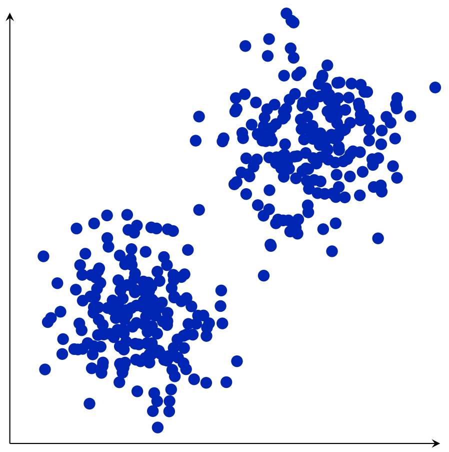
Lecture 11

- Factor analysis
- Principal component analysis
- Latent semantic indexing
- Independent component analysis

1 Factor analysis

1.1 Intuition

For the following dataset the mixture of Gaussians model can be very effective:



If the number of training example m is much bigger than the number of features n then the mixture of Gaussians is the best choice. In practice the situation when the number of features greater (or approximately the same) than number of training examples is very often. In this section we consider the algorithm that works fine for the case $n >> m$ or $n \approx m$.

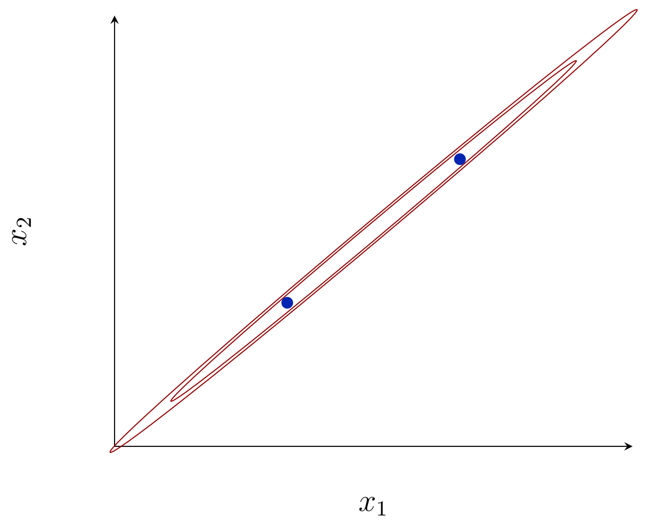
Our goal is to estimate the probability density $p(x)$ given a training set $\{x^{(1)}, \ldots, x^{(m)}\}$.

If we assume that $x \sim N(\mu, \Sigma)$, where $\Sigma \in \mathbb{R}^{n \times n}$, then maximum likelihood estimation gives a solution

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)},$$

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \mu)(x^{(i)} - \mu)^T.$$

If the number of samples is small the matrix Σ is singular (contains zero eigenvalues), which means that it is not invertible and calculation of the probability density gives $\frac{0}{0}$ undetermined form. The simple illustration of this situation is:



In this case $m = n = 2$ and the estimated density will be infinitely "long" along the axis through these two points. To fix this problem we can add some restrictions for Σ.

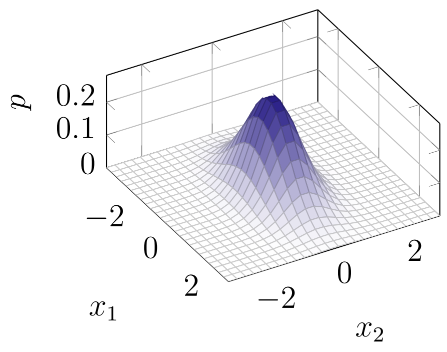
First option is to make Σ to be diagonal:

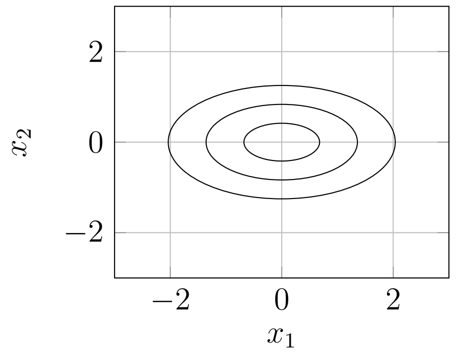
$$\Sigma = \begin{pmatrix} \sigma_1^2 & 0 & \cdots & 0 & 0 \\ 0 & \sigma_2^2 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & \sigma_{n-1}^2 & 0 \\ 0 & 0 & \cdots & 0 & \sigma_n^2 \end{pmatrix} \in \mathbb{R}^{n\times n},$$

then maximum likelihood estimates gives

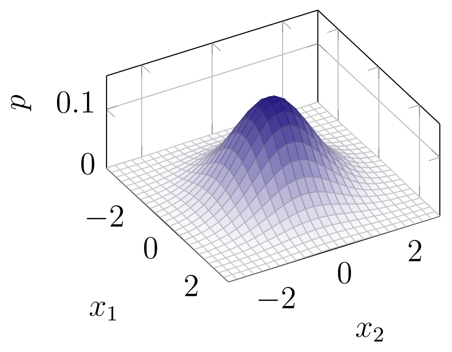
$$\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)} - \mu_j)^2$$

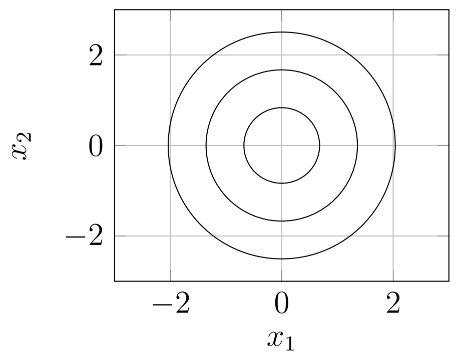
and main axes of Gaussian distribution will parallel to the coordinate axes:





More restrictive assumption could be $\Sigma = \sigma^2 I$, where I is a unit matrix, in this case the cross sections of Gaussian probability density become circular:





If we add such restrictions for the covariance matrix then we also assume that features in our training dataset are uncorrelated or independent which is not true in many cases. In the factor analysis we add some restrictions to avoid the singularity we described before, but we also try to catch some correlations between features in the dataset.

We introduce the latent variables $z \in N(\vec{0}, I)$, $z \in \mathbb{R}^d (d < n)$ and make the following assumptions:

$$x \,|\, z \sim N(\mu + \Lambda z, \Psi),$$

or, equivalently,

$$x = \mu + \Lambda z + \varepsilon,$$

where

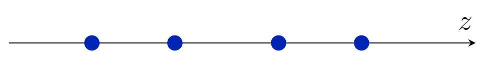
$$\varepsilon \sim N(\vec{0}, \Psi).$$

The parameters of the model are $\mu \in \mathbb{R}^n$, $\Lambda \in \mathbb{R}^{n\times d}$, $\Psi \in \mathbb{R}^{n\times n}$. We introduce an additional assumption that Ψ is a diagonal.

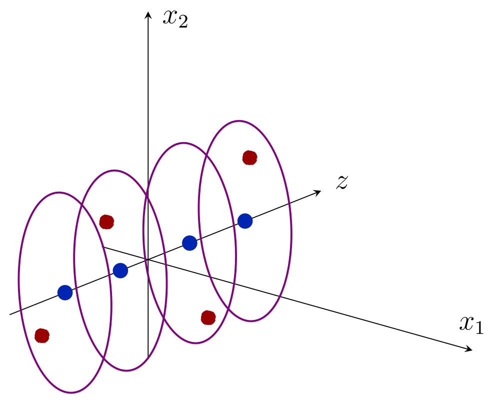
Example. For the case $z \in \mathbb{R}$, $x \in \mathbb{R}^2$ with the parameters

$$\Lambda = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \; \Psi = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}, \; \mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

the idea of the factor analysis is visualized as



$z \in \mathbb{R}$ (blue points)



$x \in \mathbb{R}^2$ (red points)

Similarly, we can take $z \in \mathbb{R}^2$ and $x \in \mathbb{R}^3$. Then the transformation $x = \mu + \Lambda z$ is a mapping of the 2D space to the 3D space, the transformation $x = \mu + \Lambda z + \varepsilon$ is the same mapping but with the added Gaussian noise.

1.2 Marginal and conditionals for Gaussians

Consider the vector partitioning of the gaussian random variable X

$$X = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \sim N(\mu, \Sigma),$$

where $X_1 \in \mathbb{R}^r$, $X_2 \in \mathbb{R}^s$, $X \in \mathbb{R}^{r+s}$. Then

$$\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}$$

and

$$\Sigma = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} = \begin{bmatrix} E[(X_1-\mu_1)(X_1-\mu_1)^T] & E[(X_1-\mu_1)(X_2-\mu_2)^T] \\ E[(X_2-\mu_2)(X_1-\mu_1)^T] & E[(X_2-\mu_2)(X_2-\mu_2)^T] \end{bmatrix},$$

where $\mu_1 \in \mathbb{R}^r$, $\mu_2 \in \mathbb{R}^s$, $\Sigma_{11} \in \mathbb{R}^{r\times r}$, $\Sigma_{12} \in \mathbb{R}^{r\times s}$, $\Sigma_{21} \in \mathbb{R}^{s\times r}$ and $\Sigma_{22} \in \mathbb{R}^{s\times s}$.

Then the random variable $X_1 \sim N(\mu_1, \Sigma_{11})$ is a gaussian random variable with the probability density

$$p(x_1) = \int\limits_{x_2} p(x_1, x_2)dx_2.$$

The conditional density function can be calculated as

$$p(x_1 \,|\, x_2) = \frac{p(x_1, x_2)}{p(x_2)},$$

where numerator and denominator are known. Using this formula it can be shown that

$$X_1 \,|\, X_2 \sim N(\mu_{1\,|\,2}, \Sigma_{1\,|\,2}),$$

where

$$\mu_{1\,|\,2} = \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2) \tag{1}$$

and

$$\Sigma_{1\,|\,2} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}. \tag{2}$$

These facts will be useful for the next section.

1.3 Factor analysis model

To find the solution for the factor analysis model we combine the latent random variable $z \sim N(\vec{0}, I)$ and the observed variable $x = \mu + \Lambda z + \varepsilon$, $\varepsilon \sim N(\vec{0}, \Psi)$ to

$$\begin{pmatrix} z \\ x \end{pmatrix} \sim N(\mu_{zx}, \Sigma).$$

Then $\mu_{zx} = E\begin{bmatrix} z \\ x \end{bmatrix} = \begin{bmatrix} Ez \\ Ex \end{bmatrix} = \begin{bmatrix} \vec{0} \\ \mu \end{bmatrix}$ is a $(d+n)$-dimensional vector and covariance matrix

$$\Sigma = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix},$$

where

$$\begin{aligned}\Sigma_{11} &= E[(z-Ez)(z-Ez)^T] = \mathrm{Cov}(z) = I,\\ \Sigma_{12} &= E[(z-Ez)(x-Ex)^T] = \Lambda^T,\\ \Sigma_{21} &= E[(x-Ex)(z-Ez)^T] = \Lambda,\\ \Sigma_{22} &= E(x-Ex)(x-Ex)^T = \Lambda\Lambda^T + \Psi.\end{aligned}$$

As an example, we show how to prove the formula for Σ_{21}:

$$\begin{aligned}\Sigma_{21} = E[(x-Ex)(z-Ez)^T] = E(\mu + \Lambda z + \varepsilon - \mu)\cdot z^T) =\\ = E\left[\Lambda z z^T\right] + E[\varepsilon z^T] = \Lambda E[zz^T] = \Lambda.\end{aligned}$$

In other words,

$$\begin{pmatrix} z \\ x \end{pmatrix} \sim N\left(\begin{bmatrix} \vec{0} \\ \mu \end{bmatrix}, \begin{bmatrix} I & \Lambda^T \\ \Lambda & \Lambda\Lambda^T + \Psi \end{bmatrix}\right),$$

and

$$x \sim N(\mu, \Lambda\Lambda^T + \Psi). \tag{3}$$

To find the parameters for this model we should maximize the log-likelihood given the training set:

$$\begin{aligned} l(\Lambda,\mu,\Psi) = \sum_{i=1}^m \ln p(x^{(i)};\Lambda,\mu,\Psi) =\\ = \sum_{i=1}^m \frac{1}{(2\pi)^{n/2}\cdot|\Lambda\Lambda^T+\Psi|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T(\Lambda\Lambda^T+\Psi)^{-1}(x-\mu)\right).\end{aligned}$$

The standard approach (finding derivatives and setting them to zero) does not work in this case: it is impossible to solve analytically the obtained system of equations.

1.4 EM steps for factor analysis

In order to find the maximum of the likelihood function we apply the EM algorithm:

- **E-step**. As before
$$Q_i(z^{(i)}) = p(z^{(i)}\,|\,x^{(i)};\Lambda,\mu,\Psi),$$
where
$$z^{(i)}\,|\,x^{(i)} \sim N(\mu_{z^{(i)}\,|\,x^{(i)}}, \Sigma_{z^{(i)}\,|\,x^{(i)}}).$$
To calculate $\mu_{z^{(i)}\,|\,x^{(i)}}$ and $\Sigma_{z^{(i)}\,|\,x^{(i)}}$ we use the formulas (1) and (2):
$$\mu_{z^{(i)}\,|\,x^{(i)}} = \Lambda^T(\Lambda\Lambda^T+\Psi)^{-1}(x^{(i)}-\mu) \tag{4}$$
and
$$\Sigma_{z^{(i)}\,|\,x^{(i)}} = I - \Lambda^T(\Lambda\Lambda^T+\Psi)^{-1}\Lambda. \tag{5}$$

- **M-step**. To maximize the log-likelihood we maximize the lower bound for the log-likelihood
$$\Lambda,\mu,\Psi = \arg\max_{\Lambda,\mu,\Psi} \sum_{i=1}^m \int\limits_{z^{(i)}} Q_i(z^{(i)}) \ln \frac{p(x^{(i)},z^{(i)};\Lambda,\mu,\Psi)}{Q_i(z^{(i)})} dz^{(i)}.$$

Notice that we have replaced sum by integration because $z^{(i)}$ is a continuous random variable now. By definition of the expected value, we have

$$\int\limits_{z^{(i)}} Q_i(z^{(i)}) \ln \frac{p(x^{(i)}, z^{(i)}; \Lambda, \mu, \Psi)}{Q_i(z^{(i)})} dz^{(i)} = E_{z^{(i)} \sim Q_i} \left[\ln \frac{p(x^{(i)}, z^{(i)}; \Lambda, \mu, \Psi)}{Q_i(z^{(i)})} \right] =$$
$$= E_{z^{(i)} \sim Q_i} \left[\ln p(x^{(i)} \mid z^{(i)}; \Lambda, \mu, \Psi) \right] + E_{z^{(i)} \sim Q_i} \left[\ln \frac{p(z^{(i)})}{Q_i(z^{(i)})} \right].$$

The second term in the last expression does not depend on the parameters ($Q_i(z^{(i)})$ is a distribution that is fixed on the E-step) which means that our goal is to maximize

$$\sum_{i=1}^{m} E_{z^{(i)} \sim Q_i} \left[\ln p(x^{(i)} \mid z^{(i)}; \Lambda, \mu, \Psi) \right] =$$
$$= \sum_{i=1}^{m} E \left[\ln \frac{1}{(2\pi)^{n/2} |\Psi|^{1/2}} \exp \left(-\frac{1}{2} (x^{(i)} - \mu - \Lambda z^{(i)})^T \Psi^{-1} (x^{(i)} - \mu - \Lambda z^{(i)}) \right) \right] =$$
$$= \sum_{i=1}^{m} E \left[-\frac{n}{2} \ln(2\pi) - \frac{1}{2} \ln |\Psi| - \frac{1}{2} (x^{(i)} - \mu - \Lambda z^{(i)})^T \Psi^{-1} (x^{(i)} - \mu - \Lambda z^{(i)}) \right]$$

(by the equation (3)). As an example, we show how to maximize with respect to Λ, in this case the first two terms are constants and the third term can be replaced by trace because Ψ is a diagonal matrix:

$$\nabla_\Lambda \sum_{i=1}^{m} E \left[\frac{1}{2} (x^{(i)} - \mu - \Lambda z^{(i)})^T \Psi^{-1} (x^{(i)} - \mu - \Lambda z^{(i)}) \right] =$$
$$= \sum_{i=1}^{m} \nabla_\Lambda E \left[-\operatorname{tr} \frac{1}{2} {z^{(i)}}^T \Lambda^T \Psi^{-1} \Lambda z^{(i)} + \operatorname{tr} {z^{(i)}}^T \Lambda^T \Psi^{-1} (x^{(i)} - \mu) \right] =$$
$$= \sum_{i=1}^{m} \nabla_\Lambda E \left[-\operatorname{tr} \frac{1}{2} \Lambda^T \Psi^{-1} \Lambda z^{(i)} {z^{(i)}}^T + \operatorname{tr} \Lambda^T \Psi^{-1} (x^{(i)} - \mu) {z^{(i)}}^T \right] =$$
$$= \sum_{i=1}^{m} E \left[-\Psi^{-1} \Lambda z^{(i)} {z^{(i)}}^T + \Psi^{-1} (x^{(i)} - \mu) {z^{(i)}}^T \right].$$

In these transformations we used the facts

$$\nabla_A \operatorname{tr} ABA^T C = CAB + C^T AB,$$
$$\operatorname{tr} AB = \operatorname{tr} BA.$$

By solving an equation $\nabla_\Lambda = 0$ we obtain

$$\sum_{i=1}^{m} \Lambda E \left[z^{(i)} {z^{(i)}}^T \right] = \sum_{i=1}^{m} (x^{(i)} - \mu) E \left[{z^{(i)}}^T \right]$$
$$\Rightarrow \Lambda = \left(\sum_{i=1}^{m} (x^{(i)} - \mu) E \left[{z^{(i)}}^T \right] \right) \left(\sum_{i=1}^{m} E \left[z^{(i)} {z^{(i)}}^T \right] \right)^{-1}.$$

But on the E-step we calculated (formulas (4) and (5))

$$E \left[{z^{(i)}}^T \right] = \mu_{z^{(i)} \mid x^{(i)}}$$

and for any random variable $z \sim N(\mu, \Sigma)$

$$\Sigma = E\left[zz^T\right] - (Ez)(Ez)^T \Rightarrow E\left[zz^T\right] = \Sigma + (Ez)(Ez)^T,$$

which implies

$$E\left[z^{(i)}{z^{(i)}}^T\right] = \Sigma_{z^{(i)}\,|\,x^{(i)}} + \mu_{z^{(i)}\,|\,x^{(i)}}\mu^T_{z^{(i)}\,|\,x^{(i)}}.$$

We leave the calculation of μ and Ψ as an exercise. Final formulas for all parameters on the M-step become

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)},$$

$$\Lambda = \left(\sum_{i=1}^{m}(x^{(i)} - \mu)\mu^T_{z^{(i)}\,|\,x^{(i)}}\right)\left(\sum_{i=1}^{m}\Sigma_{z^{(i)}\,|\,x^{(i)}} + \mu_{z^{(i)}\,|\,x^{(i)}}\mu^T_{z^{(i)}\,|\,x^{(i)}}\right)^{-1},$$

$$\Psi = \operatorname{diag}\frac{1}{m}\sum_{i=1}^{m}\left(x^{(i)}{x^{(i)}}^T - x^{(i)}\mu^T_{z^{(i)}\,|\,x^{(i)}}\Lambda^T - \Lambda\mu_{z^{(i)}\,|\,x^{(i)}}{x^{(i)}}^T + \right.$$
$$\left. +\Lambda\left(\mu_{z^{(i)}\,|\,x^{(i)}}\mu^T_{z^{(i)}\,|\,x^{(i)}} + \Sigma_{z^{(i)}\,|\,x^{(i)}}\right)\Lambda^T\right).$$

Exercise. Get an expressions for μ and Ψ by analogy with Λ.

1.5 Python implementation

Import necessary libraries:

```
In [1]: import numpy as np
        import pandas as pd
        import random
        import math
        import sklearn.datasets as ds
        import matplotlib.pyplot as plt
```

Factor analysis algorithm is implemented in the `scikit-learn` package:

```
In [2]: from sklearn.decomposition import FactorAnalysis
```

To illustrate how you can train this algorithm we use `boston` data from the `scikit-learn`.

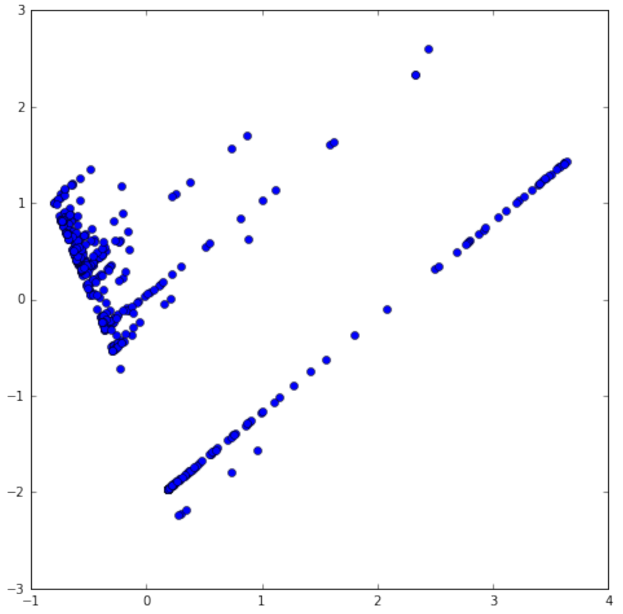
```
In [3]: boston = ds.load_boston()
        X = boston.data
        X.shape

Out[3]: (506, 13)
```

Fit the model:

```
In [4]: model = FactorAnalysis(n_components=2, random_state=324)
        model.fit(X)
        X_reduced = model.transform(X)
```

and visualize the results:

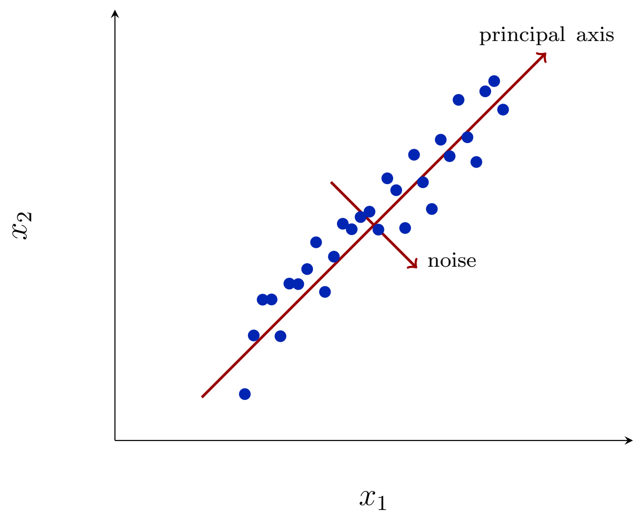
```
In [5]: fig = plt.figure(figsize=(8, 8))
        ax = fig.add_subplot(1, 1, 1)
        ax.plot(X_reduced[:,0], X_reduced[:,1], 'o')
        plt.show()
```



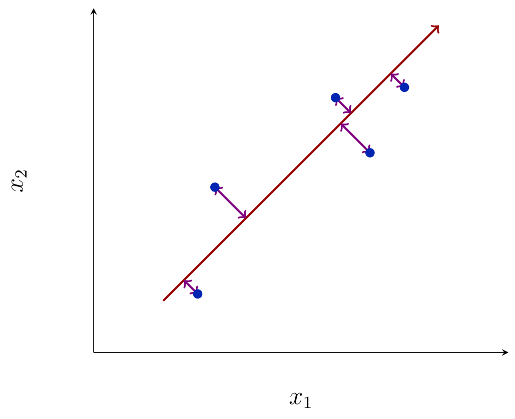
2 Principal Component Analysis

Compared to the factor analysis, the principal component analysis is not probabilistic algorithm. The main purpose of this algorithm is dimensionality reduction (though there are a lot of other applications like data vizualization, data compression, anomaly detection and distance calculation).

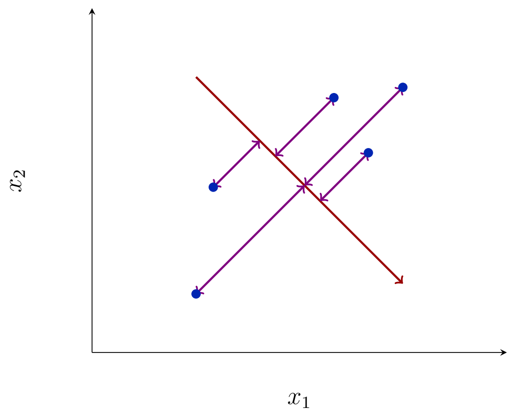
Assuming that our given data are n-dimensional: $\{x^{(1)}, \dots, x^{(m)}\}$, $x^{(i)} \in \mathbb{R}^n$, our goal is to reduce these data to k-dimensional data ($k < n$). The intuition of such reduction is the following:



On this picture 2-dimensional dataset could be reduced to the 1-dimensional. Consider the simple dataset that can give the idea how to complete this procedure:



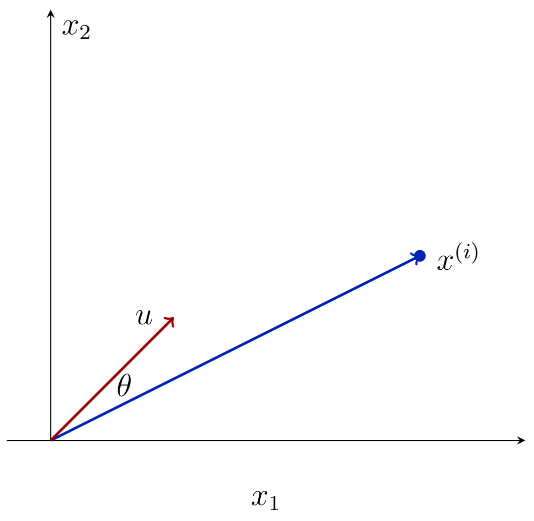
Good principal axis



Bad principal axis

We could observe that the best principle axis has the minimal sum of distances from the training data points. But a minimising the distance between point $x^{(i)}$ to the principle axis is equivalent to minimizing the angle between radius vector of $x^{(i)}$ and the principal axis. Let u be unit directional vector of the principal axis, then the cosine of angle between radius vector $x^{(i)}$ and u can be calculated as

$$\cos\theta = \frac{{x^{(i)}}^T \cdot u}{||x^{(i)}||},\ ||u|| = 1.$$



Our main goal is to find u with the following condition:

$$u = \arg\max_{u:||u||=1} \frac{1}{m}\sum_{i=1}^{m}({x^{(i)}}^T u)^2 = \arg\max_{u:||u||=1} \frac{1}{m}\sum_{i=1}^{m}(u^T x^{(i)})({x^{(i)}}^T u) =$$

$$= \arg\max_{u:||u||=1} u^T\left[\frac{1}{m}\sum_{i=1}^{m} x^{(i)}{x^{(i)}}^T\right]u.$$

Denote

$$\Sigma = \sum_{i=1}^{m} x^{(i)}{x^{(i)}}^T = X^T X \in \mathbb{R}^{n\times n},$$

then we should solve the optimization problem

$$\max_u u^T\Sigma u, \text{ given } u^T u = 1.$$

The Lagrangian

$$L(u,\lambda) = u^T\Sigma u - \lambda(u^T u - 1)$$

and it derivative with respect to u gives

$$\nabla_u L = \Sigma u - \lambda u = 0 \Leftrightarrow \Sigma u = \lambda u.$$

It implies that u is the principal eigenvector of $\Sigma = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}{x^{(i)}}^T$.

More generally, if we want k-dimensional subspace, choose $u_1,\ldots,u_k$ to be k top eigenvectors of Σ (corresponding to k highest eigenvalues). Then for each training point $x^{(i)} \in \mathbb{R}^n$ the new representation will be

$$z^{(i)} = \begin{bmatrix} u_1^T x^{(i)} \\ u_2^T x^{(i)} \\ \vdots \\ u_k^T x^{(i)} \end{bmatrix} \in \mathbb{R}^k.$$

One of the most efficient way to find the eigenvectors of the matrix Σ is to use **singular value decomposition (SVD)** for the design matrix $X \in \mathbb{R}^{m\times n}$:

$$X = UDV^T,$$

where $U \in \mathbb{R}^{m\times m}$, $V \in \mathbb{R}^{n\times n}$ and $D \in \mathbb{R}^{m\times n}$ is a diagonal matrix:

$$D = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & \sigma_{n-1} & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & \sigma_n & 0 & \cdots & 0 \end{bmatrix}$$

(matrix D has this form if $n > m$, otherwise, we have rows of zeros below the last nonzero row). Additionally, columns of the matrix U are eigenvectors of the matrix XX^T and columns of the matrix V are eigenvectors of the matrix X^TX. In most cases, finding SVD for the matrix X is very fast that allows to find top k eigenvectors of the matrix $\Sigma = X^TX$ very effectively. Another interesting fact is that for many applications a lot of singular values become zero that gives us natural dimensionality reduction (we could take number of nonzero eigenvalues as a dimensionality k of the reduced space). Finally, we formulate the PCA algorithm (notice that we have some preprocessing step in this algorithm, which is very important to implement before we calculate SVD for the matrix X).

Algorithm 1 PCA algorithm

```
1: Define the dimensionality $k$ for the reduced space
2: **Zero out mean**. Calculate
       $$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$
   and replace $x^{(i)}$ with $x^{(i)} - \mu$
3: **Normalization to unit variance**. Calculate
       $$\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}\left(x_j^{(i)}\right)^2$$
   and replace $x_j^{(i)}$ with $\frac{x_j^{(i)}}{\sigma_j}$ for all $j \in \{1,\dots,n\}$
4: Find SVD for the design matrix $X$:
       $$X = UDV^T$$
5: Set $V = [k$ left columns of the matrix $V]$
6: Calculate the matrix product
       $$Z = XV$$
7: **return** $Z$ (new design matrix of size $m \times k$)
```

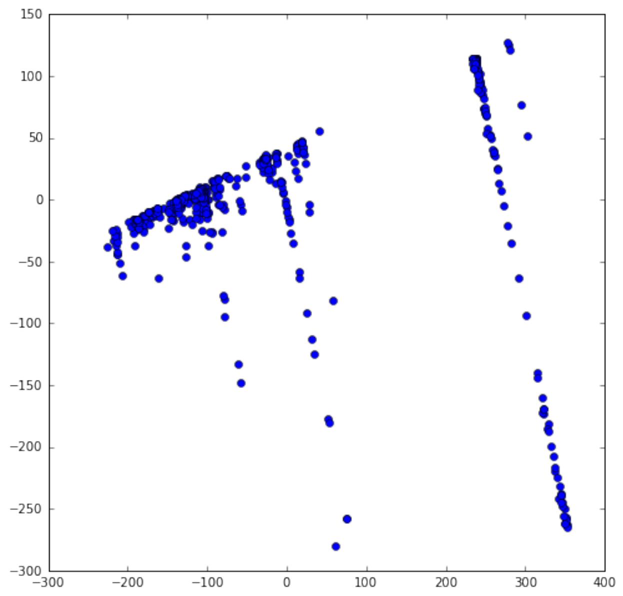
To compare different unsupervised algorithm we discussed in the lectures we combine them in the following table

	Probabilistic	**Not probabilistic**
Subspace	Factor analysis	PCA
Clustering	Mixture of Gaussians	K-means

2.1 Python implementation

The results of PCA algorithm is very close to the results obtained from the Factor Analysis algorithm:

```
In [6]: from sklearn.decomposition import PCA

In [7]: model = PCA(n_components=2)
        model.fit(X)
        X_reduced = model.transform(X)

In [8]: fig = plt.figure(figsize=(8, 8))
        ax = fig.add_subplot(1, 1, 1)
        ax.plot(X_reduced[:,0], X_reduced[:,1], 'o')
        plt.show()
```



3 Latent Semantic Indexing (LSI)

Latent Semantic Indexing (LSI) is the Principal Component Analysis algorithm applied to the text data. The main approach to the text machine learning problem is to construct binary feature vector where each feature represents the presence of some specific word:

$$x^{(i)} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ \vdots \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \begin{array}{c} a \\ as \\ advert \\ and \\ \vdots \\ claim \\ \vdots \\ light \\ \vdots \\ zip \end{array}$$

The feature vector could be very high dimensional, for example, $x^{(i)} \in \mathbb{R}^{10000}$, then in the Principal Component Analysis algorithm we should find the eigenvalues of the matrix $\Sigma = X^T X \in \mathbb{R}^{10000\times 10000}$, which is extremely large matrix. Notice that when we apply PCA to such kind of data usually we skip preprocessing steps (zero our mean and normalization to unit variance).

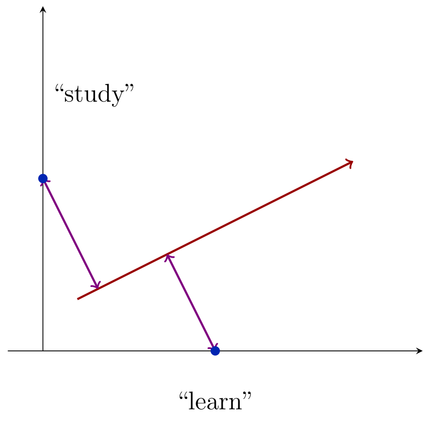
We can use SVD and find the principal components for our text data. Notice that the main goal of LSI is not a dimensionality reduction but rather calculating the similarity between documents. One general approach to measuring the similarity is to calculate cosine distance: given two documents $x^{(i)}$ and $x^{(j)}$ the cosine distance is determined as

$$\text{similarity}(x^{(i)}, x^{(j)}) = \cos\theta = \frac{{x^{(i)}}^T \cdot x^{(j)}}{||x^{(i)}|| \cdot ||x^{(j)}||},$$

where θ is the angle between two documents (angle between radius vectors of $x^{(i)}$ and $x^{(j)}$). The numerator of this similarity is

$${x^{(i)}}^T \cdot x^{(j)} = \sum_{l=1}^{n} x_l^{(i)} \cdot x_l^{(j)} = \sum_{l=1}^{n} \mathbb{1}\{\text{documents } i \text{ and } j \text{ both contain word } l\}.$$

One of the disadvantage of this approach is that if we have different words of the same meaning, for example, if one documents has a word “study” and the second document contains a word “learn”, then the similarity will be zero. When we apply PCA to our text data and find the principal components we would have something like



After the projection on the principal axes similar documents become closer to each other.

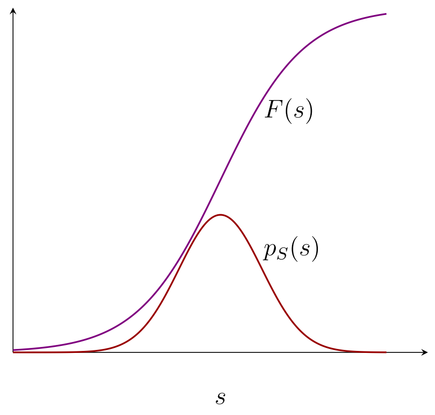
4 Independent Component Analysis (ICA)

For the given random variable S with the probability density function $p_S(s)$, the **cumulative distribution function (cdf)** is defined by

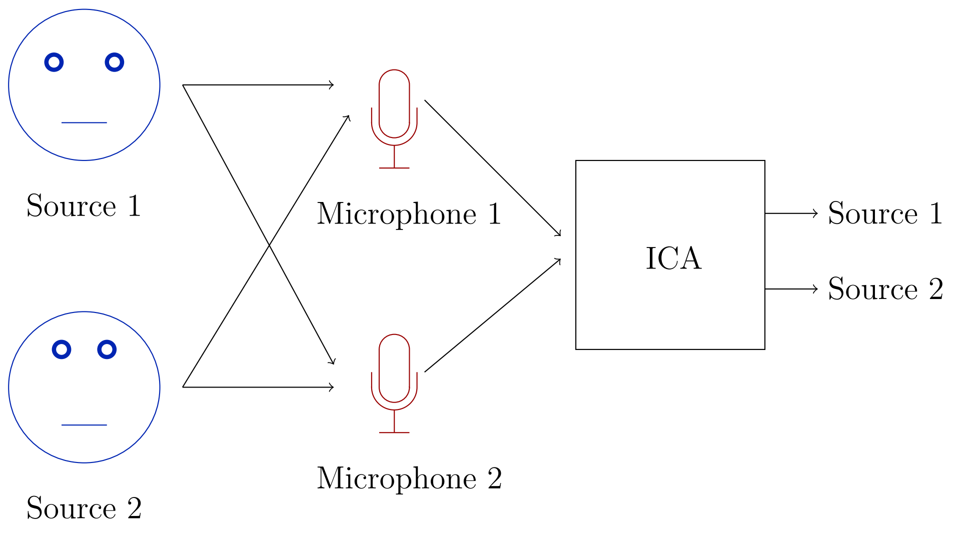
$$F(s) = P(S \leqslant s) = \int_{-\infty}^{s} p_S(t)dt.$$

From this definition the obvious connection between pdf and cdf is

$$p_S(s) = F'(s). \tag{6}$$



Consider the following problem: we use n microphones to record n speakers placed in the same room, the resulted signal at time i is denoted by $x^{(i)} \in \mathbb{R}^n$, where $x_j^{(i)}$ is a signal on the microphone j at time i. Also denote by $s_j^{(i)}$ a signal from speaker j at time i, $s^{(i)} \in \mathbb{R}^n$ (originally, $s^{(i)}$ are not known).



We assume that

$$x^{(i)} = As^{(i)},$$

where A is an unknown square matrix (**mixing matrix**), i.e.

$$x_j^{(i)} = \sum_k A_{jk} s_k^{(i)}.$$

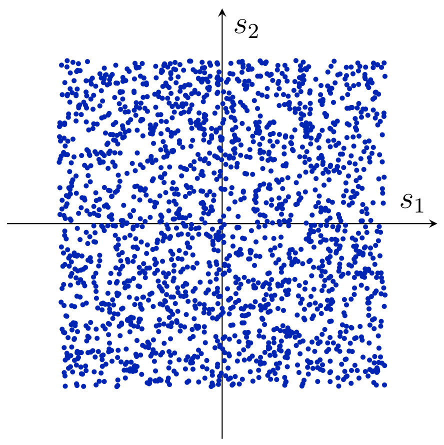
Our goal is to find **unmixing matrix** $W = A^{-1}$:

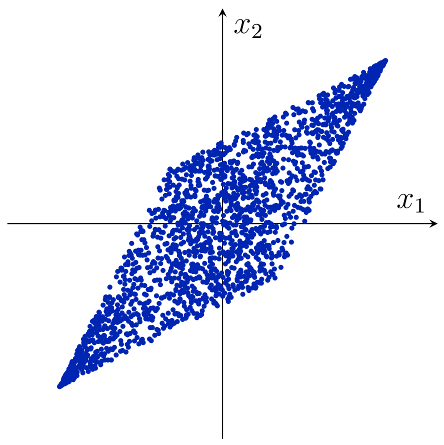
$$W = \begin{bmatrix} \longleftarrow w_1^T \longrightarrow \\ \vdots \\ \longleftarrow w_n^T \longrightarrow \end{bmatrix},$$

so that $s^{(i)} = Wx^{(i)}$.

We assume that

$$s_j^{(i)} \sim \text{Uniform } [-1, 1]$$





Analysing the problem can give an idea about ambiguities we have obtained. The first ambiguity is related to the fact that if we shuffle speakers then the output signal does not change. The second ambiguity is related to the signal sign: we cannot define if the signal is positive or negative. It turns out that if the original sources s are not Gaussian, then we do not have other ambiguities and we will be able to recover the matrix W. Otherwise, if the original signals s are Gaussians, we will have another ambiguity that is related to the invariance of Gaussian distribution to different rotations. In fact, in this situation we will have an arbitrary rotational component that does not allow us to recover the original sources.

Exercise. Show that if $s^{(i)} \sim N(0, I)$, then it is impossible to recover the original sources using ICA.

Assuming that we know the density function for $s \in \mathbb{R}^n$ and $s = Wx \Leftrightarrow x = W^{-1}s = As$, the density function for x is defined as

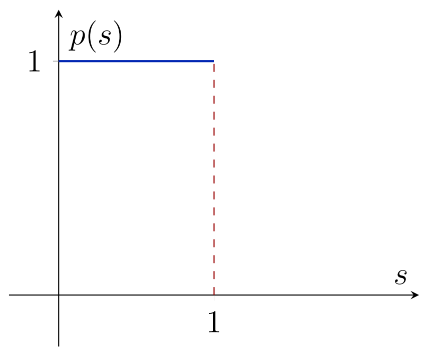
$$p_X(x) = p_S(Wx) \cdot |W|.$$

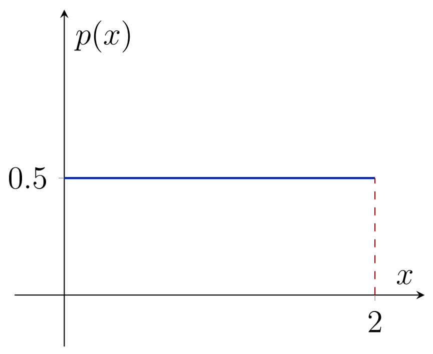
Example. Consider the uniform random variable $s \sim \text{Uniform}\,[0,1]$ with density

$$p_S(s) = \mathbb{1}\{0 \leqslant s \leqslant 1\}$$

and the random variable $x = 2s$. In this example, $A = 2$, $W = \dfrac{1}{2}$, then

$$p_X(x) = \mathbb{1}\{0 \leqslant x \leqslant 2\} \cdot \frac{1}{2}.$$





Finally, we can formulate the Independent Component Analysis algorithm. For the independent original sources we have

$$p(s) = \prod_{j=1}^{n} p_S(s_j),$$

then

$$p(x) = \left[\prod_{j=1}^{n} p_S(w_j^T x)\right] \cdot |W|,$$

where $W = A^{-1} = \begin{bmatrix} \longleftarrow w_1^T \longrightarrow \\ \vdots \\ \longleftarrow w_n^T \longrightarrow \end{bmatrix}$ and $s_j = w_j^T x$.

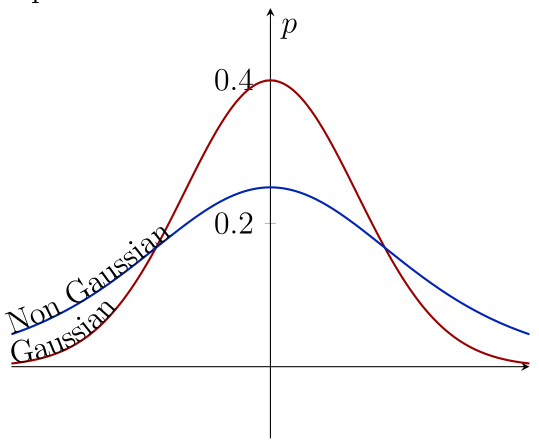
When we choose $p_S(s_j)$ we should make sure that it is not Gaussian. One of the option is to choose cdf first and after use the formula (6). It is convenient to choose

$$F(s) = \frac{1}{1+e^{-s}},$$

then

$$p(s) = F'(s) = \frac{e^{-s}}{(1+e^{-s})^2}$$

(compared to the Gaussian distribution the density of this distribution has fatter tails):



Another option is to choose Laplace distribution $p(s) = \frac{1}{2}e^{-|s|}$.

Given the set of output signals $\{x^{(1)}, \dots, x^{(m)}\}$ we write down the log-likelihood for our parameters:

$$l(W) = \sum_{i=1}^{m} \ln \left(\left[\prod_{j=1}^{n} p_S(w_j^T x^{(i)}) \right] \cdot |W| \right) = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} \ln p_S(w_j^T x^{(i)}) + \ln |W| \right).$$

By taking derivatives with respect to W and using the fact

$$\nabla_W |W| = |W| \left(W^{-1}\right)^T$$

we obtain:

$$\nabla_W l(W) = \sum_{i=1}^{m} \left(\begin{bmatrix} 1 - \dfrac{2}{1+\exp(-w_1^T x^{(i)})} \\ \vdots \\ 1 - \dfrac{2}{1+\exp(-w_n^T x^{(i)})} \end{bmatrix} {x^{(i)}}^T + (W^T)^{-1} \right).$$

Using this gradient we can write down one step of the stochastic gradient descent for the output signal $x^{(i)}$

$$W := W + \alpha \cdot \left(\begin{bmatrix} 1 - \dfrac{2}{1+\exp(-w_1^T x^{(i)})} \\ \vdots \\ 1 - \dfrac{2}{1+\exp(-w_n^T x^{(i)})} \end{bmatrix} {x^{(i)}}^T + (W^T)^{-1} \right).$$

After we find W we can recover unknown original signals as $s^{(i)} = W x^{(i)}$.

The final remark is there are a lot of applications of the ICA algorithm:

- EEG cap: split signals from different parts of the brain (for example, split brain signal to heart-beat signal, eyeblink signal and so on);
- independent component of images.