Numerical learning library 0.17 algorithm documentation

Ludovic Sibille

November 6, 2012

Contents

1 Introduction

This document will describe the main algorithms used in the Numerical Learning Library 0.17, how they are implemented and the assumptions made. It is meant to be as self-contained as possible, although some technical points requiring a great length of details will be referenced.

2 Bayesian networks

2.1 Overview

Bayesian networks are directed acyclic graphs whose nodes represent random variables which may be observable quantities, latent variables, unknown parameters. Edges represent conditional dependencies; nodes which are not connected represent variables which are conditionally independent of each other. Each node is associated with a probability function that takes as input a particular set of values for the node's parent variables and gives the probability of the variable represented by the node. For example in Figure 1, we model the joint probability $p(X, Y_1, Y_2) = p(Y_1)p(Y_2)p(X|Y_1, Y_2)$.

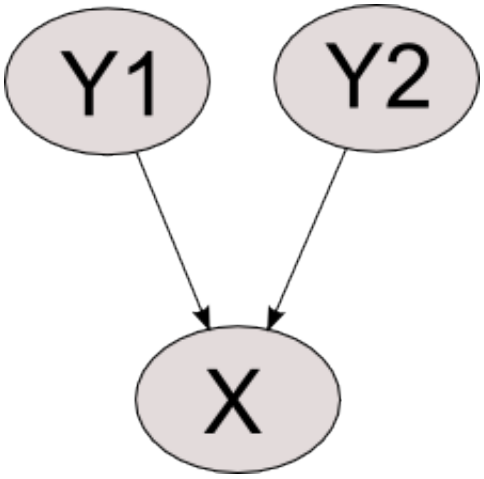

Figure 1: Bayesian network with random variables Y_1, Y_2 and X. Y_1 and Y_2 are independent, while X depends on Y_1 and Y_2.

2.2 Conditional Probability Distribution

Conditional Probability Distribution, or CPD, are modelling a conditional probability p, of a random variable X given the parents Y as following: $p(X|Y)$. Y can represent several random variables. Each CPD has a scope composed of the variables it contains (i.e., (X, Y)). Inference and learning algorithms for bayesian network will require the CPD to implement some basic operations such as:

- marginalization: given $p(X, Y)$, compute $p(X)$,
- conditioning: given $p(X, Y)$ and evidence $Y = y$, compute $p(X, Y = y)$,
- normalization: given $p(X)$, update the CPD so that it represents a probability distribution,

- multiplication: given $p(X)$ and $p(Y)$ combine the CPDs into a joint distribution $p(X, Y)$

To simplify the implementation, the scope will be identified by an integer with the following properties:

- the parents' identifier will be higher than the children's identifier,
- the scope of the CPD will always be sorted according to the identifier in increasing order. Consequently Scope[0] will always be the children variables.

2.2.1 Table Factor [*PotentialTable*]

A table CPD is encoding probabilities for random discrete variables in a dense way. Given a random variable X with N Parents $Par_i(X)$, the table will have a size of $Size = \#X \prod_{n=1}^{N} \#Par_n(X)$. The table will consider the scope as a new basis so that $index(X_1 = e_1, ..., X_n = e_n) = \sum_{j=1}^{N} e_j \prod_{i=1}^{j-1} \#X_i$. For example, see the table CPD of Figure 2.2.1.

$p(X_2, X_1)$	index
(0, 0)	0
(0, 1)	1
(1, 0)	2
(1, 1)	3
(2, 0)	4
(2, 1)	5

Table 1: Table CPD encoding the conditional distribution $p(X_1|X_2)$ or joint distribution $p(X_1, X_2)$ of two discrete random variables X_1 and X_2 of cardinality 2 and 3 respectively.

- marginalization: given $p(X, Y)$ compute $p(X) = \sum_{y_i \in Y} p(X)p(X|Y = y_i)$,
- conditioning: $p(X, Y = y)$ simply eliminates the rows that do not agree with y and drop Y from the scope,
- normalization: normalize the table so that $\sum_i p(X = i) = 1$,
- scope extension: given $p(X)$, extend the CPD's scope to (X, Y) such that $p(X = i|Y_j) = p(X = i), \forall j$
- multiplication: given $p(X)$ and $p(Y)$ merge the CPD into a single CPD $p(X, Y)$. First extend $p(X)$ scope to Y (and similarly for $p(Y)$), then multiply the tables element by element.

2.2.2 Gaussian Moment Factor [*PotentialGaussianMoment*]

Parametrization

Consider a Gaussian with mean μ and covariance matrix Σ defined on some domain x of size n.

$$\phi(x; \alpha, \mu, \Sigma) = \alpha e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)}$$

If $\alpha = (2\pi)^{-n0.5}\Sigma^{-0.5}$, the potential is exactly a gaussian PDF and will be called "normalized". Note that in most published papers, α will often be wrongly computed and this is why the computations are extensively detailed.

Block diagonalization

Consider a general partitioned matrix $M = \begin{bmatrix} E & F \\ G & H \end{bmatrix}$ To zero out the upper right and lower left part of the matrix, we can do the following:

$$\begin{bmatrix} I & -FH^{-1} \\ 0 & I \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} \begin{bmatrix} I & 0 \\ -H^{-1}G & I \end{bmatrix} = \begin{bmatrix} E - FH^{-1}G & 0 \\ 0 & H \end{bmatrix}$$

We define the Schur complement of matrix M with respect to H, $M/H = E - FH^{-1}G$. Since $(XYZ)^{-1} = Z^{-1}Y^{-1}Z^{-1} = W^{-1}$ and so $Y^{-1} = ZW^{-1}X$. Consequently we have:

$$\begin{aligned} \begin{bmatrix} E & F \\ G & H \end{bmatrix}^{-1} &= \begin{bmatrix} I & 0 \\ -H^{-1}G & I \end{bmatrix} \begin{bmatrix} (M/H)^{-1} & 0 \\ 0 & H^{-1} \end{bmatrix} \begin{bmatrix} I & -FH^{-1} \\ 0 & I \end{bmatrix} \\ &= \begin{bmatrix} (M/H)^{-1} & -(M/H)^{-1}FH^{-1} \\ -H^{-1}G(M/H)^{-1} & H^{-1} + H^{-1}G(M/H)^{-1}FH^{-1} \end{bmatrix} \end{aligned} \tag{1}$$

Note that we can also decompose the matrix as followed:

$$\begin{bmatrix} I & 0 \\ -GE^{-1} & I \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} \begin{bmatrix} I & -E^{-1}F \\ 0 & I \end{bmatrix} = \begin{bmatrix} E & 0 \\ 0 & -GE^{-1}F - H \end{bmatrix}$$

Which similarly to (1) lead to the following:

$$\begin{bmatrix} E & F \\ G & H \end{bmatrix}^{-1} = \begin{bmatrix} E^{-1} + E^{-1}F(M/E)^{-1}GE^{-1} & -E^{-1}F(M/E)^{-1} \\ -(M/E)^{-1}GE^{-1} & (M/E)^{-1} \end{bmatrix} \tag{2}$$

We can note that $|M| = |M/H||H|$.

Marginalization

Consider a Gaussian on some domain (x_1, x_2) of size (p, q) with mean $\mu = [\mu_1 \ \mu_2]^T$ and covariance $\Sigma = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}$. We have:

$$p(x_1, x_2) = \underbrace{\frac{1}{(2\pi)^{1/2*(p+q)}|\Sigma|^{1/2k}}}_{A} \exp \underbrace{\left\{ -\frac{1}{2} \begin{pmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \end{pmatrix}^T \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}^{-1} \begin{pmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \end{pmatrix} \right\}}_{B}$$

$$\begin{aligned} B &= \left\{ -\frac{1}{2} \begin{pmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \end{pmatrix}^T \begin{bmatrix} I & 0 \\ -\Sigma_{22}^{-1}\Sigma_{21} & I \end{bmatrix} \begin{bmatrix} (\Sigma/\Sigma_{22})^{-1} & 0 \\ 0 & \Sigma_{22}^{-1} \end{bmatrix} \begin{bmatrix} I & -\Sigma_{12}\Sigma_{22}^{-1} \\ 0 & I \end{bmatrix} \begin{pmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \end{pmatrix} \right\} \\ &= -\frac{1}{2}(x_1 - \mu_1 - \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2))^T(\Sigma/\Sigma_{22})^{-1}(x_1 - \mu_1 - \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2)) \\ &\quad - \frac{1}{2}(x_2 - \mu_2)^T\Sigma_{22}^{-1}(x_2 - \mu_2) \\ A &= \frac{1}{(2\pi)^{0.5(p+q)}(|\Sigma/\Sigma_{22}||\Sigma_{22}|)^{0.5}} \\ &= \left(\frac{1}{(2\pi)^{0.5p}|\Sigma/\Sigma_{22}|^{0.5}}\right)\left(\frac{1}{(2\pi)^{0.5q}|\Sigma_{22}|^{0.5}}\right) \end{aligned}$$

From the relation $p(x_1, x_2) = p(x_1|x_2)p(x_2)$ and the previous equations, we find that:

$$p(x_2) = \frac{1}{(2\pi)^{0.5q}|\Sigma_{22}|^{0.5}} \exp\left(-\frac{1}{2}(x_2 - \mu_2)^T \Sigma_{22}^{-1}(x_2 - \mu_2)\right)$$

$$p(x_1|x_2) = \frac{1}{(2\pi)^{0.5p}|\Sigma/\Sigma_{22}|^{0.5}} *$$
$$\exp\left(-\frac{1}{2}(x_1 - \mu_1 - \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2))^T (\Sigma/\Sigma_{22})^{-1}(x_1 - \mu_1 - \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2))\right)$$

Note that we extend the previous results for non-normalized gaussian distribution which gives us the following results:

$$\begin{aligned}\phi(x_2) &= \int \phi(x_1, x_2)dx_1 \\ &= \phi(x_2; \alpha, \mu_2, \Sigma_2) \int p(x_1|x_2)dx_1 \\ &= \phi(x_2; \frac{\alpha}{\alpha_{1|2}}, \mu_2, \Sigma_2) \text{ where } \alpha_{1|2} = \frac{1}{(2\pi)^{0.5p}|\Sigma/\Sigma_{22}|^{0.5}}\end{aligned}$$

Conditioning
Suppose we are observing x_2 with value $\hat{x}_2$, then we have:

$$\begin{aligned}\phi(x_1, x_2 = \hat{x}_2) &= \alpha \exp(B) \\ &= \phi(x_1; \alpha\alpha_2, \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(\hat{x}_2 - \mu_2), \Sigma_{11} + \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21})\end{aligned}$$

Where $\alpha_2 = \exp\left(-\frac{1}{2}\hat{x}_2^T \Sigma_{22}^{-1}\hat{x}_2\right)$

2.2.3 Gaussian Canonical Factor [*PotentialGaussianCanonical*]

Parametrization
Unfortunately multiplication operations are difficult to realize in the moment form as it involves the inversion of the two potentials' covariance matrices which may not exist. To help with this, the computation will be done on another form, namely the canonical representation.

$$\phi(x; K, h, g) = \exp(g + h^T x - \frac{1}{2}x^T K x)$$

Moment to Canonical representation

$$\begin{aligned}\phi(x; \alpha, \mu, \Sigma) &= \alpha \exp(-\frac{1}{2}(x - \mu)^T \Sigma^{-1}(x - \mu)) \\ &= \exp(\log(\alpha) - \frac{1}{2}(x^T\Sigma^{-1}x - x^T\Sigma^{-1}\mu - \mu^T\Sigma^{-1}x + \mu^T\Sigma^{-1}\mu)) \\ &= \exp(\{\log(\alpha) - \frac{1}{2}\mu^T\Sigma^{-1}\mu\} + \{(\Sigma^{-1}\mu)^T x\} + \{x^T\Sigma^{-1}x\})\end{aligned}$$

We conclude that:

$$K = \Sigma^{-1} \tag{3}$$
$$h = \Sigma^{-1}\mu \tag{4}$$
$$g = \log(\alpha) - \frac{1}{2}\mu^T\Sigma^{-1}\mu \tag{5}$$

Canonical to Moment representation

$$\Sigma = K^{-1} \tag{6}$$
$$\mu = K^{-1}h \tag{7}$$

Additionally we have:

$$\begin{aligned} g &= \log(\alpha) - \frac{1}{2}(K^{-1}h)^T K^{-1}h \\ &= \log(\alpha) - \frac{1}{2}h^T K^{-1}h \\ \log(\alpha) &= g + \frac{1}{2}h^T K^{-1}h \\ \alpha &= \exp(g + \frac{1}{2}h^T K^{-1}h) \end{aligned} \tag{8}$$

Multiplication/Division
When multiplied, potentials are first extended to the other's domain by adding zeros. Multiplying/Dividing potentials in exponential forms is straightforward:

$$\phi(K_a, h_a, g_a) * \phi(K_b, h_b, g_b) = \phi(K_a + K_b, h_a + h_b, g_a + g_b) \tag{9}$$
$$\phi(K_a, h_a, g_a)/\phi(K_b, h_b, g_b) = \phi(K_a - K_b, h_a - h_b, g_a - g_b) \tag{10}$$

Marginalization
Consider the potential:

$$\phi(x_1, x_2) = \left[\begin{pmatrix} h_1 \\ h_2 \end{pmatrix}, \begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix}, g \right]$$

We want to find:

$$\begin{aligned} \phi(x_1; h', K', g') &= \int \phi(x_1, x_2) dx_2 \\ &= \int \phi(x_1; h', K', g) p(x_2|x_1) dx_1 \\ &= \phi(x_1; h', K', g) \int p(x_2|x_1) dx_1 \end{aligned}$$

By definition, we have:

$$\begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix}^{-1} = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}$$

Using (1) we find that

$$\boldsymbol{K' = K_{11} = (\Sigma/\Sigma_{22})^{-1} = K_{11} - K_{12}K_{22}^{-1}K_{21}}$$

Using (2) we have:

$$\begin{aligned} K_{22} &= (\Sigma/\Sigma_{11})^{-1} \\ K_{12} &= -\Sigma_{11}^{-1}\Sigma_{12}(\Sigma/\Sigma_{11})^{-1} \\ &= -\Sigma_{11}^{-1}\Sigma_{12}K_{22} \\ \text{meaning that } \Sigma_{12} &= -\Sigma_{11}K_{12}K_{22}^{-1} \end{aligned} \tag{11}$$

Using (7) we have:

$$\begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix} = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} \begin{pmatrix} h_1 \\ h_2 \end{pmatrix}$$

$$\mu_1 = \Sigma_{11} h_1 + \Sigma_{12} h_2$$

$$\text{By definition } h' = \Sigma_{11}^{-1} \mu_1$$

$$\text{Injecting } \mu_1 \; h' = \Sigma_{11}^{-1}(\Sigma_{11} h_1 + \Sigma_{12} h_2)$$

$$\text{Using (11) we find that } \boldsymbol{h' = h_1 - K_{12} K_{22}^{-1} h_2}$$

Finally, we compute the constant in the same way as we did for the moment case. We use α_2 as the normalization constant for $p(x_2)$:

$$\begin{aligned} \phi(x_1) &= \int \phi(x_1, x_2) dx_2 \\ &= \int \phi(x_1; h', K', g) p(x_2|x_1) dx_1 \\ &= \phi(x_1; h', K', g) \int p(x_2|x_1) dx_1 \\ &= \boldsymbol{\phi(x_1; h', K', g - (\log(\alpha_2) - \frac{1}{2} \mu_2^T \Sigma_{22}^{-1} \mu_2)} \end{aligned}$$

$$\text{With } \alpha_2 = \frac{1}{(2\pi)^{0.5q} |\Sigma_{22}|^{0.5}}$$

Conditioning
Suppose we are observing x_2 with value $\hat{x}_2$, then we have:

$$\begin{aligned} \phi(x_1, x_2 = \hat{x}_2) &= \exp\left(g + \begin{pmatrix} h_1 \\ h_2 \end{pmatrix}^T \begin{pmatrix} x_1 \\ \hat{x}_2 \end{pmatrix} - \frac{1}{2} \begin{pmatrix} x_1 \\ \hat{x}_2 \end{pmatrix}^T \begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix} \begin{pmatrix} x_1 \\ \hat{x}_2 \end{pmatrix} \right) \\ &= \exp\left(\left\{ g + h_2^T \hat{x}_2 - \frac{1}{2} \hat{x}_2^T K_{22} \hat{x}_2 \right\} + \left\{ x_1^T (h_1 - K_{12} \hat{x}_2) \right\} + \left\{ -\frac{1}{2} x_1^T K_{11} x_1 \right\} \right) \end{aligned}$$

2.2.4 Linear Gaussian Factor [*PotentialLinearGaussian*]

Let Y be a continuous set of nodes with continuous parents X. We says that Y has a linear Gaussian model if it can be described using parameters A, μ_0 and variance Σ_0 such that:

$$p(Y|X = x) = N(\mu_0 + Ax; \Sigma_0)$$

From the linear transformations of gaussians, we have the following results:

- If X is drawn by a gaussian $N(\mu, \Sigma)$ such that $y = Ax + b$
- Then $p(Y) = N(A\mu + b, A\Sigma A^T)$
- In fact,

$$p(X, Y) = N\left(\begin{bmatrix} \mu_x \\ \mu_0 + A\mu_x \end{bmatrix}, \begin{bmatrix} \Sigma_x & \Sigma_x A^T \\ A\Sigma_x & A\Sigma_x A^T \end{bmatrix}, \right)$$

In this model, the noise is accumulating (i.e., the noise of y depends on Σ_0 as well as the parents noise). If X is drawn by a gaussian $N(\mu_x, \Sigma_x)$ we have this joint distribution:

$$p(X, Y) = N\left(\begin{bmatrix} \mu_x \\ \mu_0 + A\mu_x \end{bmatrix}, \begin{bmatrix} \Sigma_x & \Sigma_x A^T \\ A\Sigma_x & A\Sigma_x A^T + \Sigma_0 \end{bmatrix}, \right)$$

Since the conditional probability may not be a PDF, the covariance matrix may not be invertible. Consequently, we directly convert to a canonical gaussian to avoid the covariance inversion:

$$\begin{aligned} p(y|x) &= N(\mu_0 + Ax, \Sigma_0) \\ &= \frac{1}{Z} \exp\left(-\frac{1}{2}(y - Ax - \mu_0)^T \Sigma_0^{-1} (y - Ax - \mu_0)\right) \\ &= \frac{1}{Z} \exp\left(-\frac{1}{2}\left\{ \begin{bmatrix} y & x \end{bmatrix} \begin{bmatrix} I & -A \end{bmatrix}^T \Sigma_0^{-1} \begin{bmatrix} I & -A \end{bmatrix} \begin{bmatrix} y & x \end{bmatrix}^T \right\} + A\right) \end{aligned}$$

$$\text{with } A = \mu_0^T \Sigma_0^{-1} \begin{bmatrix} I & -A \end{bmatrix} \begin{bmatrix} y & x \end{bmatrix} - \frac{1}{2}\mu_0^T \Sigma_0^{-1} \mu_0$$

We conclude that:

$$K = \begin{bmatrix} I & -A \end{bmatrix}^T \Sigma_0^{-1} \begin{bmatrix} I & -A \end{bmatrix} \tag{12}$$

$$h = \begin{bmatrix} I & -A \end{bmatrix}^T \Sigma_0^{-1} \mu_0 \tag{13}$$

$$g = \log(\frac{1}{Z}) - \frac{1}{2}\mu_0^T \Sigma_0^{-1} \mu_0 \tag{14}$$

2.2.5 Conditional Linear Gaussian Factor [*PotentialLinearGaussianConditional*]

We are building on top of the Linear Gaussian CPD (LG), where discrete parents of continuous variables are allowed in the model. We are facing two problems:

- if we marginalize a discrete variable of a CLG, the resulting PDF will be a mixture of gaussians of higher dimension than each of the LG it is composed of,
- we cannot represent CLG with a discrete variable having a gaussian parent

2.2.6 Mixture of truncated exponentials

2.2.7 Always Observed Nodes

If we can guaranty that some variables are always observed then we can easily combine any model in the bayesian network (neural nets, trees...) as we do not need to manipulate its PDF.

2.3 Inference

2.3.1 Naive algorithm

The naive algorithm is an exact inference algorithm. It simply constructs the full joint probability after incorporating the evidence to form the result of the inference query. Since it is extremely simple but generally inefficient (except for the Gaussian or Linear Gaussian CPD) it is only used for debugging purposes.

2.3.2 Variable elimination

2.4 Learning

2.4.1 Fully observed case

3 Other Algorithms

3.1 k-means clustering [*KMeans*]

k-means clustering is a method of cluster analysis which aims to partition N observations into K clusters c_k in which each observation belongs to the cluster with the nearest mean.

We want to determine:

$$\begin{aligned}\min_{c_k, r_{ik}} D &= \sum_{i=1}^{N}\sum_{k=1}^{K} r_{ik}\|x_i - c_k\|^2 \\ \frac{\partial}{\partial c_k} D &= 2\sum_{i=1}^{N} r_{ik}(x_i - c_k) \\ c_k &= \frac{\sum_{i=1}^{N} r_{ik} x_i}{\sum_{i=1}^{N} r_{ik}} \end{aligned} \tag{15}$$

With $r_{ik} = 1$ if the point i belongs to cluster k, 0 otherwise. The algorithm repeats these two steps until convergence:

- assign r_{ik} so that each point belongs to the closest cluster
- update the cluster position using (15)

It must be noted that the algorithm is very sensitive to the initial cluster position.

3.2 Hessian approximation FD [*HessianCalculatorForwardFiniteDifference*]

Using finite differences, we have:

$$\begin{aligned}\frac{\partial f(x)}{\partial x_i \partial x_j} &= \frac{\partial}{\partial x_j}\left(\frac{\partial f(x)}{x_i}\right) \\ &= \frac{\partial}{\partial x_j}\left(\frac{f(x + he_i) - f(x)}{h}\right) \\ &= \frac{1}{h}\left(\frac{f(x + he_i + he_j) - f(x + he_i)}{h} - \frac{f(x + he_j) - f(x)}{h}\right) \\ &= \frac{f(x + he_i + he_j) - f(x + he_i) - f(x + he_j) + f(x)}{h^2} \end{aligned} \tag{16}$$

with h close to 0 and e_k the basis vector for the parameter k. See [1] for other implementations and drawbacks. Note that the step size h must be sufficiently low for a good accuracy but it must be as well sufficiently high to avoid underflow problems; This is problem dependent.

4 Optimization algorithms

4.1 Newton's method [*OptimizerNewton*]

The Newton's method finds a **local extremum** (i.e., minimum or maximum) by constructing a sequence x_n from an initial guess x_0 that converges towards x_* such that the function $\nabla_f(x_n) = 0$. Consequently finding a global extremum is not guaranteed. The closer the initial guess is from the global extremum, the more likely it is to find the global extremum.

It is assuming the function f is twice differentiable and locally approximated well by a quadratic function. The function f is locally approximated by a Taylor series of degree two:

$$
\begin{aligned}
f(x_n + p) &= f(x_n) + p^T \nabla_f(x_n) + \frac{1}{2} p^T H_f(x_n) p + \mathcal{O}(p^3) \\
\frac{d}{dp} f(x_n) &= 0 \\
\frac{d}{dp} f(x_n) &= \nabla_f(x_n) + H_f(x_n) p \\
p &= H_f^{-1}(x_n) \nabla f(x_n) \\
\text{taking } p = x_{n+1} - x_n \text{ we obtain } x_{n+1} &= x_n - H_f^{-1}(x_n) \nabla f(x_n)
\end{aligned}
$$

Additionally, the iteration step is scaled by $\gamma \in]0..1]$ to help with the convergence, leading to the following update:

$$x_{n+1} = x_n - \gamma H_f^{-1}(x_n) \nabla f(x_n) \tag{17}$$

In the case H_f is not invertible, the hessian matrix is reconditioned such that $H_f = H_f + \lambda I$, with λ a small value. Note that since we are inverting the Hessian matrix as in (17), this algorithm is not suitable for functions with a high number of variables.

4.2 Gradient Descent [*OptimizerGradientDescent*]

The Newton's method is attempting to construct a sequence x_n from an initial guess x_0 that converges towards x_* such that the function $\nabla_f(x_n) = 0$. Consequently finding a global extremum is not guaranteed. The closer the initial guess is from the global minimum, the more likely it is to find the global minimum.

It is assuming the function f is differentiable. The parameters must be properly scaled (i.e., all the parameters are comparable and none of them is dominating the others). The update is simply:

$$x_{n+1} = x_n - \lambda \nabla f(x_n) \tag{18}$$

λ must be sufficiently small to avoid the bouncing effect and sufficiently by for fast convergence. λ is a single value if the parameters are scaled or can be a vector so that each parameter is scaled independently. The gradient can be normalized to have more comparable λ accross problems.

If the parameters are not properly scaled (e.g., narrow valley), convergence may be extremely slow. In this case a second order method is advised.

4.3 Hybrid Gradient Descent [*OptimizerGradientDescentLineSearch*]

One problem with the Gradient Descent is that the step size λ is constant (it might be a good step at the beginning of the optimization, but not so afterward) and needs to be chosen by hand with all the problems that implies. An alternative is to have a variable step size and its value being computed by a line search. Note that this approach requires more function evaluations. This leads to the following parameter update:

$$\begin{aligned} d_n &= \frac{\nabla f(x_n)}{\|\nabla f(x_n)\|} \\ \lambda_n &= lineSearch(x_n, d_n) \\ x_{n+1} &= x_n - \lambda_n d_n \end{aligned} \tag{19}$$

If the parameters are not properly scaled (e.g., narrow valley), convergence may be extremely slow. In this case a second order method is advised.

Very similarly, other optimizers are combined with a line search such as [*OptimizerNewtonLineSearch*]

4.4 Grid Search [*OptimizerGridSearch*]

Find the optimal parameters using an exhaustive search at discrete intervals. The search step for each parameter is defined by **param.next(currentVal)**. Grid search could be extremely expensive in time, but it is guaranteed that it will find the minimum value on this grid. If the evaluation function is quite fast, and the range search is small, this algorithm is suitable. It is often used for parameter selection for example for a classifier.

4.5 Harmony Search [*OptimizerHarmonySearch*]

Harmony search (HS) is a meta-heuristic optimization method imitating the music improvisation process where musicians improvise their instruments' pitches searching for a perfect state of harmony. The following inputs are required:

- hms: harmony memory rate. The memory size used to improvise new harmony from. Typical range is [10..100],
- hmrc: harmony considering rate. Indicates the probability of generating a value from the memory. Typical range is [0.7..0.99],
- par: pitch adjusting rate. Given a value generated from memory, this indicates the probability of modifying it. [0.1..0.4],
- bw: bandwith. The update size,
- f: the function to minimize.

Algorithm 1 HarmonySearch(f, hms, hmrc, par, bw)

```
memory ← InitializeRandomlyMemory(hms)
while stoppingCriteria do
  // generate each value of vector from random,
  // or taken from memory and slighly updated
  x ← empty
  for (i=0; n < dim; ++n) do
    if U(0,1) < hmrc then
      x[i] ← memory[U(1..hms)][i]
      if U(0,1) < par then
        x[i] ← x[i] ± U(0..1) * bw
      end if
    else
      x[i] ← random()
    end if
  end for
  // finally evaluate the solution and remove
  // from memory the worst one
  if f(x) < memory.worst() then
    memory.removeWorstAndAdd(x)
  end if
end while
return memory.best()
```

4.6 Genetic Algorithm [*OptimizerGeneticAlgorithm*]

The genetic algorithm (GA) optimizer uses a GA to perform the minimization. It is configured as following:

- mutation operator: the value is randomly updated within the range of the parameter,
- selection operator: elitist. The best solution is always selected and never mutated,
- crossover operator: vector 2-point split. Two Potential solutions are selected and split at two random positions. Some segments of the solutions are swaped,
- termination: defined by the number of epochs.

4.7 Powell's conjugate direction method [*powell*]

The method minimises the function by a bi-directional search along each search vector, in turn. The new position can then be expressed as a linear combination of the search vectors. The new displacement vector becomes a new search vector, and is added to the end of the search vector list. Meanwhile the search vector which contributed most to the new direction, i.e. the one which was most successful, is deleted from the search vector list. The algorithm iterates an arbitrary number of times until no significant improvement is made. The function needs not be differentiable. Implemented exactly as described in [2].

4.8 Bracketing [*minimumBracketing*]

Given a function f and two initial points a and b, find a triplet (a', b', c') such that $f(a') > f(b')$ and $f(c') > f(b')$. This is often to make sure that a function has at least a minima in this interval, especially for a line search algorithm. Implemented exactly as described in [2] using a golden section search.

4.9 Line Search [*lineMinimization*]

The line search is using [*minimumBracketing*] for an initial bracketing of the minimum followed by a line minimization, using brent's method which is a root-finding algorithm combining the bisection method, the secant method and inverse quadratic interpolation. It has the reliability of bisection but it can be as quick as some of the less reliable methods. The idea is to use the secant method or inverse quadratic interpolation if possible, because they converge faster, but to fall back to the more robust bisection method if necessary. The function needs not be differentiable. Implemented exactly as described in [2].

5 Classifiers

5.1 Feed Forward Neural Networks

5.2 Radial Basis Function Neural networks

5.3 Gaussian Mixture Models [*gmm*]

A gaussian mixture model (gmm) is a generative model modelling a distribution using a mixture of gaussian models. It is defined as following $p(x|\Theta) = \sum_{k=1}^{K} \lambda_k N(x|\mu_k, \Sigma_k)$ Where (μ_k, Σ_k) is the parametrization of a gaussian and λ_k its weight within the mixture.

Assuming we have N samples $X = \{x_1, ..., x_n\}$ which are independent and identically distributed, we have the log-likelihood function defined as:

$$\begin{aligned} L(\Theta|X) &= \log \prod_{i=1}^{N} p(x_i|\Theta) \\ &= \sum_{i=1}^{N} \log p(x_i|\Theta) \\ &= \sum_{i=1}^{N} \sum_{k=1}^{K} \log \lambda_k N(x_i|\mu_k, \Sigma_k) \end{aligned}$$

The log-likelihood is used as this is generally more stable numerically as it will avoid the overflow problem. We found that the parameters updates are:

$$\begin{aligned} \mu_k &= \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{ik} x_i \text{ with} \\ \Sigma_k &= \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{ik} (x_i - \mu_k)(x_i - \mu_k)^T \\ \lambda_k &= \frac{N_k}{N} \end{aligned}$$

With $N_k = \sum_{i=1}^{N} \gamma_{ik}$ and $\gamma_{ik} = \frac{\lambda_k N(x_i|\mu_k, \theta_k)}{\sum_{j=1}^{K} \lambda_j N(x_i|\mu_j, \theta_j)}$

Initial values for $\lambda_k, \mu_k, \Sigma_k$ are found using K-means algorithm. For improved stability of the algorithm, Σ_k is approximated by its diagonal (i.e., often if the number of mixture is too high, the matrix Σ may not be invertible).

In the classification context:
- learning: for each class k, fit a gmm_k with all the samples belonging to the class k.
- classify: $class = \arg\max_k L(x, \Theta_k)$

5.3.1 Derivation of the algorithm

Recall that $\frac{d}{dx}\log f(x) = \frac{f'(x)}{f(x)}$ and $\frac{d}{dx}e^x = x'e^x$, lets compute μ_k:

$$\begin{aligned}
\frac{\partial L(\Theta|X)}{\partial \mu_k} &= \sum_{i=1}^{N} \frac{\lambda_k N(x_i|\mu_k,\theta_k)}{\sum_{j=1}^{K}\lambda_j N(x_i|\mu_j,\theta_j)}\Sigma_k^{-1}(x_i-\mu_k) \\
&= \sum_{i=1}^{N}\gamma_{ik}\Sigma_k^{-1}(x_i-\mu_k) \text{ with} \\
\text{rearranging } \sum_{i=1}^{N}\gamma_{ik}\mu_k &= \sum_{i=1}^{N}\gamma_{ik}x_i \\
\boldsymbol{\mu_k} &= \frac{\mathbf{1}}{\boldsymbol{N_k}}\sum_{\boldsymbol{i=1}}^{\boldsymbol{N}}\boldsymbol{\gamma_{ik}x_i} \text{ with } N_k = \sum_{i=1}^{N}\gamma_{ik} \qquad (20)
\end{aligned}$$

Recall that:

$$\begin{aligned}
\frac{d|M|}{dM} &= |M|(M^{-1})^T \\
\frac{d(a^TMb)}{dM} &= ab^T
\end{aligned}$$

Lets name $c = e^{-\frac{1}{2}(x_n-\mu_k)\Sigma_k(x_n-\mu_k)}$. We have:

$$\begin{aligned}
\frac{\partial c}{\partial \Sigma_k^{-1}} &= c(-\frac{1}{2})(x_n-\mu_k)(x_n-\mu_k)^T \\
\frac{\partial \lambda_k N(x_n|\mu_k,\Sigma_k)}{\partial \Sigma_k^{-1}} &= \frac{\partial(\frac{\lambda_k|\Sigma_k^{-1}|^{0.5}}{(2\pi)^{D/2}}c)}{\partial\Sigma_k^{-1}} \\
&= \frac{\lambda_k c}{(2\pi)^{D/2}}\frac{\partial(|\Sigma_k^{-1}|^{0.5})}{\Sigma_k^{-1}} + \frac{\lambda_k|\Sigma_k^{-1}|^{0.5}}{(2\pi)^{D/2}}\frac{\partial c}{\partial\Sigma_k^{-1}} \\
&= \frac{\lambda_k c}{(2\pi)^{D/2}}\frac{1}{2}|\Sigma_k^{-1}|^{-1/2}|\Sigma_k^{-1}|\Sigma_k + \frac{\lambda_k|\Sigma_k^{-1}|^{0.5}}{(2\pi)^{D/2}}c(-0.5(x_n-\mu_k)(x_n-\mu_k)^T) \\
&= \frac{1}{2}\frac{\lambda_k|\Sigma_k^{-1}|^{0.5}c}{(2\pi)^{D/2}}(\Sigma_k-(x_n-\mu_k)(x_n-\mu_k)^T) \\
&= \frac{1}{2}N(x_n|\mu_k,\Sigma_k)(\Sigma_k-(x_n-\mu_k)(x_n-\mu_k)^T) \\
\frac{\partial lnp(X|\Theta)}{\partial\Sigma_k^{-1}} &= \sum_{n=1}^{N}\frac{1}{\sum_{j=1}^{K}\lambda_k N(x_n|\mu_j,\Sigma_j)}\frac{\partial\lambda_k N(x_n|\mu_k,\Sigma_k)}{\partial\Sigma_k^{-1}} \\
&= \frac{1}{2}\sum_{n=1}^{N}\frac{\lambda_k N(x_n|\mu_k,\Sigma_k)(\Sigma_k-(x_n-\lambda_k)(x_n-\lambda_k)^T))}{\sum_{j=1}^{K}\lambda_j N(x_n|\mu_j,\Sigma_j)} \\
&= \frac{1}{2}\sum_{n=1}^{N}\gamma_{nk}(\Sigma_k-(x_n-\lambda_k)(x_n-\lambda_k)^T))
\end{aligned}$$

Setting $\frac{\partial lnp(X|\Theta)}{\partial\Sigma_k^{-1}}$ to 0, we get:

$$\boldsymbol{\Sigma_k} = \frac{\mathbf{1}}{\boldsymbol{N_k}}\sum_{\boldsymbol{n=1}}^{\boldsymbol{N}}\boldsymbol{\gamma_{nk}(x_n-\mu_k)(x_n-\mu_k)^T}$$

Finally, compute λ_k

$$\frac{\partial lnp(X|\Theta)}{\partial \mu_k^{-1}} = \sum_{n=1}^{K} \frac{N(x_n|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} N(x_j|\mu_j, \Sigma_j)}$$
$$= \sum_{n=1}^{K} \frac{\gamma_{nk}}{\lambda_k} \text{ (i.e., we just add the missing term from } \gamma_{nk})$$

We need to consider the constraint $\sum_{k=1}^{K} \lambda_k = 1$ and form the Lagrangian

$$L = \ln p(X|\Theta) + \lambda((\sum_{k=1}^{K} \lambda_k - 1))$$
$$\frac{\partial L}{\partial \lambda_k} = 0$$
$$0 = \sum_{n=1}^{K} \frac{\gamma_{nk}}{\lambda_k} - \lambda$$
$$\sum_{k=1}^{K}\sum_{n=1}^{N} \gamma_{nk} = \lambda \sum_{n=1}^{K} (\lambda_K)$$
$$\sum_{n=1}^{N} 1 = \lambda$$
$$\lambda = N$$

Substituting we have: $\boldsymbol{\lambda_k} = \dfrac{\boldsymbol{N_k}}{\boldsymbol{N}}$

5.4 Boosting

5.5 Decision Tree

5.6 Stump

5.7 Perceptron

6 Data Preprocessing

6.1 Feature Selection

blabla

8 Notations

- function f is defined throughout the document as $f : \mathbf{R}^n \rightarrow \mathbf{R}$
- U(a..b) generate a random number uniformly in the range [a..b]
- N(m, v) generate a random number following a gaussian distribution mean m and standard deviation v
- for a discrete random X, $\#X$ will denote the number of states of X

Glossary

CLG Conditional Linear Gaussian, 10

GMM Gaussian Mixture Model, 15

LG Linear Gaussian, 9

MTE Mixture of truncated exponentials, 10

References

[1] Juliana Gambini Hugo D. Scolnik. A new method to compute second derivatives, 2001.

[2] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 3 edition, 2007.