Probabilistic & Unsupervised Learning

Week 2: Latent Variable Models

Maneesh Sahani
maneesh@gatsby.ucl.ac.uk

Gatsby Computational Neuroscience Unit, and
MSc ML/CSML, Dept Computer Science
University College London

Term 1, Autumn 2011

Unsupervised Learning

The story so far . . .

- beliefs about data represented by parameterised distribution

$$P(\mathcal{D} \mid \theta, m) = \prod_{i=1}^{n} P(x_i \mid \theta, m)$$

- learning involves estimating a value (or distribution) for θ

Bayes $$P(\theta \mid \mathcal{D}, m) = \frac{P(\mathcal{D} \mid \theta, m) P(\theta \mid m)}{P(\mathcal{D} \mid m)}$$

MAP $$\theta^* = \operatorname*{argmax}_{\theta} P(\theta \mid \mathcal{D}, m)$$

ML $$\theta^* = \operatorname*{argmax}_{\theta} P(\mathcal{D} \mid \theta, m)$$

and (possibly) model selection

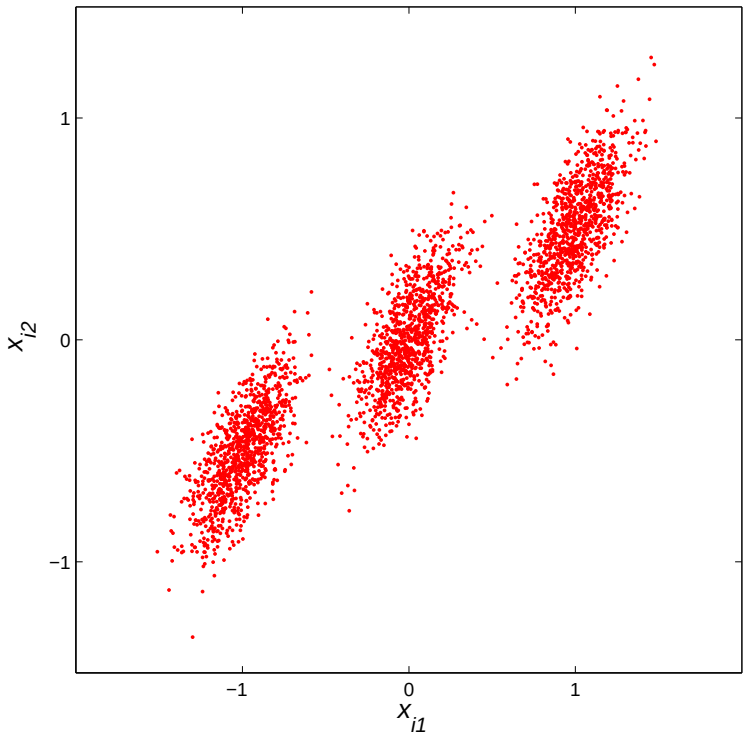
$$P(m|\mathcal{D}) \propto \int d\theta \; P(\mathcal{D}|\theta, m) P(\theta|m)$$

Aren't We Done?

No!

- direct application of Bayes' rule is intractable for all but the simplest models.
- even maximum-likelihood (or similar) learning may be prohibitive.
- algorithms (and approximations) are dictated by form of distribution.

What about these data?

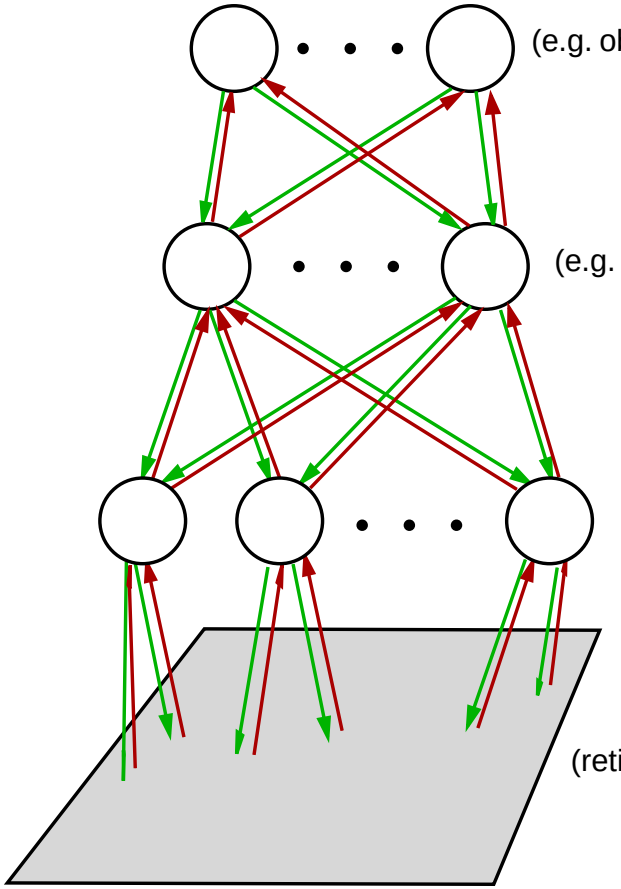


- joint distribution $p(x_{i1}, x_{i2})$ is not Normal.
- conditional distributions $p(x_{i1} \mid x_{i2})$ and $p(x_{i2} \mid x_{i1})$ vary, and are difficult to codify.
- easier to describe by an latent tri-valued discrete process

$$s_i \sim Discrete[1/3, 1/3, 1/3]$$
$$\mathbf{x}_i \sim \mathcal{N}(\mu_{s_i}, \Sigma)$$

Latent Variable Models

Explain correlations in **x** by assuming some latent variables **y**



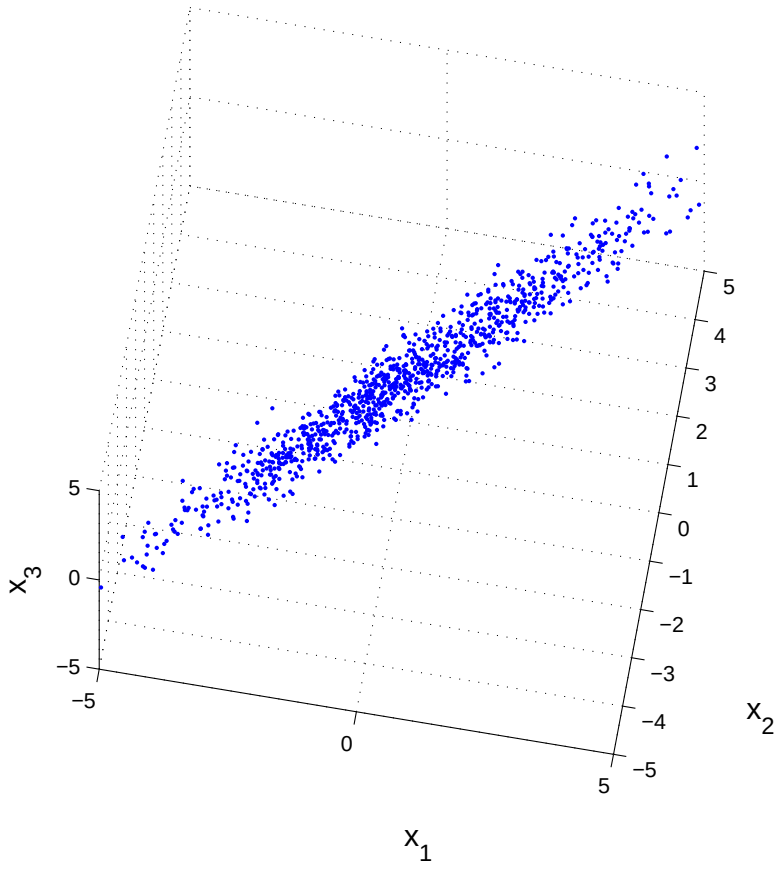
$$\mathbf{y} \sim \mathcal{P}[\theta_y]$$

$$\mathbf{x} \mid \mathbf{y} \sim \mathcal{P}[\theta_x]$$

$$p(\mathbf{x}, \mathbf{y}; \theta_x, \theta_y) = p(\mathbf{x} \mid \mathbf{y}; \theta_x) p(\mathbf{y}; \theta_y)$$

$$p(\mathbf{x}; \theta_x, \theta_y) = \int d\mathbf{y}\; p(\mathbf{x} \mid \mathbf{y}; \theta_x) p(\mathbf{y}; \theta_y)$$

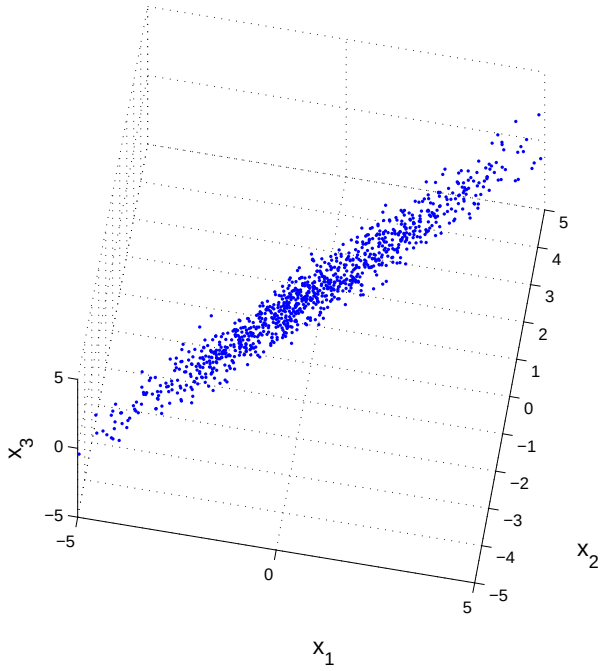
Latent variables and Gaussians



The data were measured in 3D, but are mostly specified by position on a 2D surface.

Principal Components Analysis

Data such as these are often modelled using Principal Components Analysis (PCA).



Assume data $\mathcal{D} = \{\mathbf{x}_i\}$ have zero mean (if not, subtract it!).

- Find direction of greatest variance – $\boldsymbol{\lambda}_{(1)}$.

$$\boldsymbol{\lambda}_{(1)} = \underset{\|\mathbf{v}\|=1}{\operatorname{argmax}} \sum_n (\mathbf{x}_n^\mathsf{T}\mathbf{v})^2$$

- Find direction orthogonal to $\boldsymbol{\lambda}_{(1)}$ with greatest variance – $\boldsymbol{\lambda}_{(2)}$

  ⋮

- Find direction orthogonal to $\{\boldsymbol{\lambda}_{(1)}, \boldsymbol{\lambda}_{(2)}, \ldots, \boldsymbol{\lambda}_{(n-1)}\}$ with greatest variance – $\boldsymbol{\lambda}_{(n)}$.
- Terminate when remaining variance drops below a threshold.

Eigendecomposition of a Covariance Matrix

The eigendecomposition of a covariance matrix makes finding the PCs easy.
Recall that **u** is an eigenvector, with scalar eigenvalue ω, of a matrix C if

$$C\mathbf{u} = \omega\mathbf{u}$$

u can have any norm, but we will define it to be unity (i.e., $\mathbf{u}^{\mathsf{T}}\mathbf{u} = 1$).
For a covariance matrix $C = \left\langle \mathbf{x}\mathbf{x}^{\mathsf{T}} \right\rangle$ (which is $D \times D$, symmetric, positive semi-definite):

- In general there are D eigenvector-eigenvalue pairs $(\mathbf{u}_{(i)}, \omega_{(i)})$, except if two or more eigenvectors share the same eigenvalue (in which case the eigenvectors are degenerate — any linear combination is also an eigenvector).
- The D eigenvectors are orthogonal (or orthogonalisable, if $\omega_{(i)} = \omega_{(j)}$). Thus, they form an orthonormal basis. $\sum_i \mathbf{u}_{(i)}{\mathbf{u}_{(i)}}^{\mathsf{T}} = I$.
- Any vector **v** can be written as

$$\mathbf{v} = \left(\sum_i \mathbf{u}_{(i)}{\mathbf{u}_{(i)}}^{\mathsf{T}}\right)\mathbf{v} = \sum_i ({\mathbf{u}_{(i)}}^{\mathsf{T}}\mathbf{v})\mathbf{u}_{(i)} = \sum_i v_{(i)}\mathbf{u}_{(i)}$$

- The original matrix C can be written:

$$C = \sum_i \omega_{(i)}\mathbf{u}_{(i)}{\mathbf{u}_{(i)}}^{\mathsf{T}} = USU^{\mathsf{T}}$$

where $U = [\mathbf{u}_{(1)}, \mathbf{u}_{(2)}, \ldots, \mathbf{u}_{(D)}]$ collects the eigenvectors and $S = \text{diag}\left[(\omega_{(1)}, \omega_{(2)}, \ldots, \omega_{(D)})\right]$.

PCA and Eigenvectors

- The variance in direction $\mathbf{u}_{(i)}$ is

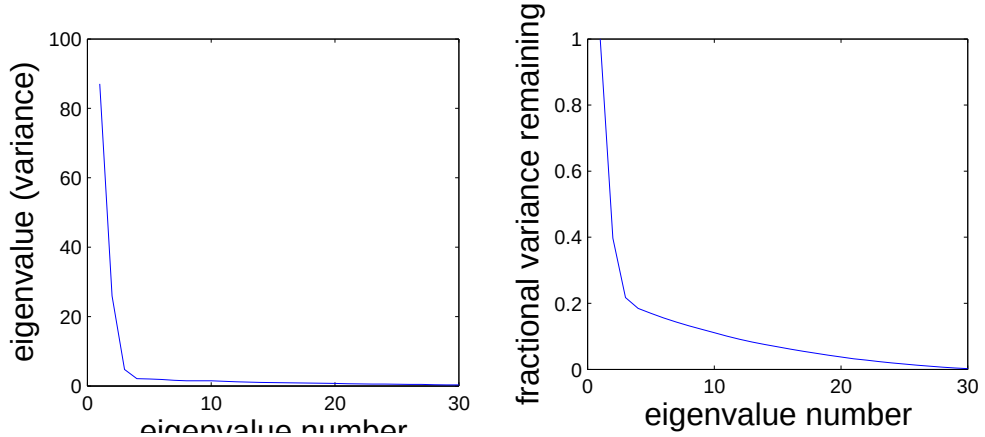
$$\left\langle (\mathbf{x}^{\mathsf{T}}\mathbf{u}_{(i)})^2 \right\rangle = \left\langle \mathbf{u}_{(i)}{}^{\mathsf{T}}\mathbf{x}\mathbf{x}^{\mathsf{T}}\mathbf{u}_{(i)} \right\rangle = \mathbf{u}_{(i)}{}^{\mathsf{T}} C \mathbf{u}_{(i)} = \mathbf{u}_{(i)}{}^{\mathsf{T}} \omega_{(i)} \mathbf{u}_{(i)} = \omega_{(i)}$$

- The variance in an arbitrary direction $\mathbf{v}$ is

$$\left\langle (\mathbf{x}^{\mathsf{T}}\mathbf{v})^2 \right\rangle = \left\langle \left(\mathbf{x}^{\mathsf{T}} \left(\sum_i v_{(i)} \mathbf{u}_{(i)} \right) \right)^2 \right\rangle = \sum_{ij} v_{(i)} \mathbf{u}_{(i)}{}^{\mathsf{T}} C \mathbf{u}_{(j)} v_{(j)}$$

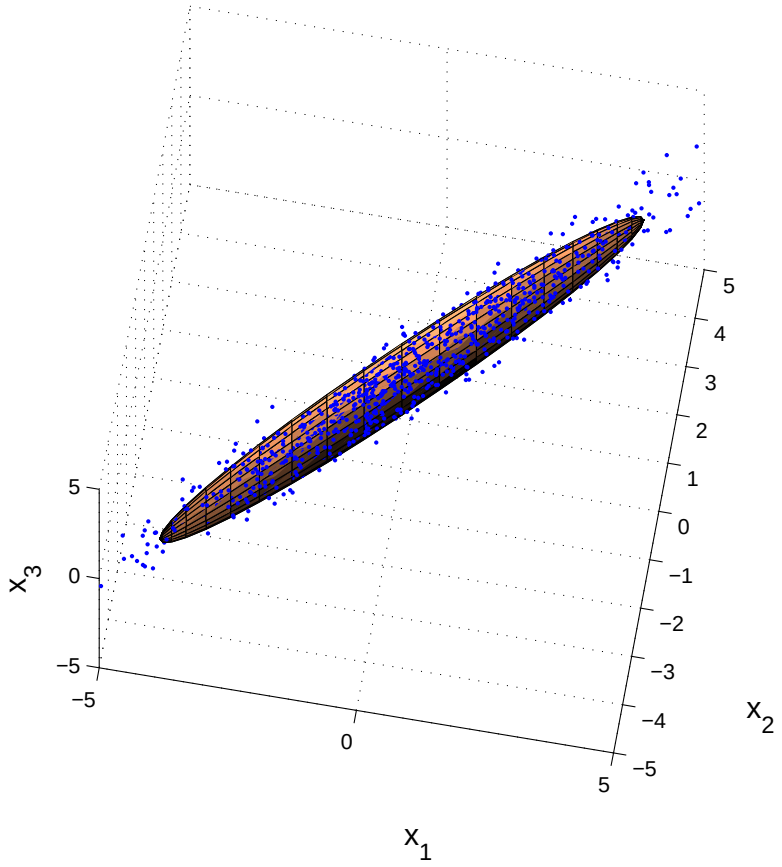
$$= \sum_{ij} v_{(i)} \omega_{(j)} v_{(j)} \mathbf{u}_{(i)}{}^{\mathsf{T}} \mathbf{u}_{(j)} = \sum_i v_{(i)}^2 \omega_{(i)}$$

- If $\mathbf{v}^{\mathsf{T}}\mathbf{v} = 1$, then $\sum_i v_{(i)}^2 = 1$ and so $\operatorname{argmax}_{\|\mathbf{v}\|=1} \left\langle (\mathbf{x}^{\mathsf{T}}\mathbf{v})^2 \right\rangle = \mathbf{u}_{(\text{max})}$
  The direction of greatest variance is the eigenvector the largest eigenvalue.
- In general, the PCs are exactly the eigenvectors of the empirical covariance matrix, ordered by decreasing eigenvalue.
- The eigenspectrum shows how the variance is distributed across dimensions



PCA subspace

The K principle components define the K-dimensional subspace of greatest variance.

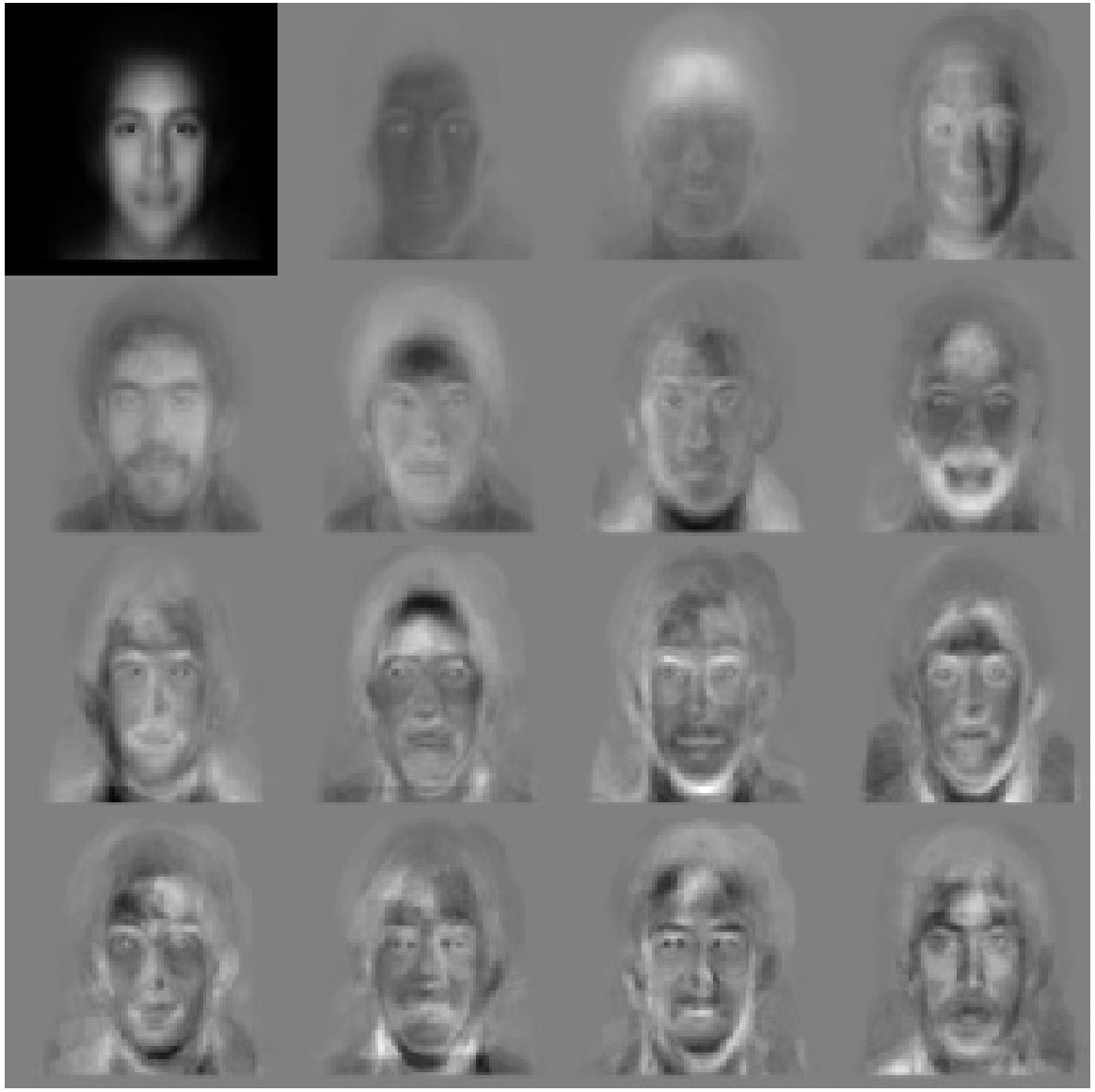


- Each data point $\mathbf{x}_n$ is associated with a projection $\hat{\mathbf{x}}_n$ into the principle subspace.

$$\hat{\mathbf{x}}_n = \sum_{k=1}^{K} x_{n(k)} \boldsymbol{\lambda}_{(k)}$$

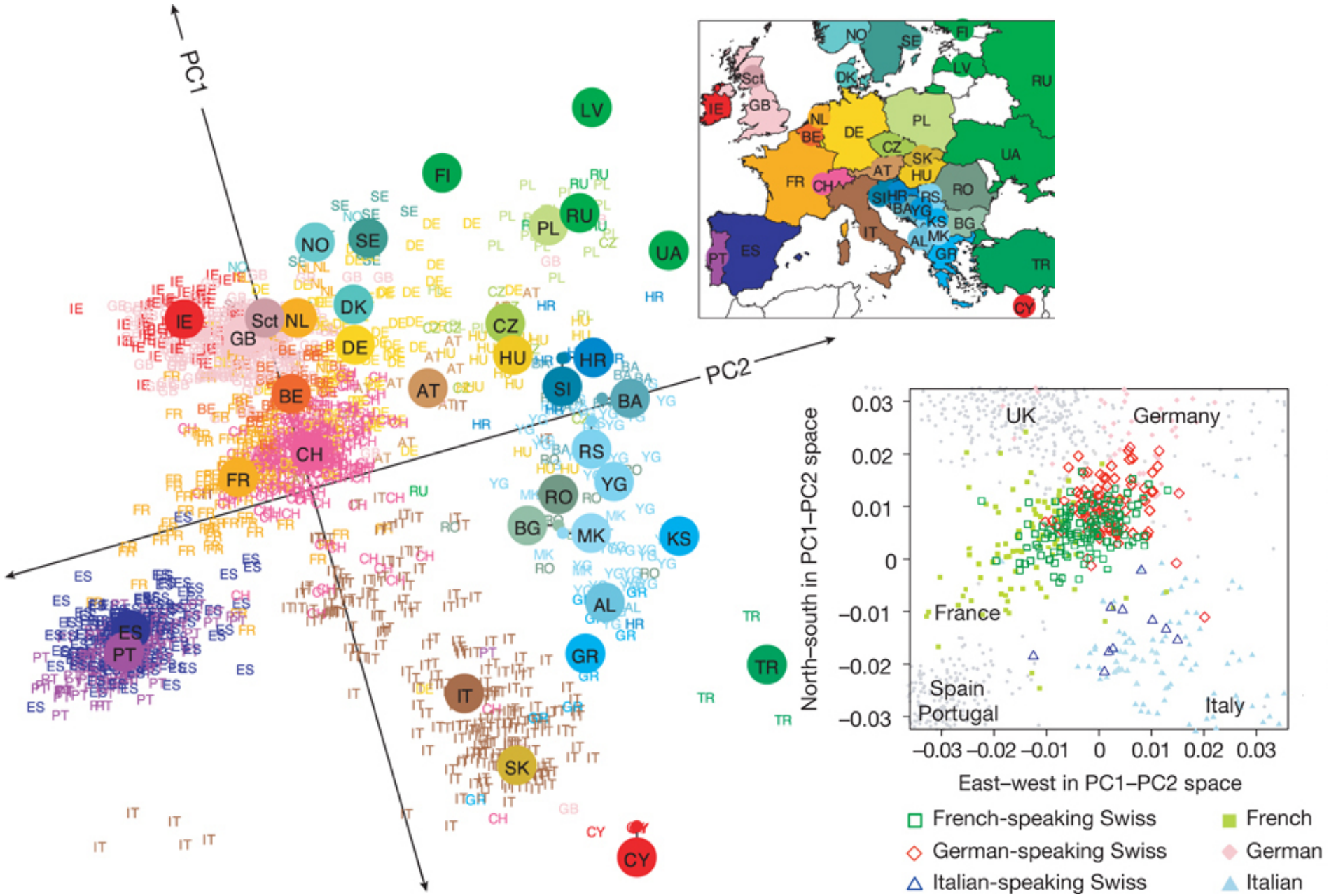
- This can be used for lossy compression, denoising, recognition, . . .

Example of PCA: Eigenfaces



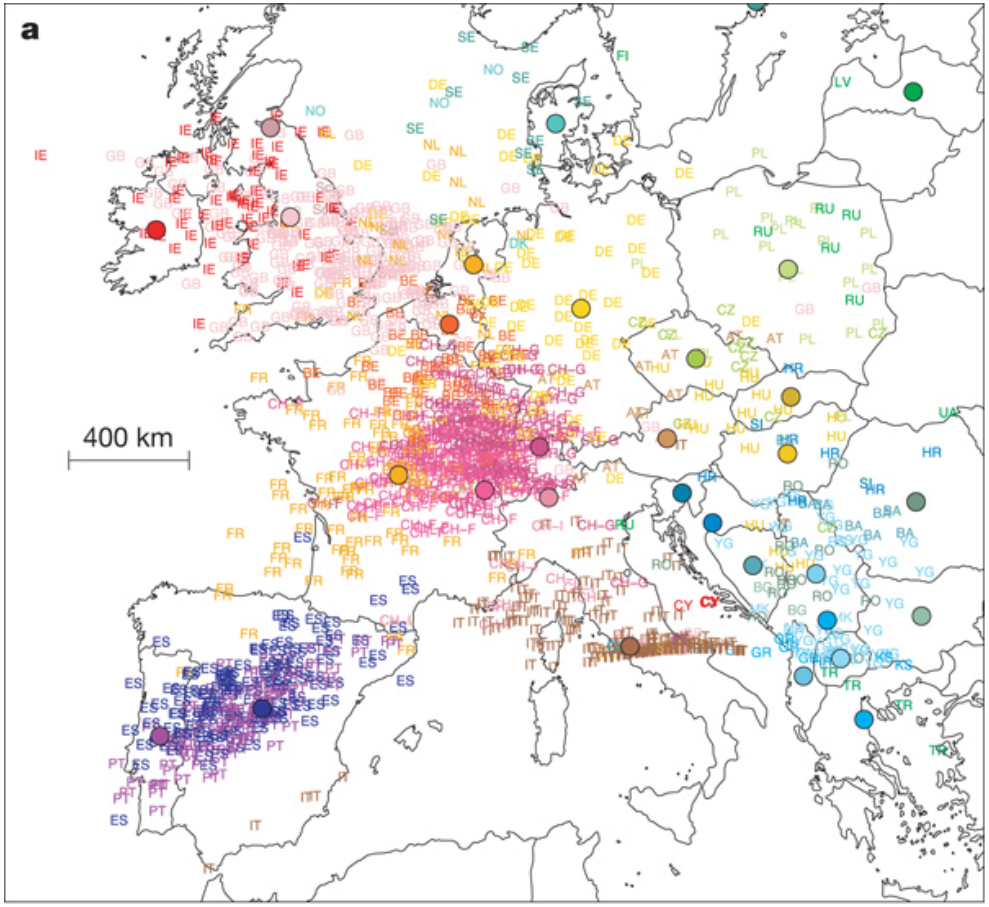
from http://vismod.media.mit.edu/vismod/demos/facerec/basic.html

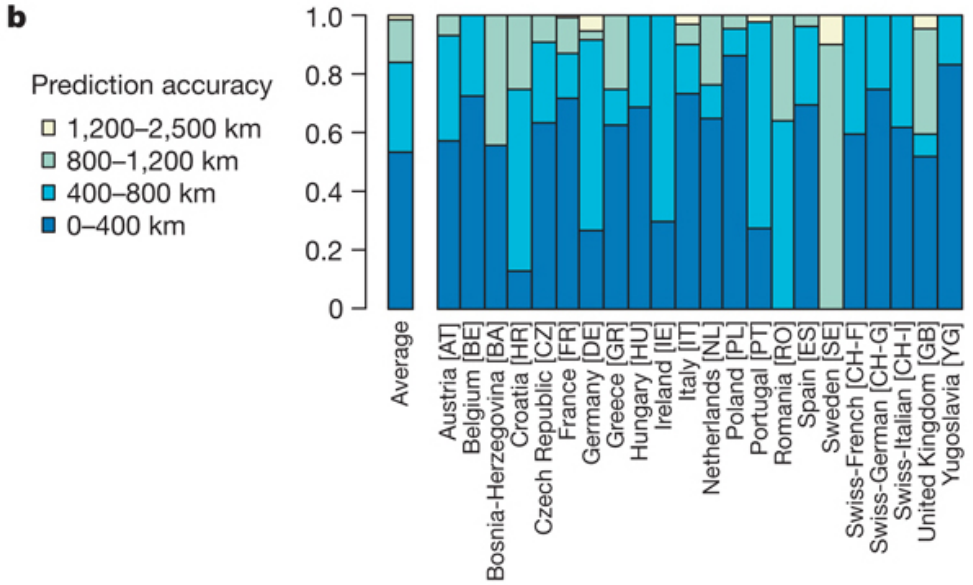
Example of PCA: Genetic variation within Europe



Novembre et al. (2008) Nature 456:98-101

Example of PCA: Genetic variation within Europe





Novembre et al. (2008) Nature 456:98-101

Another view of PCA: Mutual Information

Problem: Given $\mathbf{x}$, find $\mathbf{y} = A\mathbf{x}$ with columns of A unit vectors, s.t. $I(\mathbf{y}; \mathbf{x})$ is maximised (assuming that $P(\mathbf{x})$ is Gaussian).

$$I(\mathbf{y}; \mathbf{x}) = H(\mathbf{y}) + H(\mathbf{x}) - H(\mathbf{y}, \mathbf{x}) = H(\mathbf{y})$$

So we want to maximise the entropy of $\mathbf{y}$. What is the entropy of a Gaussian?

$$H(\mathbf{z}) = -\int d\mathbf{z}\, p(\mathbf{z}) \ln p(\mathbf{z}) = \frac{1}{2} \ln |\Sigma| + \frac{D}{2}(1 + \ln 2\pi)$$

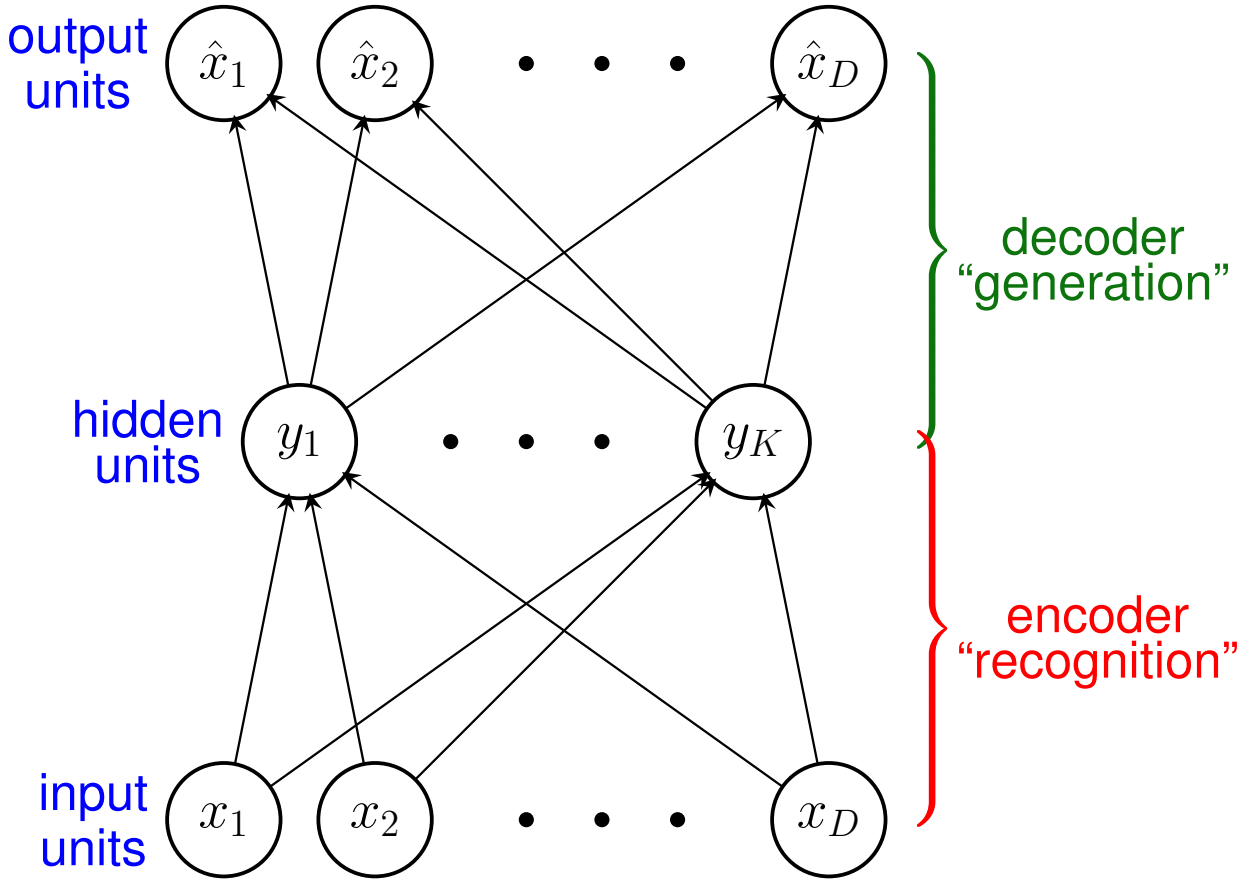
Therefore we want the distribution of $\mathbf{y}$ to have largest volume (i.e. det of covariance matrix).

$$\Sigma_y = A\Sigma_x A^\top = AUS_x U^\top A^\top$$

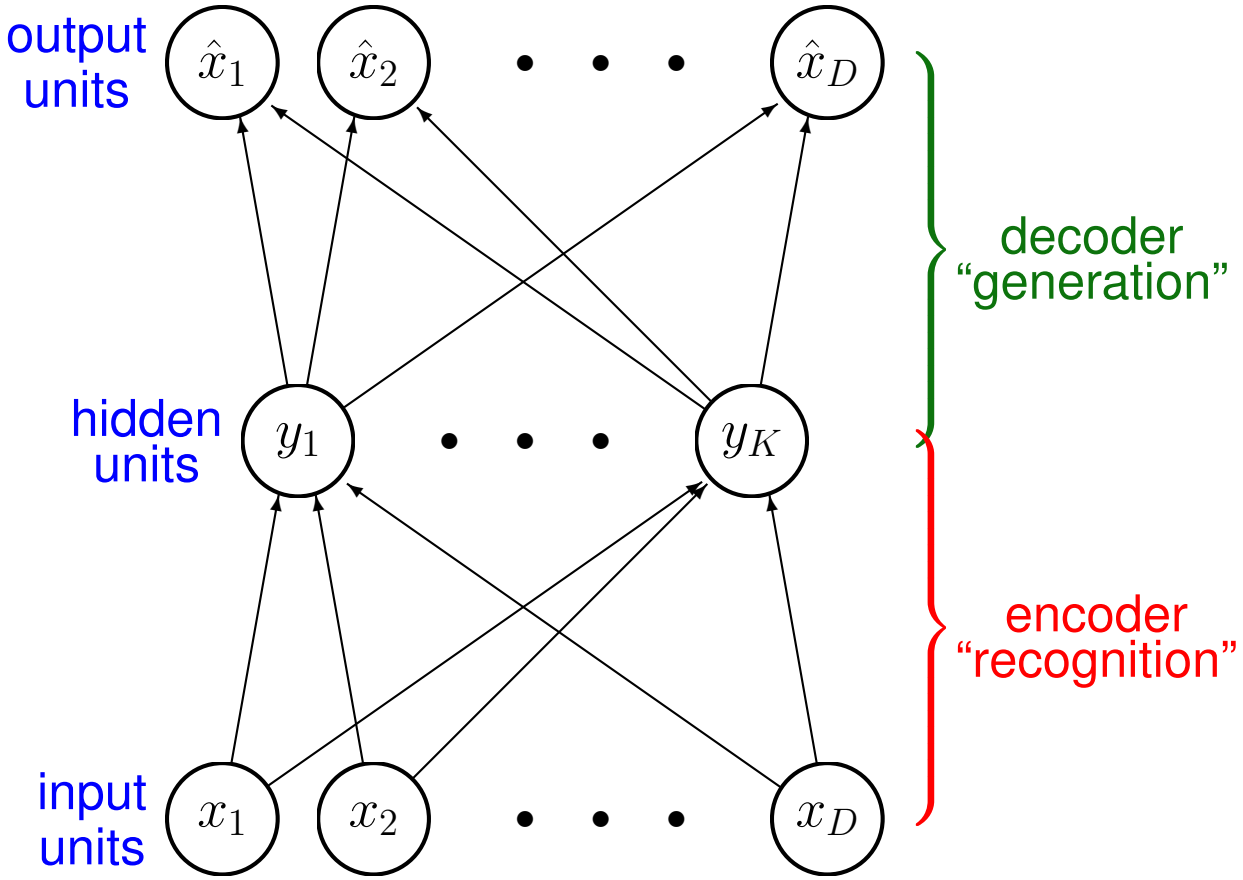
So, A should be aligned with the columns of U which are associated with the largest eigenvalues (variances).

Projection to the principal component subspace preserves the most information about the (Gaussian) data.

Network Interpretations and Encoder-Decoder Duality



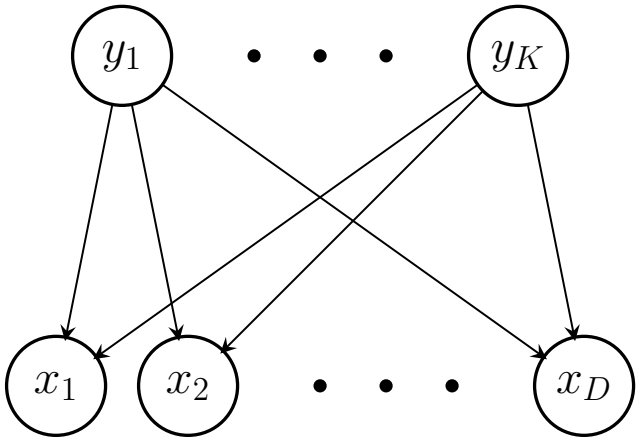
From Supervised Learning to PCA



A linear autoencoder neural network trained to minimise squared error learns to perform PCA (Baldi & Hornik, 1989).

Probabilistic PCA

We could think of the projection $\tilde{\mathbf{x}}_n$ as a latent variable, but it is more usual to construct a standard normal latent $\mathbf{y}$.



Observed vector data $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}; \mathbf{x}_i \in \mathbb{R}^D$

Assumed latent variables $\{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N\}; \mathbf{y}_i \in \mathbb{R}^K$

Linear generative model: $x_d = \sum_{k=1}^{K} \Lambda_{dk}\, y_k + \epsilon_d$

- y_k are independent $\mathcal{N}(0, 1)$ Gaussian factors
- ϵ_d are independent $\mathcal{N}(0, \psi)$ Gaussian noise
- $K < D$

So, model for observations $\mathbf{x}$ is still Gaussian with:

$$p(\mathbf{y}) = \mathcal{N}(0, I)$$

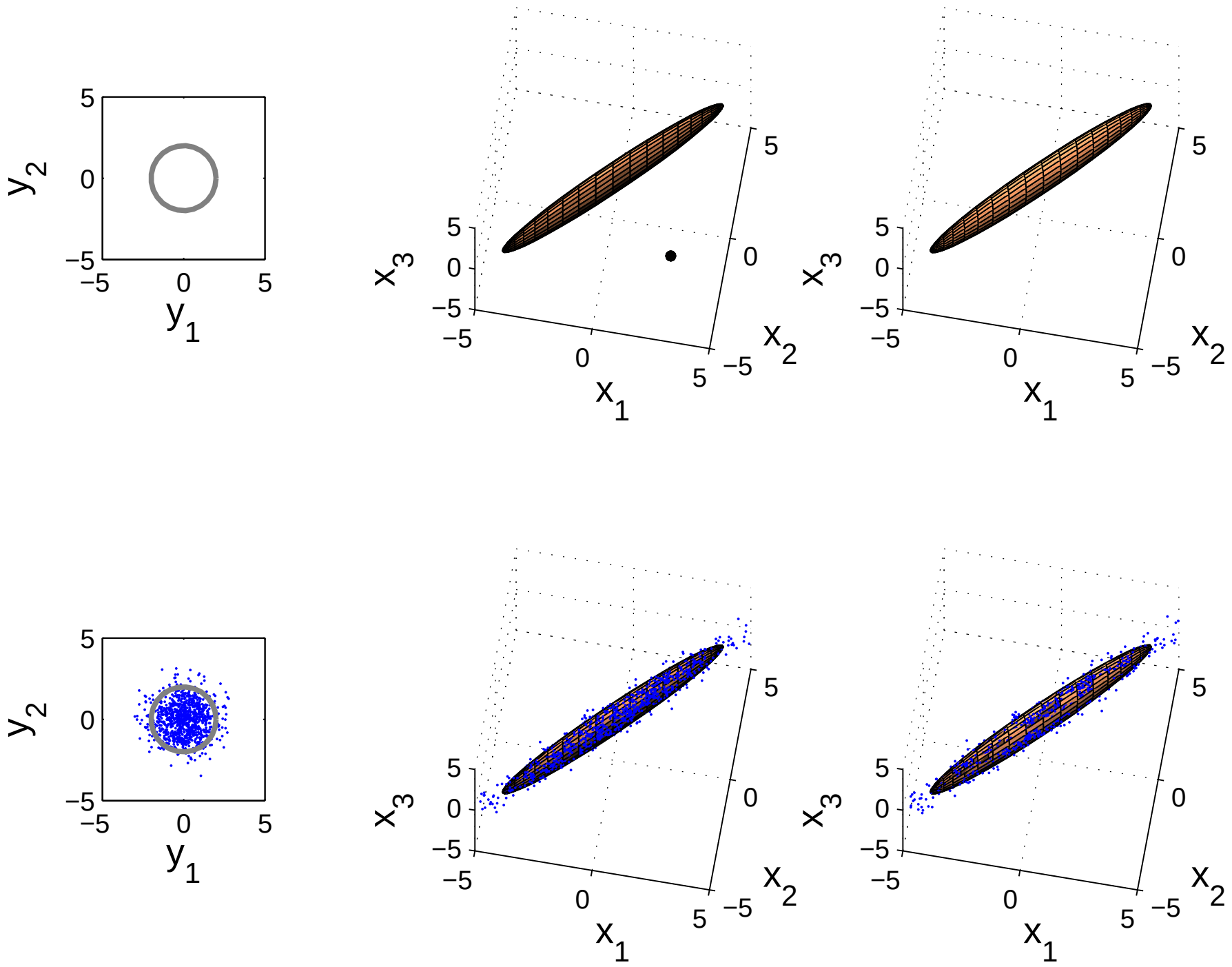
$$p(\mathbf{x}|\mathbf{y}) = \mathcal{N}(\Lambda\mathbf{y}, \psi I)$$

$$p(\mathbf{x}) = \int p(\mathbf{y})p(\mathbf{x}|\mathbf{y})d\mathbf{y} = \mathcal{N}\left(0, \Lambda\Lambda^\top + \psi I\right)$$

where Λ is a $D \times K$ matrix.

This is Probabilistic PCA (pPCA). The maximum-likelihood parameter Λ, for fixed ψ, lies in the K-principal subspace.

Probabilistic PCA



PCA and pPCA

- In PCA the "noise" is orthogonal to the subspace, and we can project $\mathbf{x}_n \to \tilde{\mathbf{x}}_n$ trivially.
- In pPCA, the noise is more sensible (equal in all directions). But what is the projection?

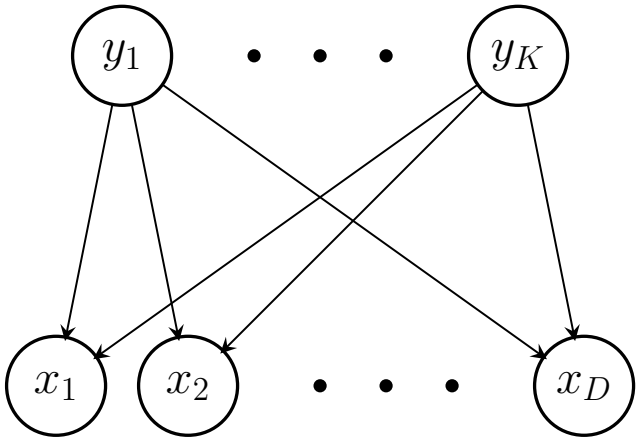
  Find the expected value of $\mathbf{y}_n|\mathbf{x}_n$ and then take $\tilde{\mathbf{x}}_n = \Lambda\bar{\mathbf{y}}_n$.
- **Tactic:** write $p(\mathbf{y}_n, \mathbf{x}_n|\theta)$, consider $\mathbf{x}_n$ to be fixed. What is this as a function of $\mathbf{y}_n$?

$$
\begin{aligned}
p(\mathbf{y}_n, \mathbf{x}_n) &= p(\mathbf{y}_n)p(\mathbf{x}_n|\mathbf{y}_n) \\
&= (2\pi)^{-\frac{K}{2}} \exp\{-\frac{1}{2}\mathbf{y}_n{}^\top\mathbf{y}_n\}\ |2\pi\Psi|^{-\frac{1}{2}} \exp\{-\frac{1}{2}(\mathbf{x}_n - \Lambda\mathbf{y}_n)^\top\Psi^{-1}(\mathbf{x}_n - \Lambda\mathbf{y}_n)\} \\
&= \text{c} \times \exp\{-\frac{1}{2}[\mathbf{y}_n{}^\top\mathbf{y}_n + (\mathbf{x}_n - \Lambda\mathbf{y}_n)^\top\Psi^{-1}(\mathbf{x}_n - \Lambda\mathbf{y}_n)]\} \\
&= \text{c'} \times \exp\{-\frac{1}{2}[\mathbf{y}_n{}^\top(I + \Lambda^\top\Psi^{-1}\Lambda)\mathbf{y}_n - 2\mathbf{y}_n{}^\top\Lambda^\top\Psi^{-1}\mathbf{x}_n]\} \\
&= \text{c''} \times \exp\{-\frac{1}{2}[\mathbf{y}_n{}^\top\Sigma^{-1}\mathbf{y}_n - 2\mathbf{y}_n{}^\top\Sigma^{-1}\mu + \mu^\top\Sigma^{-1}\mu]\}
\end{aligned}
$$

  So $\Sigma = (I + \Lambda^\top\Psi^{-1}\Lambda)^{-1} = I - \beta\Lambda$ and $\mu = \Sigma\Lambda^\top\Psi^{-1}\mathbf{x}_n = \beta\mathbf{x}_n$. Where $\beta = \Sigma\Lambda^\top\Psi^{-1}$.
- Thus, $\tilde{\mathbf{x}}_n = \Lambda(I + \Lambda^\top\Psi^{-1}\Lambda)^{-1}\Lambda^\top\Psi^{-1}\mathbf{x}_n = \mathbf{x}_n - \Psi(\Lambda\Lambda^\mathsf{T} + \Psi)^{-1}\mathbf{x}_n$
- This is not the same projection. pPCA takes into account noise in the principal subspace.
- As $\psi \to 0$, pPCA $\to$ PCA.

Factor Analysis

If dimensions are not equivalent, equal variance makes no sense.



Observed vector data $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}; \mathbf{x}_i \in \mathbb{R}^D$

Assumed latent variables $\{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N\}; \mathbf{y}_i \in \mathbb{R}^K$

Linear generative model: $x_d = \sum_{k=1}^{K} \Lambda_{dk}\, y_k + \epsilon_d$

- y_k are independent $\mathcal{N}(0, 1)$ Gaussian factors
- ϵ_d are independent $\mathcal{N}(0, \Psi_{dd})$ Gaussian noise
- $K < D$

So, model for observations **x** is still Gaussian with:

$$p(\mathbf{y}) = \mathcal{N}(0, I)$$

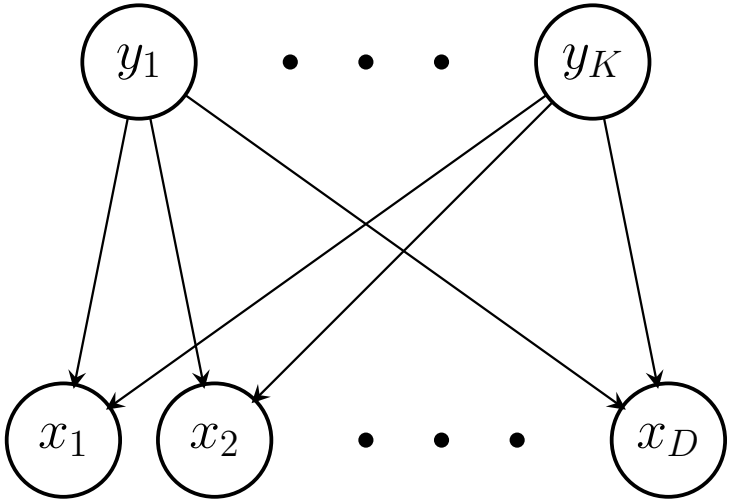
$$p(\mathbf{x}|\mathbf{y}) = \mathcal{N}(\Lambda\mathbf{y}, \Psi)$$

$$p(\mathbf{x}) = \int p(\mathbf{y})p(\mathbf{x}|\mathbf{y})d\mathbf{y} = \mathcal{N}\left(0, \Lambda\Lambda^\top + \Psi\right)$$

where Λ is a $D \times K$ matrix, and Ψ is $K \times K$ and diagonal.

Dimensionality Reduction: Finds a low-dimensional projection of high dimensional data that captures the correlation structure of the data.

Factor Analysis (cont.)



- ML learning finds Λ and Ψ given data
- parameters (corrected for symmetries): $DK + D - \frac{K(K-1)}{2}$
- If number of parameters $> \frac{D(D+1)}{2}$ model is not identifiable
- no closed form solution for ML params: $\mathcal{N}(0, \Lambda\Lambda^\top + \Psi)$

- [Bayesian treatment would also have priors over Λ and Ψ and would average over them for prediction.]

Factor Analysis Projections

Our analysis for pPCA still applies:

$$\tilde{\mathbf{x}}_n = \Lambda(I + \Lambda^\top \Psi^{-1}\Lambda)^{-1}\Lambda^\top \Psi^{-1}\mathbf{x}_n = \mathbf{x}_n - \Psi(\Lambda\Lambda^\mathsf{T} + \Psi)^{-1}\mathbf{x}_n$$

but now Ψ is diagonal but not spherical.
Note, though, that Λ is generally different from that found by pPCA.

Gradient Methods of Learning FA

Write down negative log likelihood:

$$\frac{1}{2}\log|2\pi(\Lambda\Lambda^\top+\Psi)|+\frac{1}{2}\mathbf{x}^\top(\Lambda\Lambda^\top+\Psi)^{-1}\mathbf{x}$$

Optimise w.r.t. Λ and Ψ (need matrix calculus) subject to constraints

We will soon see an easier way to learn latent variable models...

FA vs PCA

- PCA and pPCA are rotationally invariant; FA is not

$$\text{If } \mathbf{x} \to U\mathbf{x} \text{ for unitary } U, \quad \text{then } \boldsymbol{\lambda}_{(i)}^{\text{PCA}} \to U\boldsymbol{\lambda}_{(i)}^{\text{PCA}}$$

- FA is measurement scale invariant; PCA and pPCA are not

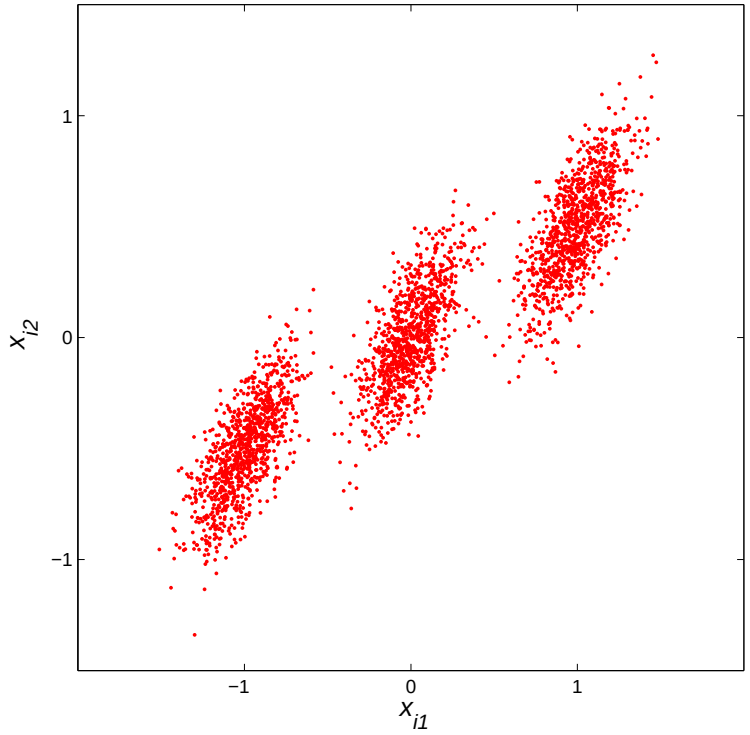
$$\text{If } \mathbf{x} \to S\mathbf{x} \text{ for diagonal } S, \quad \text{then } \boldsymbol{\lambda}_{(i)}^{\text{FA}} \to S\boldsymbol{\lambda}_{(i)}^{\text{FA}}$$

- FA and pPCA define a probabilistic model; PCA does not

Limitations of Gaussian, FA and PCA models

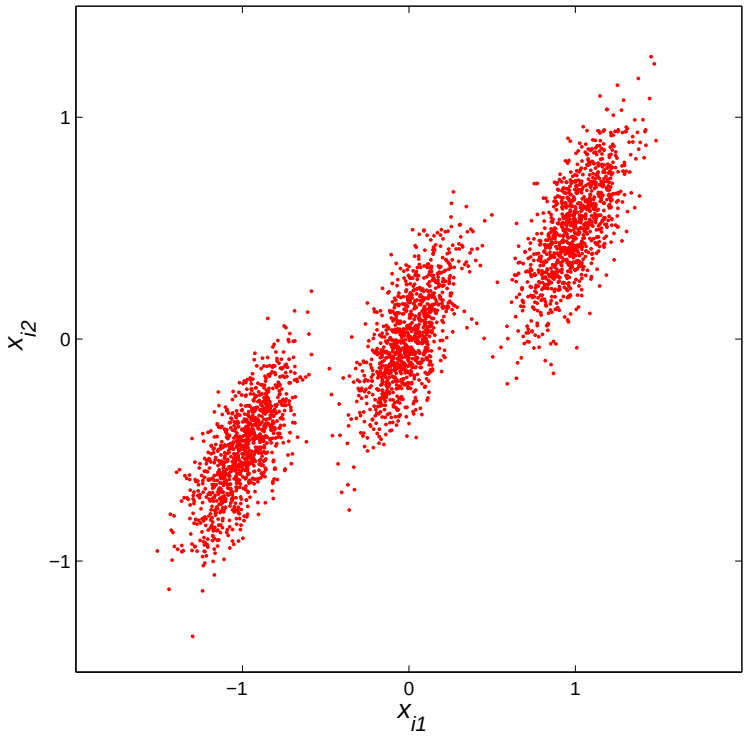
- Gaussian, FA and PCA models are easy to understand and use in practice.
- They are a convenient way to find interesting directions in very high dimensional data sets, eg as preprocessing
- Their problem is that they make very strong assumptions about the distribution of the data, only the mean and variance of the data are taken into account.

The class of densities which can be modelled is too restrictive.



By using *mixtures* of simple distributions, such as Gaussians, we can expand the class of densities greatly.

Mixture Distributions



A mixture distribution has a single discrete latent variable:

$$s_i \overset{\text{iid}}{\sim} Discrete[\boldsymbol{\pi}]$$
$$\mathbf{x}_i \mid s_i \sim \mathcal{P}_{s_i}[\theta_{s_i}]$$

Mixtures arise naturally when observations from different sources have been collated. They can also be used to *approximate* arbitrary distributions.

The Mixture Likelihood

The mixture model is

$$s_i \overset{\text{iid}}{\sim} \; Discrete[\boldsymbol{\pi}]$$
$$\mathbf{x}_i \mid s_i \sim \mathcal{P}_{s_i}[\theta_{s_i}]$$

Under the discrete distribution

$$P(s_i = m) = \pi_m; \qquad \pi_m \geq 0, \; \sum_{m=1}^{k} \pi_m = 1$$

Thus, the probability (density) at a single data point $\mathbf{x}_i$ is

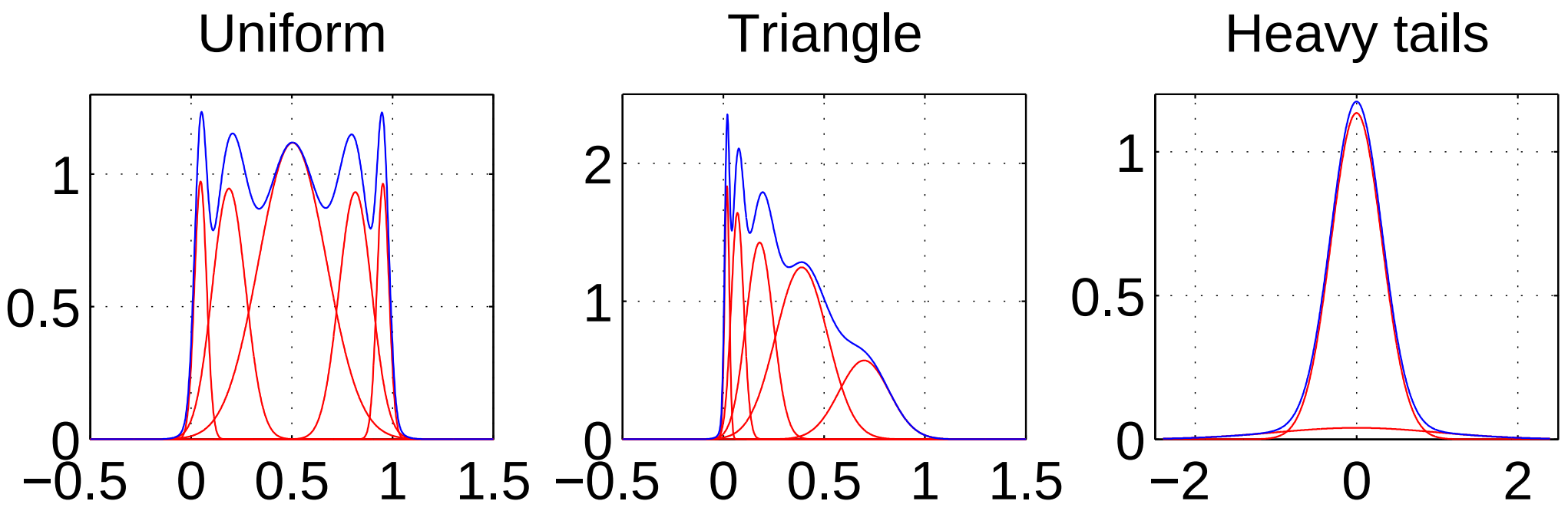
$$\begin{aligned} P(\mathbf{x}_i) &= \sum_{m=1}^{k} P(\mathbf{x}_i \mid s_i = m) P(s_i = m) \\ &= \sum_{m=1}^{k} \pi_m P_m(\mathbf{x}_i; \theta_m) \end{aligned}$$

The mixture distribution (density) is a convex combination (or *weighted average*) of the component distributions (densities).

Approximation with a Mixture of Gaussians (MoG)

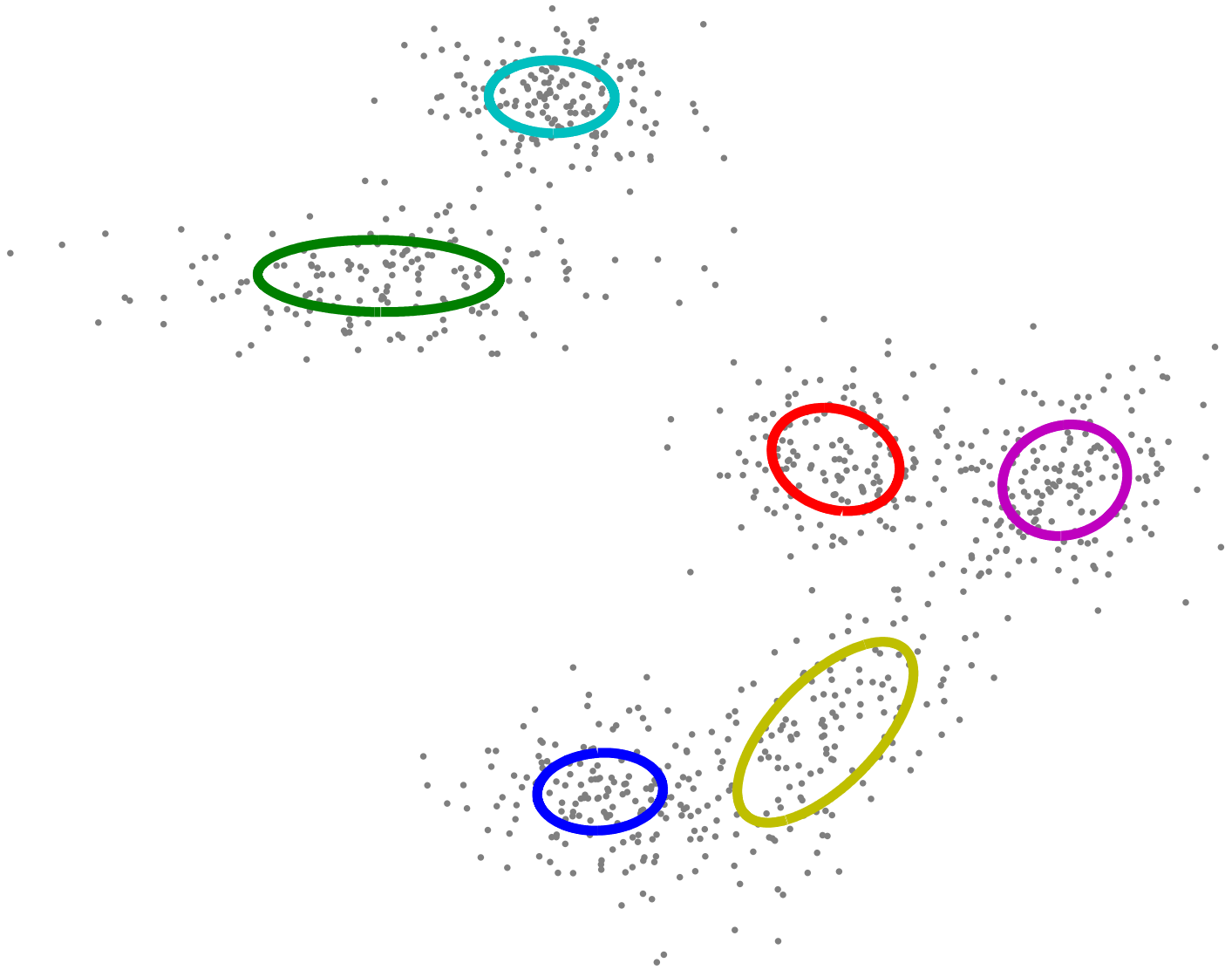
The component densities may be viewed as elements of a *basis* which can be combined to approximate arbitrary distributions.

Here are examples where non-Gaussian densities are modelled (aproximated) as a mixture of Gaussians. The red curves show the (weighted) Gaussians, and the blue curve the resulting density.



Given enough mixture components we can model (almost) any density (as accurately as desired), but still only need to work with the well-known Gaussian form.

Clustering with a MoG



Clustering with a MoG

In clustering applications, the latent variable s_i represents the (unknown) identity of the cluster to which the ith observation belongs.

Thus, the latent distribution gives the *prior* probability of a data point coming from each cluster.

$$P(s_i = m \mid \pi) = \pi_m$$

Data from the mth cluster are distributed according to the mth component:

$$P(\mathbf{x}_i \mid s_i = m) = P_m(\mathbf{x}_i)$$

Once we observe a data point, the *posterior* probability distribution for the cluster it belongs to is

$$P(s_i = m \mid \mathbf{x}_i) = \frac{P_m(\mathbf{x}_i)\pi_m}{\sum_m P_m(\mathbf{x}_i)\pi_m}$$

This is often called the responsibility of the mth cluster for the ith data point.

The MoG likelihood

Each component of a MoG is a Gaussian, with mean $\boldsymbol{\mu}_m$ and covariance matrix Σ_m. Thus, the probability density evaluated at a set of n iid observations, $\mathcal{D} = \{\mathbf{x}_1 \dots \mathbf{x}_n\}$ (i.e. the likelihood) is

$$\begin{aligned} p(\mathcal{D} \mid \{\boldsymbol{\mu}_m\}, \{\Sigma_m\}, \boldsymbol{\pi}) &= \prod_{i=1}^{n} \sum_{m=1}^{k} \pi_m \mathcal{N}\left(\mathbf{x}_i \mid \boldsymbol{\mu}_m, \Sigma_m\right) \\ &= \prod_{i=1}^{n} \sum_{m=1}^{k} \pi_m \left|2\pi\Sigma_m\right|^{-1/2} \exp\left[-\frac{1}{2}(\mathbf{x}_i - \boldsymbol{\mu}_m)^{\mathsf{T}} \Sigma_m^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_m)\right] \end{aligned}$$

The log of the likelihood is

$$\log p(\mathcal{D} \mid \{\boldsymbol{\mu}_m\}, \{\Sigma_m\}, \boldsymbol{\pi}) = \sum_{i=1}^{n} \log \sum_{m=1}^{k} \pi_m \left|2\pi\Sigma_m\right|^{-1/2} \exp\left[-\frac{1}{2}(\mathbf{x}_i - \boldsymbol{\mu}_m)^{\mathsf{T}} \Sigma_m^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_m)\right]$$

Note that the logarithm fails to simplify the component density terms. A mixture distribution does not lie in the exponential family. Direct optimisation is not easy.

Maximum Likelihood for a Mixture Model

The log likelihood is:

$$\mathcal{L} = \sum_{i=1}^{n} \log \sum_{m=1}^{k} \pi_m P_m(\mathbf{x}_i; \theta_m)$$

Its partial derivative wrt θ_m is

$$\frac{\partial \mathcal{L}}{\partial \theta_m} = \sum_{i=1}^{n} \frac{\pi_m}{\sum_{m=1}^{k} \pi_m P_m(\mathbf{x}_i; \theta_m)} \frac{\partial P_m(\mathbf{x}_i; \theta_m)}{\partial \theta_m}$$

or, using $\partial P / \partial \theta = P \times \partial \log P / \partial \theta$,

$$= \sum_{i=1}^{n} \underbrace{\frac{\pi_m P_m(\mathbf{x}_i; \theta_m)}{\sum_{m=1}^{k} \pi_m P_m(\mathbf{x}_i; \theta_m)}} \frac{\partial \log P_m(\mathbf{x}_i; \theta_m)}{\partial \theta_m}$$

$$= \sum_{i=1}^{n} r_{im} \frac{\partial \log P_m(\mathbf{x}_i; \theta_m)}{\partial \theta_m}$$

And its partial derivative wrt π_m is

$$\frac{\partial \mathcal{L}}{\partial \pi_m} = \sum_{i=1}^{n} \frac{P_m(\mathbf{x}_i; \theta_m)}{\sum_{m=1}^{k} \pi_m P_m(\mathbf{x}_i; \theta_m)} = \sum_{i=1}^{n} \frac{r_{im}}{\pi_m}$$

MoG Derivatives

For a MoG, with $\theta_m = \{\boldsymbol{\mu}_m, \Sigma_m\}$ we get

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\mu}_m} = \sum_{i=1}^{n} r_{im} \Sigma_m^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_m)$$

$$\frac{\partial \mathcal{L}}{\partial \Sigma_m^{-1}} = \frac{1}{2} \sum_{i=1}^{n} r_{im} \left(\Sigma_m - (\mathbf{x}_i - \boldsymbol{\mu}_m)(\mathbf{x}_i - \boldsymbol{\mu}_m)^{\mathsf{T}} \right)$$

These equations can be used (along with the derivatives wrt to π_m) for gradient based learning; e.g., taking small steps in the direction of the gradient (or using conjugate gradients).

The K-means Algorithm

The K-means algorithm is a limiting case of the mixture of Gaussians (c.f. PCA and Factor Analysis).

Take $\pi_m = 1/k$ and $\Sigma_m = \sigma^2 I$, with $\sigma^2 \to 0$. Then the responsibilities become binary

$$r_{im} \to \delta(m, \underset{l}{\operatorname{argmin}} \|\mathbf{x}_i - \boldsymbol{\mu}_l\|^2)$$

with 1 for the component with the closest mean and 0 for all other components. We can then solve directly for the means by setting the gradient to 0.

The k-means algorithm iterates these two steps:

- assign each point to its closest mean $\left(\text{set } r_{im} = \delta(m, \underset{l}{\operatorname{argmin}} \|\mathbf{x}_i - \boldsymbol{\mu}_l\|^2)\right)$
- update the means to the average of their assigned points $\left(\text{set } \boldsymbol{\mu}_m = \frac{\sum_i r_{im}\mathbf{x}_i}{\sum_i r_{im}}\right)$

This usually converges within a few iterations, although the fixed point depends on the initial values chosen for $\boldsymbol{\mu}_m$. The algorithm has no learning rate, but the assumptions are quite limiting.

A preview of the EM algorithm

We wrote the k-means algorithm in terms of binary responsibilities. Suppose, instead, we used the fractional responsibilities from the full (non-limiting) MoG, but still neglected the dependence of the responsibilities on the parameters. We could then solve for both $\boldsymbol{\mu}_m$ and Σ_m.

The EM algorithm for MoGs iterates these two steps:

- Evaluate the responsibilities for each point given the current parameters.
- Optimise the parameters assuming the responsibilities stay fixed:

$$\boldsymbol{\mu}_m = \frac{\sum_i r_{im}\mathbf{x}_i}{\sum_i r_{im}} \quad \text{and} \quad \Sigma_m = \frac{\sum_i r_{im}(\mathbf{x}_i - \boldsymbol{\mu}_m)(\mathbf{x}_i - \boldsymbol{\mu}_m)^\mathsf{T}}{\sum_i r_{im}}$$

Although this appears *ad hoc*, we will see (later) that it is a special case of a general algorithm, and is actually guaranteed to increase the likelihood at each iteration.

Issues

There are several problems with the new algorithms:

- slow convergence for the gradient based method
- gradient based method may develop invalid covariance matrices
- local minima; the end configuration may depend on the starting state
- how do you adjust k? Using the likelihood alone is no good.
- singularities; components with a single data point will have their covariance going to zero and the likelihood will tend to infinity.

Issues