Unsupervised Learning

Statistical Methods in NLP 2

ISCL-BA-08

Çağrı Çöltekin
ccoltekin@sfs.uni-tuebingen.de

University of Tübingen
Seminar für Sprachwissenschaft

Summer Semester 2025

version: 5b6dd30 @2025-06-19

Unsupervised learning

- In unsupervised learning, we 'train' our models without knowing the outcome variable
- The aim is to find useful patterns/structure in the data
- Two main categories:
  - *Clustering* groups the data into discrete categories
  - *Dimensionality reduction* reduces dimensionality of data while preserving the structure as much as possible
- Evaluation is difficult: no '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

Hierarchical clustering

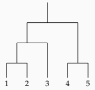
- Hierarchical clustering builds a tree based on similarity of the data points
- There are two main 'modes of operation':

Bottom-up or *agglomerative* clustering
- starts with individual data points,
- merges the clusters until all data is in a single cluster

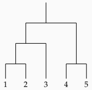
Top-down or *divisive* clustering
- starts with a single cluster,
- and splits until all leaves are single data points

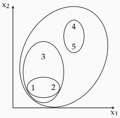
Agglomerative clustering

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





K-means algorithm

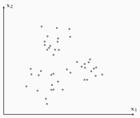
- K-means is a popular method for clustering.
- The idea is finding 'k' centroids for each cluster
- Each data point belongs to the cluster with the closest centroid

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

- The algorithm minimizes distances of data poitns from their cluster centroid
- This is *local minimum* of the *within-cluster scatter* with Euclidean distances

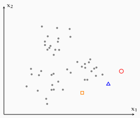
$$\frac{1}{2}\sum_{k=1}^{K}\sum_{a\in C_k}\sum_{b\in C_k}\|a-b\|^2$$

K-means clustering: visualization



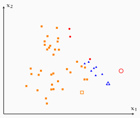
- The data
- Set cluster centroids randomly
- Assign data points to the closest centroid
- Recalculate the centroids

K-means clustering: visualization



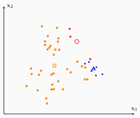
- The data
- Set cluster centroids randomly
- Assign data points to the closest centroid
- Recalculate the centroids

K-means clustering: visualization



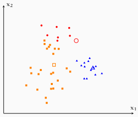
- The data
- Set cluster centroids randomly
- Assign data points to the closest centroid
- Recalculate the centroids

K-means clustering: visualization



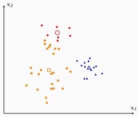
- The data
- Set cluster centroids randomly
- Assign data points to the closest centroid
- Recalculate the centroids

K-means clustering: visualization



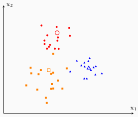
- The data
- Set cluster centroids randomly
- Assign data points to the closest centroid
- Recalculate the centroids

K-means clustering: visualization



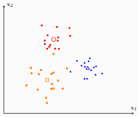
- The data
- Set cluster centroids randomly
- Assign data points to the closest centroid
- Recalculate the centroids

K-means clustering: visualization



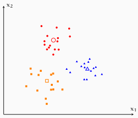
- The data
- Set cluster centroids randomly
- Assign data points to the closest centroid
- Recalculate the centroids

K-means clustering: visualization



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

K-means clustering: visualization



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

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

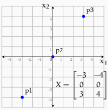
Dimensionality reduction

- The idea is to reduce the dimensionality of data while keeping the most important information
- Dimensionality reduction is useful in many practical applications
  - Factor Analysis: finding underlying ‘factors’ in observed data, such as surveys, exams
  - Multidimensional Scaling (MDS): transforming high-dimensional data to lower dimensions while preserving some relations between objects
  - t-SNE: a non-linear dimensionality reduction technique that places similar objects in closer proximity in low-dimensional space
- We will study two important techniques: *principal component analysis* (PCA) and *autoencoders*

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 lower-dimensional variables retain most of the variation (=information) in the input
- PCA can be used for
  - visualization
  - data compression
  - reducing dimensionality of features for other machine learning methods
  - eliminating noise

PCA: a toy example



$$X = \begin{bmatrix} -3 & -4 \\ 0 & 0 \\ 3 & 4 \end{bmatrix}$$

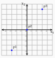
Questions:

- How many dimensions do we have?
- How many dimensions do we need?
- What is the rank of the data matrix X?
- Short divergence: calculate the covariance matrix

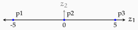
$$\Sigma = \begin{bmatrix} \frac{18}{3} & 8 \\ 8 & \frac{32}{3} \end{bmatrix}$$

  - What is the correlation between x_1 and x_2?

PCA: A toy example (2)



What if we reduce the data to:



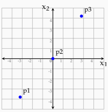
Going back to the original coordinates is easy, rotate using:

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

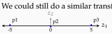
$$p1 = A \times \begin{bmatrix} -5 \\ 0 \end{bmatrix} = \begin{bmatrix} -3 \\ -4 \end{bmatrix} \quad p2 = A \times \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad p3 = A \times \begin{bmatrix} 5 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

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:

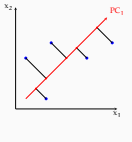
$$p1 = A \times \begin{bmatrix} -5 \\ 0 \end{bmatrix} = \begin{bmatrix} -3 \\ -4 \end{bmatrix}$$

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

Why do we want to reduce the dimensionality

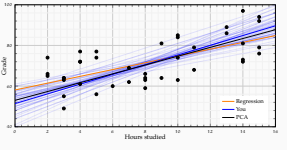
- Visualizing high-dimensional data becomes possible
- If we use the data for other ML methods,
  - we reduce the computation time
  - we may avoid ‘the curse of dimensionality’
- Decorrelation is useful in some applications
- We compress the data (in a lossy way)
- We eliminate noise (assuming a high signal to noise ratio)

Different views on PCA



- Find the direction of the largest variance
- 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)

A short divergence: your regression estimates and PCA



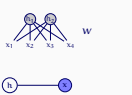
Some practical notes on PCA

- Scales of the variables matter, standardizing may be a good idea depending on the units/scales of the individual variables
- The sign/direction of the principal component (vector) is not important
- The most common way to perform PCA is through SVD
- 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 correlated, there are extensions for dealing with nonlinearities (e.g., kernel PCA, ICA, t-SNE)

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 ($\mathbf{h}$)

The distribution defined by RBMs

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

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

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

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

Learning in RBMs

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

Autoencoders

- Simple 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
- The weights are often shared/tied ($W' = W^\top$)

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 approximates the 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 unsupervised (or self-supervised) models is 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
- Alternatively, the representations learned for inputs can be used as input to other methods
- This approach has been one of the reasons for success of deep networks

More complex unsupervised methods

- Any network architecture that produce a representation (encoding) of the input can be used as autoencoders
- It is common to use more complex autoencoders in many applications
  - CNNs over images (e.g., for digit/letter recognition)
  - RNNs or Transformers over sequences (e.g., for language modeling)
- Another unsupervised framework is *generative adversarial networks* (GANs)
  - Use a generative network (e.g., an RNN or transposed convolution) to generate the (fake) objects of interests
  - Use a discriminator network (a classifier) that discriminates between fake and real objects
  - Train both end-to-end, optimizing generative network weights using discriminator's success as loss

Summary

- In unsupervised learning, we do not have labels. Our aim is to find/exploit (latent) structure in the data
- Unsupervised methods try to discover 'hidden' structure in the data

  Clustering finds groups in the data

  Density estimation estimates parameters of latent probability distributions

  Dimensionality reduction transforms the data in a low dimensional space while keeping most of the information in the original data

Next:

- N-gram language models. Reading: Jurafsky and Martin (2025, Chapter 3)

Additional reading, references, credits

Jurafsky, Daniel and James H. Martin (2025). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. 3rd. Online manuscript released January 12, 2025. URL: https://web.stanford.edu/~jurafsky/slp3/.