the projection with the least reconstruction error
- Find a lower dimensional latent Gaussian variable such that the observed variable is a mapping of the latent variable to a higher dimensional space (with added noise)

How to find PCs

- When viewed as *maximizing variance* or *reducing the re construction error*, we can write the appropriate objective function and find the vectors that minimize it
- In latent variable interpretation, we can use EM as in estimating mixtures of Gaussians
- The principle components are the eigenvectors of the correlation matrix, where large eigenvalues correspond to components with large variation
- A numerically stable way to obtain principal components is doing *singular value decomposition* (SVD) on the input data

PCA as matrix factorization (eigenvalue decomposition)

- One can compute PCA by decomposing the covariance matrix as (note $\Sigma = X^T X$)

$$\Sigma = U \Lambda U^T$$

  - the columns of U are the principal components (eigenvectors)
  - Λ is a diagonal matrix of eigenvalues
- Another option is SVD, which factorizes the input vector (k variables $\times$ n data points) as

$$X = U D V^*$$

  - U ($k \times k$) contains the eigenvectors as before,
  - D ($k \times k$) diagonal matrix $D^2 = \Lambda$
  - V^* is a $k \times n$ unitary matrix

* The above is correct for standardized variables, otherwise the formulas get slightly more complicated.

A practical example
(with simplified/fake data)

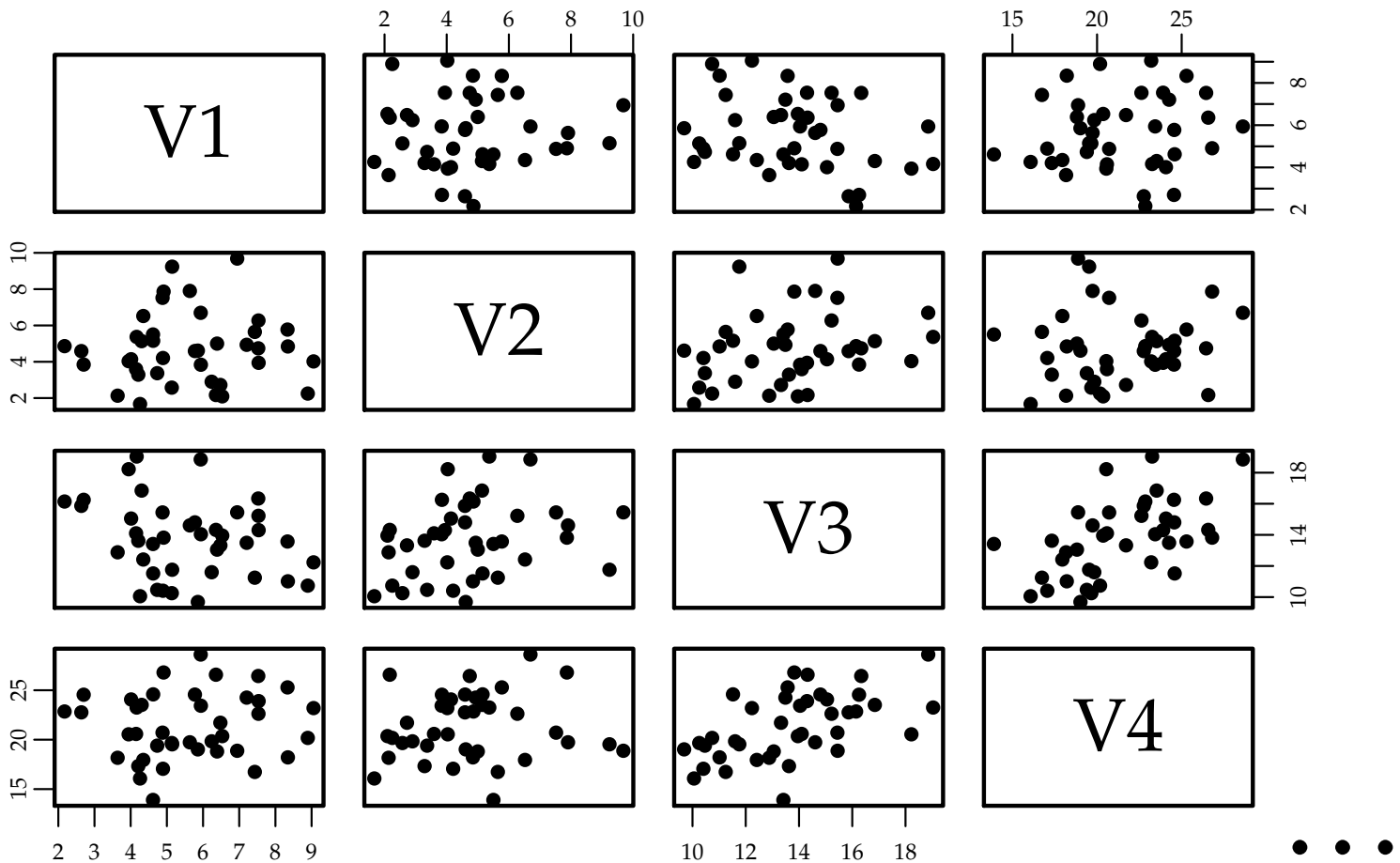
- Our data consists of 'measurements' from speech signal of instances of two vowels, we have 12 measurements for each vowel instance

$$\begin{bmatrix} 5.19 & 4.33 & 14.76 & 30.08 & 14.73 & 7.06 & 15.56 & 24.46 & 8.51 & \dots \\ 2.99 & 5.25 & 11.69 & 19.27 & 18.02 & 11.04 & 13.34 & 38.13 & 8.70 & \dots \\ 6.25 & 6.05 & 13.88 & 19.26 & 17.81 & 6.95 & 12.58 & 39.74 & 9.58 & \dots \\ 7.24 & 5.43 & 15.15 & 18.93 & 15.69 & 10.18 & 14.89 & 34.86 & 10.03 & \dots \\ 6.07 & 6.27 & 13.34 & 17.60 & 19.98 & 11.04 & 13.28 & 36.02 & 8.66 & \dots \\ & & & & \dots & & & & & \end{bmatrix}$$

- How do we visualize this data?
- Are all 12 variables useful?

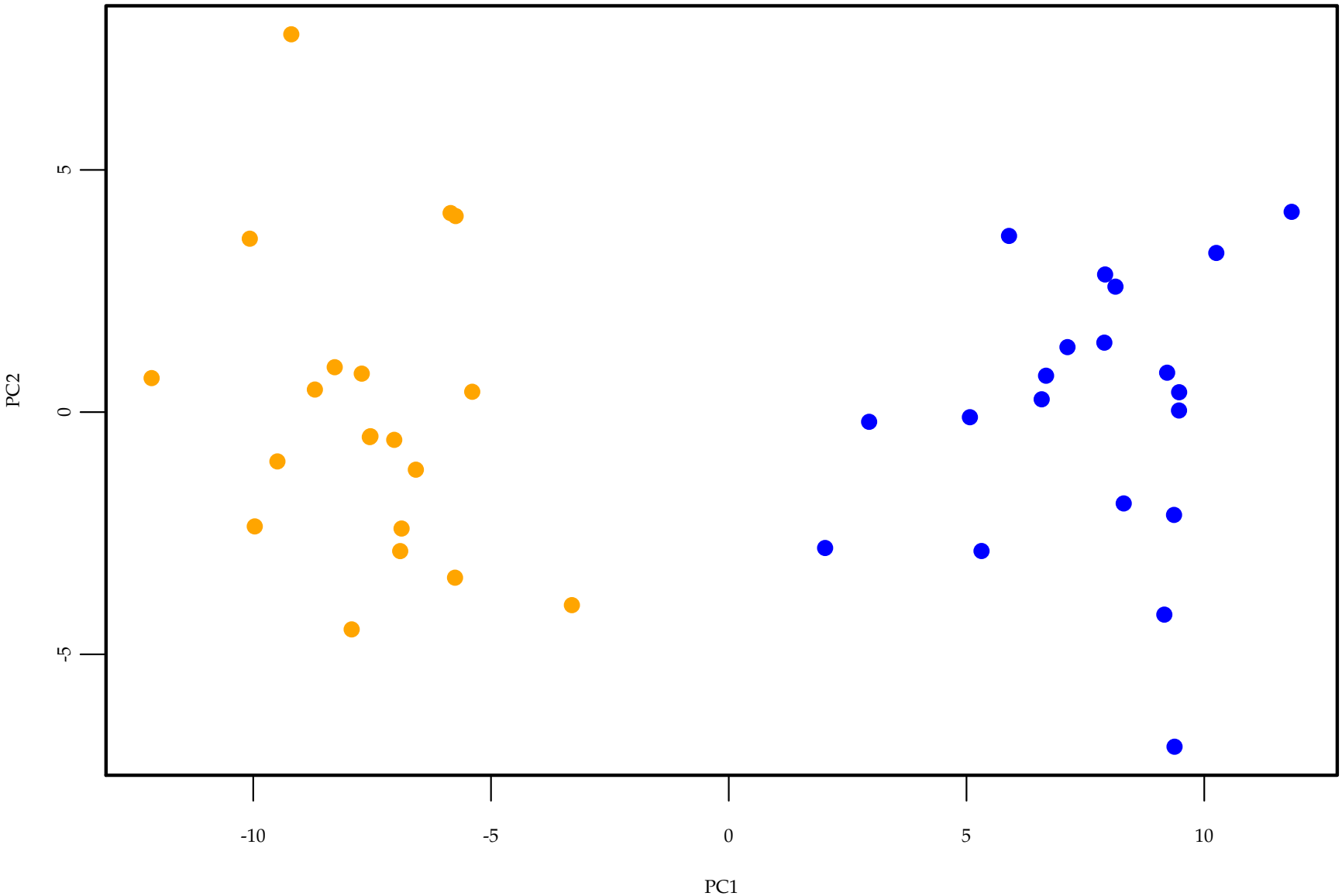
A practical example

Visualizing with pairwise scatter plots



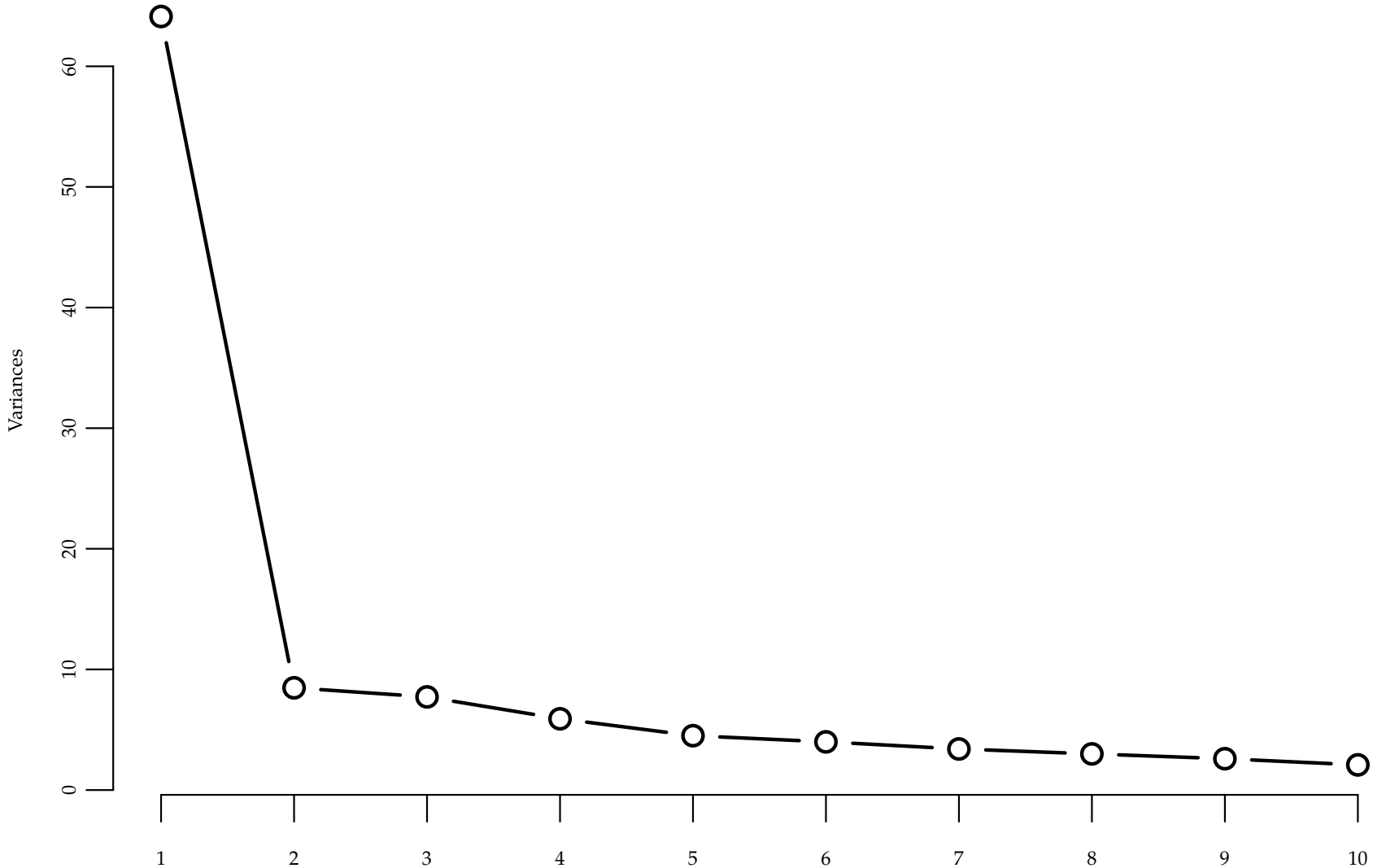
A practical example

Plotting the first two principal components



A practical example

How many components to keep? (scree plot)



Some practical notes on PCA

- Variables need to be centered
- Scales of the variables matter, standardizing may be a good idea depending on the units/scales of the individual variables
- The sign of the principal component (vector) is not important
- If there are more variables than the data points, we can still calculate the principal components, but there will be at most $n - 1$ PCs
- PCA will be successful if variables are linearly correlated, there are extensions for dealing with nonlinearities (e.g., kernel PCA, ICA)

Unsupervised learning: a summary (so far)

- In unsupervised learning, we do not have labels. Our aim is to find/exploit (latent) structure in the data
- We studied a number of related methods

  Clustering finds groups in the data

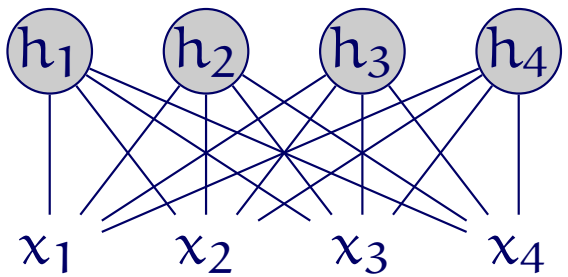
  Mixture densities are a 'soft' version of the clustering, assuming data is generated by a number of distributions

  Dimensionality reduction methods try to summarize the data with fewer variables/dimensions
- The evaluation of unsupervised methods are problematic, without knowing what we should exactly find in the data

Unsupervised learning in ANNs

- *Restricted Boltzmann machines* (RBM)
  similar to the latent variable models (e.g., Gaussian mixtures), consider the representation learned by hidden layers as hidden variables ($\mathbf{h}$), and learn $p(\mathbf{x}, \mathbf{h})$ that maximize the probability of the (unlabeled)data
- *Autoencoders*
  train a constrained feed-forward network to predict its output

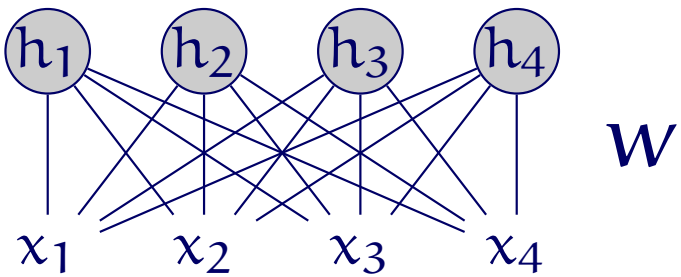
Restricted Boltzmann machines (RBMs)



- RBMs are unsupervised latent variable models, they learn only from unlabeled data
- They are generative models of the joint probability $p(\mathbf{h}, \mathbf{x})$
- They correspond to undirected graphical models
- No links within layers
- The aim is to learn useful features ($\mathbf{h}$)

*Biases are omitted in the diagrams and the formulas for simplicity.

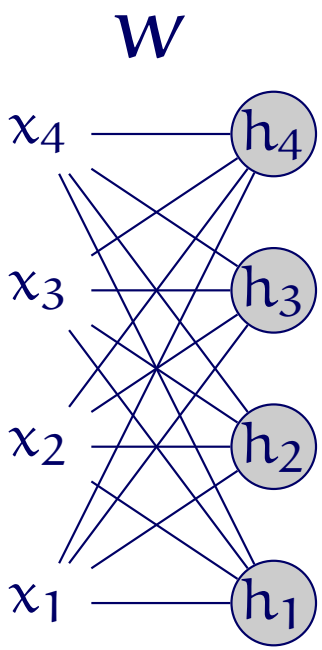
Restricted Boltzmann machines (RBMs)



- RBMs are unsupervised latent variable models, they learn only from unlabeled data
- They are generative models of the joint probability $p(\mathbf{h}, \mathbf{x})$
- They correspond to undirected graphical models
- No links within layers
- The aim is to learn useful features ($\mathbf{h}$)

*Biases are omitted in the diagrams and the formulas for simplicity.

The distribution defined by RBMs



$$p(\mathbf{h}, \mathbf{x}) = \frac{e^{h^T \mathbf{W} x}}{Z}$$

This calculation is intractable (Z is difficult to calculate).
But conditional distributions are easy to calculate

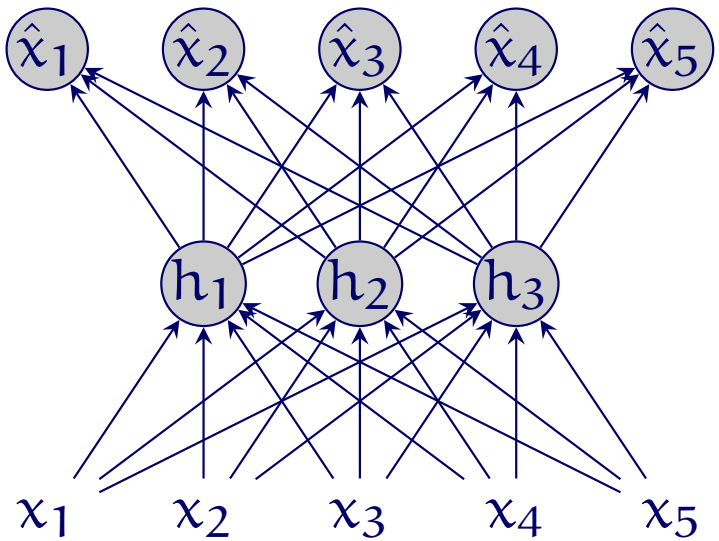
$$p(\mathbf{h}|\mathbf{x}) = \prod_j p(h_j|\mathbf{x}) \quad = \frac{1}{1 + e^{\mathbf{W}_j \mathbf{x}}}$$

$$p(\mathbf{x}|\mathbf{h}) = \prod_k p(x_k|\mathbf{h}) \quad = \frac{1}{1 + e^{\mathbf{W}_k^T \mathbf{h}}}$$

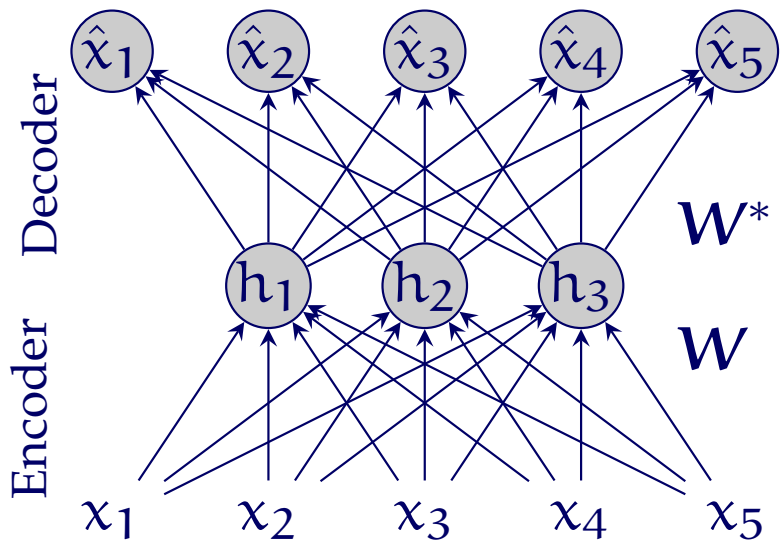
Learning in RBMs

- We want to maximize the probability the model assigns to the input, $p(x)$, or equivalently minimize $-\log p(x)$
- In general, this is computationally expensive
- *Contrastive divergence algorithm* is a well known algorithm that efficiently finds an approximate solution

Autoencoders

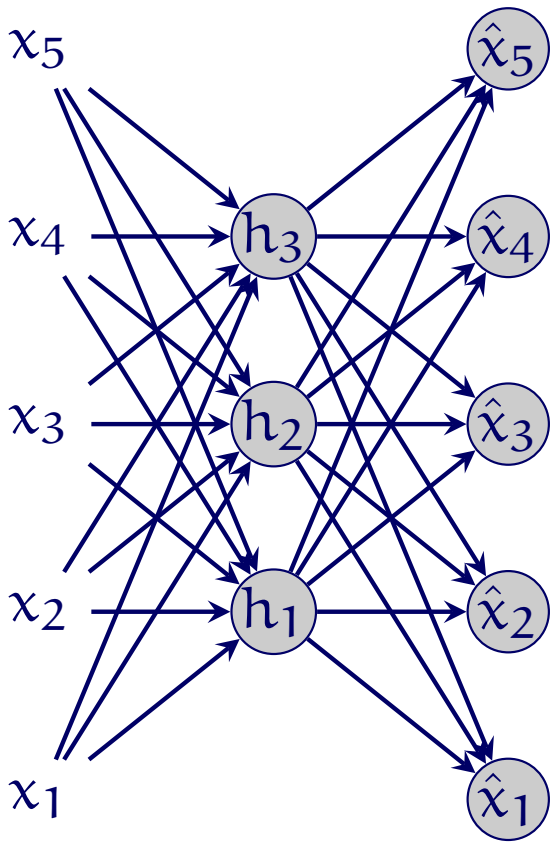


Autoencoders



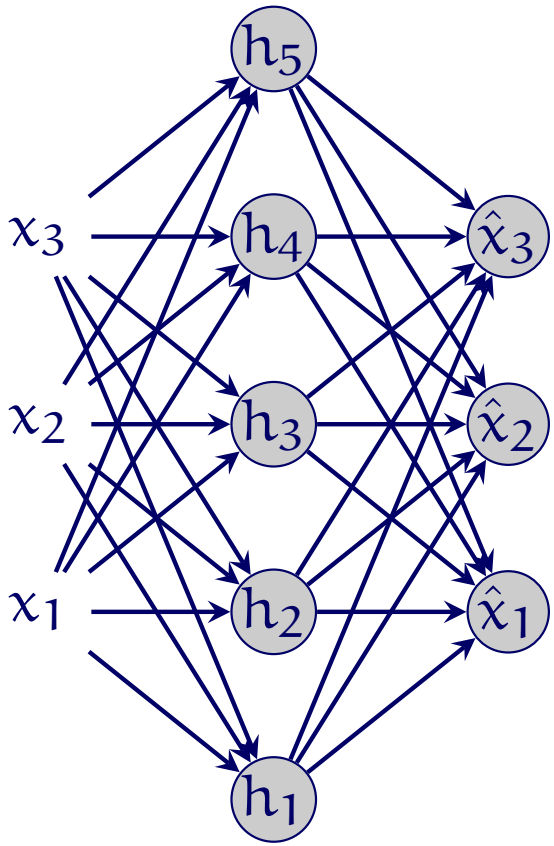
- Autoencoders are standard feed-forward networks
- The main difference is that they are trained to predict their input (they try to learn the identity function)
- The aim is to learn useful representations of input at the hidden layer
- Typically weights are tied ($W^* = W^\intercal$)

Under-complete autoencoders



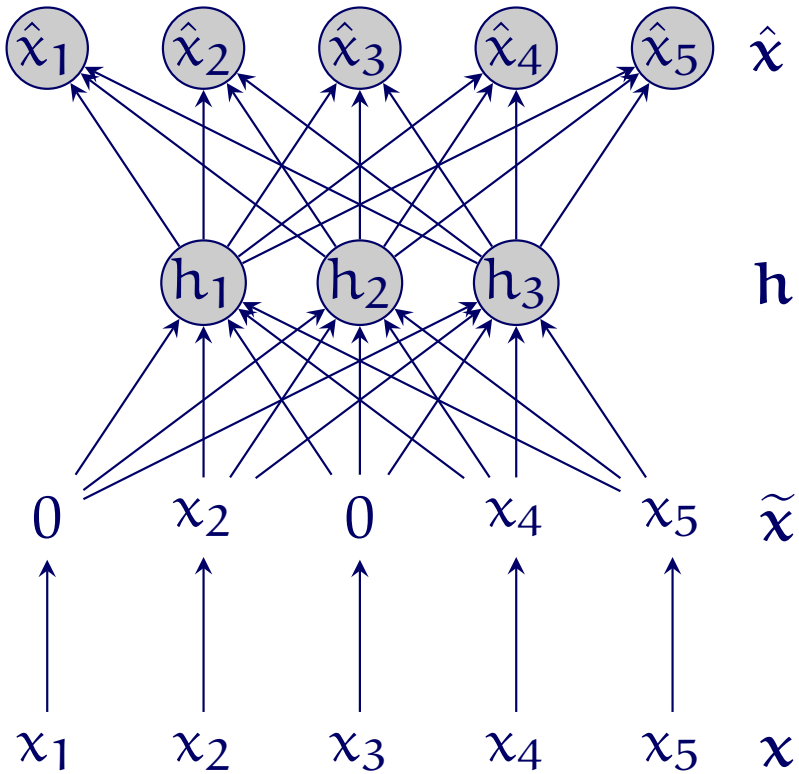
- An autoencoder is said to be *under-complete* if there are fewer hidden units than inputs
- The network is forced to learn a compact representation of the input (compress)
- An autoencoder with a single hidden layer is equivalent to PCA
- We need multiple layers for learning non-linear features

Over-complete autoencoders



- An autoencoder is said to be *over-complete* if there are more hidden units than inputs
- The network can normally memorize the input perfectly
- This type of networks are useful if trained with a regularization term resulting in sparse hidden units (e.g., L1 regularization)

Denoising autoencoders



- Instead of providing the exact input, we introduce noise by
  - randomly setting some inputs to 0 (dropout)
  - adding random (Gaussian) noise
- Network is still expected to reconstruct the original input (without noise)

Unsupervised pre-training

- A common use case for RBMs and autoencoders are as pre-training methods for supervised networks
- Autoencoders or RBMs are trained using unlabeled data
- The weights learned during the unsupervised learning is used for initializing the weights of a supervised network
- This approach has been one of the reasons for success of deep networks

Deep unsupervised learning

- Both autoencoders and RBMs can be 'stacked'
- Learn the weights of the first hidden layer from the data
- Freeze the weights, and using the hidden layer activations as input, train another hidden layer, …
- This approach is called *greedy layer-wise training*
- In case of RBMs resulting networks are called *deep belief networks*
- Deep autoencoders are called *stacked autoencoders*

Summary

- Unsupervised methods try to discover 'hidden' structure in the data
- Clustering is used for finding groups of clusters in the data without labels
- Dimensionality reduction transforms the data in a low dimensional space while keeping most of the information in the original data
- RBM and autoencoders learn (typically lover dimensional, dense, continuous) representations of the input that are useful in other tasks

Summary

- Unsupervised methods try to discover 'hidden' structure in the data
- Clustering is used for finding groups of clusters in the data without labels
- Dimensionality reduction transforms the data in a low dimensional space while keeping most of the information in the original data
- RBM and autoencoders learn (typically lover dimensional, dense, continuous) representations of the input that are useful in other tasks

Next:

Today(?) Distributed representations

Wed(?) Text classification

Exam

- On Wednesday July 26
- It should take about an hour
- Mix of true/false questions and long-answer questions
- Your main source is the course slides, the recommended reading will help
- Understanding the graded/ungraded exercises is important
- Focus will be on NLP methods/applications
- Questions measuring your understanding of the methods/topics, no emphasis on memorizing
- You can bring an A4 cheat sheet, both sides are OK, should be readable to the unaided eye

Last two graded assignments

- Assignment 2 is posted today
- Two deadlines

Jul 31 you get 1 point bonus, detailed feedback, an initial attempt may be helpful for the exam

Oct 2 Full grade, but no feedback

- Assignment 3 will be posted next week
  - Topic will be sentiment analysis
  - Similar two-deadline schedule

Derivation of PCA by maximizing the variance

- We focus on the first PC (z_1), which maximizes the variance of the data onto itself
- We are interested only on the direction, so we choose z_1 to be a unit vector ($\|z_1\| = 1$)
- Remember that to project a vector onto another, we simply use dot product, So the projected data points are zx_i for $i = 1, \ldots, N$.
- The variance of the projected data points (that we want to maximize) is,

$$\sigma_{z_1} = \frac{1}{N} \sum_{i}^{N} (z_1 x_i - z_1 \bar{x}_i)^2 = z_1^T \Sigma z$$

where Σ_x is the covariance matrix of the unprojected data

Derivation of PCA by maximizing the variance (cont.)

- The problem becomes maximize

$$z_1^\top \Sigma z$$

  with the constraint $\|z_1\| = z_1^\top z_1 = 1$
- Turning it into a unconstrained optimization problem with Lagrange multipliers, we minimize

$$z_1^\top \Sigma z + \lambda_1 (1 - z_1^\top z_1)$$

- Taking the derivative and setting it to 0 gives us

$$\Sigma z_1 = \lambda_1 z_1$$

  Note: by definition, z_1 is an eigenvector of Σ, and λ_1 is the corresponding eigenvalue
- z_1 is the first principal component, we can now compute the second principal component with the constraint that it has to be orthogonal to the first one