Machine Learning

Winter 2011/12

Roland Memisevic

Lecture 5, Nov. 21, 2011

Outline

- Unsupervised learning and latent variables.
- K-means clustering.
- Gaussian mixture models.
- The EM-algorithm.

Unsupervised learning

- So far we looked almost exclusively at *supervised learning tasks*

$$\mathbf{x} \rightarrow \mathbf{t}$$

  where the goal is to learn dependencies from data and thus to build systems "by example".
- This is the simplest and most obvious approach to learning, and it has obvious and well-known applications (like spam filtering, face recognition, stock price prediction, etc.).
- There is another, less obvious, but somewhat deeper, type of learning, often referred to as unsupervised learning:

Unsupervised learning

- Given *just* data

$$\mathbf{x}$$

  learn to "understand" the data, by re-representing it in some intelligent way.
- Example: **Clustering** – Find natural grouping in data.
- Example: **Dimensionality reduction** – Find projections that carry important information.
- Example: **Compression** – Represent data using fewer bits.
- Unsupervised learning is like supervised learning with missing outputs (or with missing inputs).

Latent variables

- Most unsupervised learning methods can be formalized elegantly using the concept of latent or hidden variables:
- We assume the data generating process has internal parameters which we cannot directly observe, but which affect the data nevertheless.
- Formally, we postulate that there is a hidden variable $\mathbf{z}_n$ associated with each training case $\mathbf{x}_n$.
- The goal of learning is to infer the values of the $\mathbf{z}_n$ given just the data.

Latent variables

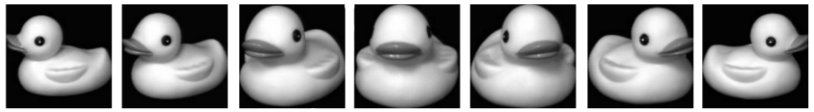
- Example:

(images from "Indoor-Outdoor Image Classification", M. Szummer, R. Picard)

- A discrete latent variable may determine where some picture was taken.

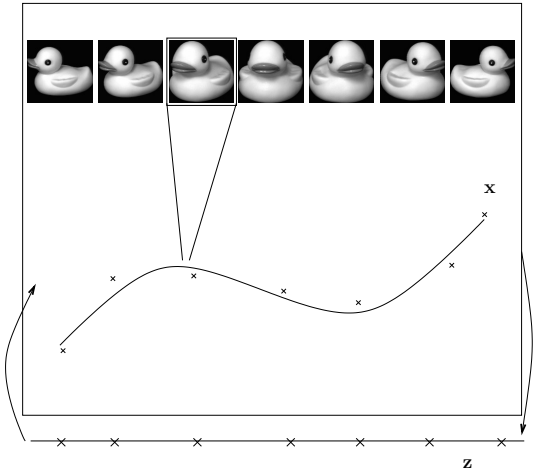
Latent variables

- Another example:



- A continuous latent variable may determine the angle under which an object is depicted.

Latent variables

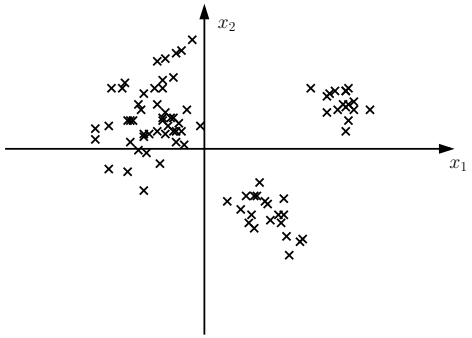


- *Learning* about these hidden causes of variability in the data can help compress, understand or pre-process the data.

K-means clustering

- We consider only discrete latent variables in this lecture. We shall deal with continuous latent variables later.
- We can model a discrete latent variable using a one-of-K-encoding $\mathbf{z}_n$.
- Inferring this discrete latent variable from data is known as clustering, since it amounts to grouping each point into one of K groups.
- We can think of $\mathbf{z}_n$ as *assigning* each point $\mathbf{x}_n$ to one cluster.
- One of the most common clustering methods is K-means clustering.

K-means clustering



- Find the groups!

K-means clustering

- Notation: We shall stack one-hot encodings $\mathbf{z}_n$ of discrete latent variables row-wise in a matrix $\mathbf{Z}$.
- We further assume that there is a set of K *prototypes* $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$ that represent the K clusters. The dimensionality of the prototypes is the same as that of the data $\mathbf{x}$.
- Assume for a moment, that we *knew* the cluster assignments $\mathbf{z}_n$ for each point $\mathbf{x}_n$.
- The objective function that K-means clustering tries to minimize is the *average distance between points* $\mathbf{x}$ *and their cluster-representatives*:

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2$$

K-means clustering

- *Learning* amounts to finding *both* the prototypes $\boldsymbol{\mu}_k$ *and* the assignments $\mathbf{z}_n$ for each point, so as to minimize J.
- This is a tricky optimization problem, because the $\mathbf{z}_n$ are discrete (and the $\boldsymbol{\mu}_k$ are continuous).
- Learning can be greatly simplified if we decouple learning the $\mathbf{z}_n$ from learning the $\boldsymbol{\mu}_k$.
- This gives rise to a *block coordinate-descent method*, which is a special case of a general optimization approach in unsupervised learning, known as the **EM-algorithm.**

Finding the optimal $\mathbf{z}_n$

- Note that given the $\boldsymbol{\mu}_k$, we can optimize all the $\mathbf{z}_n$ independently, because the objective is just the sum over n.
- But the squared error will be smallest if we set $z_{nk} = 1$ for whichever $\boldsymbol{\mu}_k$ is *closest*.
- Formally, to optimize all $\mathbf{z}_n$, given the set of $\boldsymbol{\mu}_k$, set:

$$z_{nk} = \begin{cases} 1 & \text{if } k = \arg\min_j \|\mathbf{x}_n - \boldsymbol{\mu}_j\|^2 \\ 0 & \text{otherwise.} \end{cases}$$

Finding the optimal $\boldsymbol{\mu}_k$

- Given the $\mathbf{z}_n$, the objective function J is a quadratic function of $\boldsymbol{\mu}_k$ which we can optimize by setting the derivative to zero:

$$2\sum_{n=1}^{N} z_{nk}(\mathbf{x}_n - \boldsymbol{\mu}_k) = 0$$

- Solving for $\boldsymbol{\mu}_k$ yields:

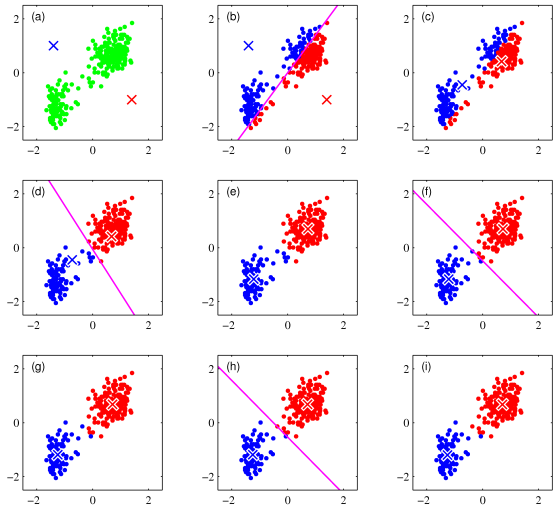
$$\boldsymbol{\mu}_k = \frac{\sum_n z_{nk}\mathbf{x}_n}{\sum_n z_{nk}}$$

- This solution has a simple interpretation: Set each $\boldsymbol{\mu}_k$ to the mean of all points currently assigned to cluster k !

Iterating inference and parameter adaptation

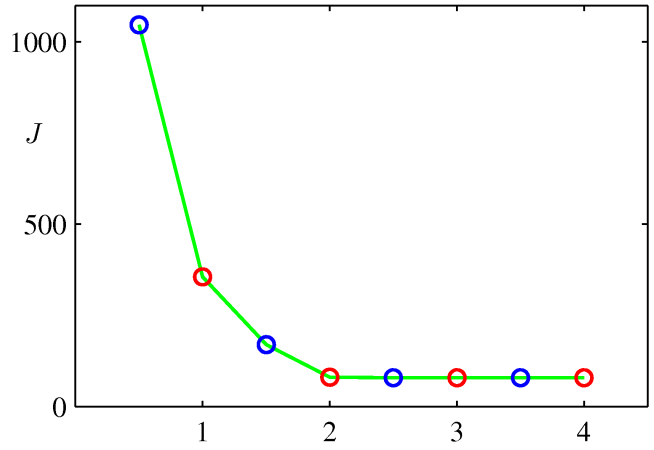
- Learning amounts to iterating inference of the $\mathbf{z}_n$, and adapting the parameters $\boldsymbol{\mu}_k$ until there are no more changes.
- This training procedure always converges: J is positive, and every step either decreases it or leaves it unchanged.
- But note that there can be local minima.
- One way to deal with them is to try multiple runs with different initializations for the parameters $\boldsymbol{\mu_k}$ and to pick the solution with the lowest final cost.

K-means example



- Learning a model with 2 cluster centers.

The value of J as learning progresses



Inference for test-data

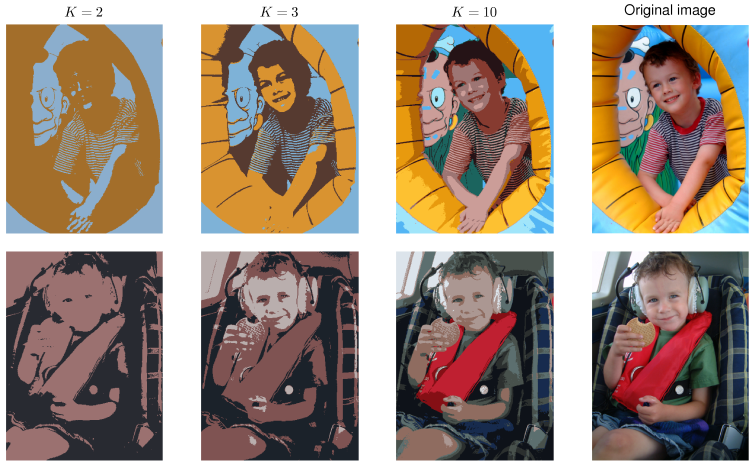
- Given a trained model, we can infer the cluster-centers for new test-data points $\mathbf{x}$ not seen during training: Pick the nearest $\boldsymbol{\mu}_k$ like we did during training:

$$z_k = \begin{cases} 1 & \text{if } k = \arg\min_j \|\mathbf{x}_n - \boldsymbol{\mu}_j\|^2 \\ 0 & \text{otherwise.} \end{cases}$$

- Since $\mathbf{z}$ represents the high-dimensional vector $\mathbf{x}$ using only one of K integers, K-means is a way to perform *lossy compression*.
- The set of all K prototypes $\boldsymbol{\mu}_k$ is sometimes called *codebook*.
- Clustering and K-means are also known as *vector quantization*.

A simple application

- Replace the RGB-value (a 3-D vector) at each pixel with one of K prototypes:



Gaussian mixture models

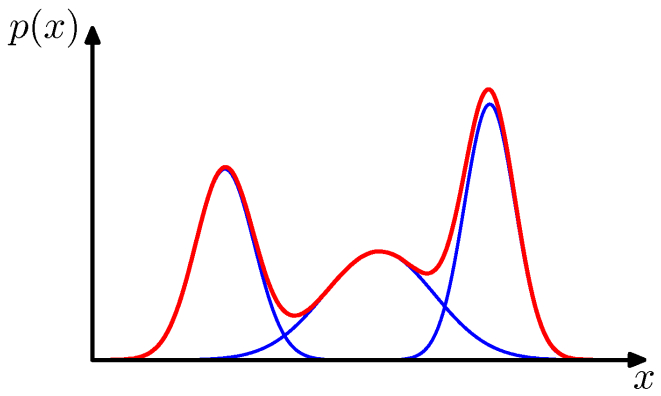
- K-means is closely related to a probabilistic model known as the

Mixture of Gaussians

$$p(\mathbf{x}) = \sum_k \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

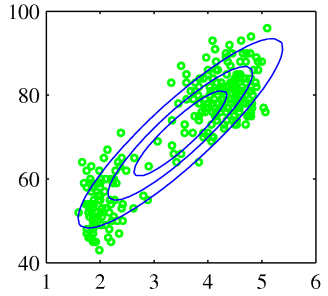
- $\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$ are parameters. π_k are called *mixing proportions*, each Gaussian is called a *mixture component*.
- The model is simply a weighted sum of Gaussians. But it is much more powerful than a single Gaussian, because it can model *multi-modal* distributions:

Gaussian mixture models example

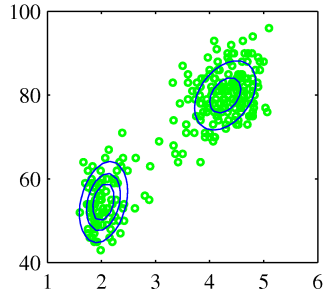


- A mixture of three Gaussians.

Gaussian mixture models example



A Gaussian fit to some data.



Gaussian mixture fit to same data.

Gaussian mixture models

$$p(\mathbf{x}) = \sum_k \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

- Note that for $p(\mathbf{x})$ to be a probability distribution, we require that $\sum_k \pi_k = 1$ and that $\pi_k > 0 \quad \forall k$
- Thus, we may interpret the π_k as probabilities themselves!
- This motivates introducing latent variables $\boldsymbol{z}$ and re-writing the model, equivalently, in terms of two distributions $p(\mathbf{z})$ and $p(\mathbf{z}|\mathbf{x})$ as follows:

$$p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{z}) p(\mathbf{x}|\mathbf{z})$$

Gaussian mixture models

- Here

$$p(\mathbf{z}) = \prod_{k=1}^{K} \pi_k^{z_k}$$

  is a discrete distribution (that is, $\mathbf{z}$ is a one-hot encoding like in K-means.)
- And

$$p(\mathbf{x}|z_k = 1) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

  is a conditional Gaussian distribution.
- Why rewrite the mixture model like this?

Gaussian mixture models

- We can now think of the model as a generative process, where we first draw a mixture component from a discrete distribution, and then we draw the observation from a Gaussian distribution, whose parameters depend on the component.
- This allows us to use Bayes' rule to compute posteriors

$$p(\mathbf{z}_n|\mathbf{x}_n) = \frac{p(\mathbf{x}_n|\mathbf{z}_n)p(\mathbf{z}_n)}{\sum_{\mathbf{z}_n} p(\mathbf{x}_n|\mathbf{z}_n)p(\mathbf{z}_n)}$$

  which represent how likely a given observation $\mathbf{x}_n$ is to come from a particular mixture component.
- $p(z_{nk} = 1|\mathbf{x}_n)$ is often referred to as the *responsibility* of mixture component k, and it is often abbreviated $\gamma(z_{kn})$.
- The Gaussian mixture is thus like a "soft" version of K-means.

The EM algorithm

- We now turn to parameter estimation.
- While we could use gradient-based optimization, there is a more convenient two-step procedure similar to K-means.
- Given training data $\{\mathbf{x}_n\}$, we can write

$$\begin{aligned} L := \sum_n \log p(\mathbf{x}_n) &= \sum_n \log \sum_{\mathbf{z}_n} p(\mathbf{x}_n|\mathbf{z}_n)p(\mathbf{z}_n) \\ &= \sum_n \log \sum_{\mathbf{z}_n} q(\mathbf{z}_n) \frac{p(\mathbf{x}_n|\mathbf{z}_n)p(\mathbf{z}_n)}{q(\mathbf{z}_n)} \\ &\geq \sum_n \sum_{\mathbf{z}_n} q(\mathbf{z}_n) \log \frac{p(\mathbf{x}_n|\mathbf{z}_n)p(\mathbf{z}_n)}{q(\mathbf{z}_n)} \\ &:= \mathcal{L} \end{aligned}$$

  where we use **Jensen's inequality**:
  $\log \sum_i a_i b_i \geq \sum_i a_i \log b_i \quad \text{if } \forall i : a_i > 0 \text{ and } \sum_i a_i = 1$

The EM algorithm

- Instead of optimizing L, we will now optimize the lower bound $\mathcal{L}$ with respect to *both* the original parameters and the newly introduced auxiliary variables $q(\mathbf{z})$.
- To avoid clutter, it is convenient to rewrite

$$\mathcal{L} = \sum_{nk} q_{nk} \log p(\mathbf{x}_n|\mathbf{z}_n = k)p(\mathbf{z}_n = k) - \sum_{nk} q_{nk} \log q_{nk}$$

  where we abbreviate $q_{nk} = q(\mathbf{z}_n = k)$
- Note that the first term of $\mathcal{L}$ is the expectation of $p(\mathbf{x}_n, \mathbf{z}_n)$ with respect to $q(\mathbf{z}_n)$. It is commonly referred to as "expected complete log-likelihood".
- This is the only term that depends on model parameters. As usual, we can set derivatives to zero to optimize it:

The EM algorithm

- We have

$$\frac{\partial \mathcal{L}}{\partial \mu_k} = \sum_n q_{nk} \boldsymbol{\Sigma}_k (\mathbf{x}_n - \boldsymbol{\mu}_k) = 0$$

$$\Leftrightarrow \boldsymbol{\mu}_k = \frac{\sum_n q_{nk} \mathbf{x}_n}{\sum_n q_{nk}}$$

- Likewise, we get

$$\boldsymbol{\Sigma}_k = \frac{\sum_n q_{nk} (\mathbf{x}_n - \boldsymbol{\mu}_k)(\mathbf{x}_n - \boldsymbol{\mu}_k)^{\mathrm{T}}}{\sum_n q_{nk}}$$

  and

$$\pi_k = p(\mathbf{z}_k) = \frac{\sum_n q_{nk}}{N}$$

The EM algorithm

- But what about the auxiliary variables q_{nk}?
- We rewrite $\mathcal{L}$ once more in a different way:

$$\begin{aligned}
\mathcal{L} &= \sum_n \sum_{\mathbf{z}_n} q(\mathbf{z}_n) \log \frac{p(\mathbf{x}_n|\mathbf{z}_n)p(\mathbf{z}_n)}{q(\mathbf{z}_n)} \\
&= \sum_n \sum_{\mathbf{z}_n} q(\mathbf{z}_n) \log \frac{p(\mathbf{z}_n|\mathbf{x}_n)p(\mathbf{x}_n)}{q(\mathbf{z}_n)} \\
&= \sum_n \sum_{\mathbf{z}_n} q(\mathbf{z}_n) \log \frac{p(\mathbf{z}_n|\mathbf{x}_n)}{q(\mathbf{z}_n)} + \sum_n \sum_{\mathbf{z}_n} q(\mathbf{z}_n) \log p(\mathbf{x}_n) \\
&= -\sum_n \mathrm{KL}\left(q(\mathbf{z}_n) \,||\, p(\mathbf{z}_n|\mathbf{x}_n) \right) + L
\end{aligned}$$

The EM algorithm

- The

Kullback-Leibler divergence (KL-divergence)

$$\mathrm{KL}\left(p_1(\mathbf{z}) \,||\, p_2(\mathbf{z}) \right) = \sum_{\mathbf{z}} p_1(\mathbf{z}) \log \frac{p_1(\mathbf{z})}{p_2(\mathbf{z})}$$

measures the similarity between two probability distributions p_1 and p_1.

- The KL-divergence is always non-negative, and it is zero only for identical distributions (!)
- This means, that $\mathcal{L}$ will be equal to L, if we set $q(\mathbf{z}_n) = p(\mathbf{z}_n|\mathbf{x}_n)$!

The EM algorithm

- In other words, setting $q(\mathbf{z}_n) = p(\mathbf{z}_n|\mathbf{x}_n)$ will make the bound $\mathcal{L}$ on L tight!
- $p(\mathbf{z}_n|\mathbf{x}_n) = (\gamma(z_{nk}))$ is easy to compute as we saw before.
- We already know how to optimize $\mathcal{L}$ with respect to the model parameters. So we can repeatedly compute (by inferring q_{nk}), and then optimize, a tight lower bound on L.
- To summarize, the EM algorithm iterates two steps to find a (local) optimum of the log-likelihood of a mixture model:

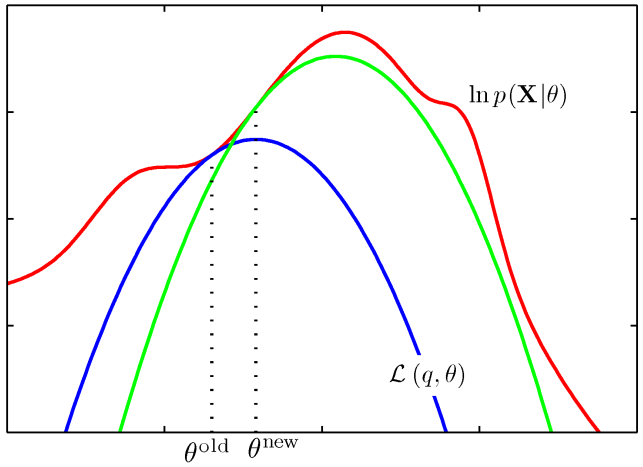
The EM algorithm

EM algorithm

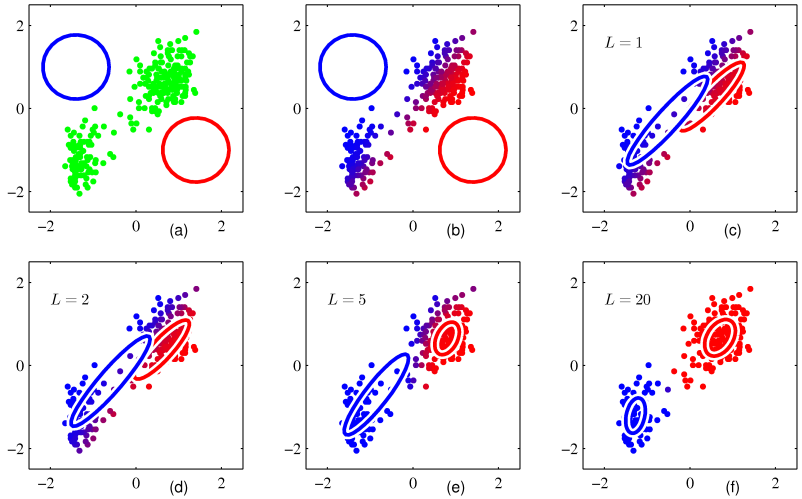
1. E-step: Evaluate the posteriors $p(\mathbf{z}_n|\mathbf{x}_n)$.
2. M-step: Optimize $\mathcal{L}$ with respect to the model parameters, keeping $q(\mathbf{z}_n) = p(\mathbf{z}_n|\mathbf{x}_n)$ fixed.

- The E-step computes the expected complete log-likelihood. It amounts to evaluating the responsibilities $\gamma(z_{nk})$ for each point.
- The M-step maximizes the expected complete log-likelihood. In a Gaussian mixture, this amounts to setting parameters to responsibility-weighted sums.

EM as optimizing a sequence of lower bounds



Example: Training a Gaussian mixture with EM



EM algorithm comments

- The EM algorithm can be applied to many latent variable models, not just mixtures of Gaussians.
- The E-step and the M-step have to be derived individually for each model, but the view from the lower bound $\mathcal{L}$ of the log-likelihood is always the same.
- One of the first models that deployed EM was the Hidden Markov Model.
- There are models where computing $p(\mathbf{z}|\mathbf{x})$ is not tractable. In this case, it is still common to deploy a variation of EM, where we only improve the KL-divergence in the E-step rather than finding the exact posterior.

Gaussian mixtures and K-means

- A Gaussian mixture model is like K-means where we use a "soft" assignment to clusters.
- One can formally derive K-means as the limit of a mixture model with infinitely small variances.