

6.1 Apprentissage par EM pour l'analyse factorielle

On reprend les notations du cours de la semaine dernière :

$$Y \in \mathbb{R}^p, X \in \mathbb{R}^q, q \leq p$$

$$Y = \Lambda X + \mu + \text{bruit}$$

Sous les hypothèses :

$$\mathcal{L}(X) = \mathcal{N}(0, I), \quad \mathcal{X}|\mathcal{Y} = \mathcal{N}(\Lambda X + \mu, \psi)$$

Nous avons montré que (X, Y) et $X|Y$ étaient gaussiens et trouvé leurs paramètres respectifs. Pour les apprendre, on peut appliquer la recette EM :

- On se donne $Y_1, \dots, Y_n \in \mathbb{R}^p$, on souhaite estimer Λ, ψ et μ .
- On écrit la vraisemblance complète :

$$l := \log p(X, Y) = \sum_i \log p(X_i|\theta) + \sum_i \log p(Y_i|X_i, \theta)$$

qui devient ici :

$$l = \text{cste} - \sum_i \log((2\pi)^{p/2} \sqrt{\det \psi}) + 0.5(Y_i - \mu - \Lambda X_i)' \psi^{-1} (Y_i - \mu - \Lambda X_i)$$

- On répète les étapes E-Step et M-Step consistant à prendre l'espérance de cette log-vraisemblance par rapport à la variable observée Y et à la maximiser par rapport à μ, Λ et ψ .

6.2 Inférence sur les MG : l'arbre de jonction

6.2.1 Le cadre

Soit $G = (V, E)$ un graphe non orienté. On suppose toujours que G n'a qu'une composante connexe. Soit \mathcal{C} l'ensemble des cliques maximales de G .

6.2.2 Définitions

Définition 6.1 (Arbre de cliques) $\mathcal{T} = (\mathcal{C}', \mathcal{E})$ est un arbre de cliques si :

- \mathcal{T} est un arbre,
- Il est couvrant, c'est à dire $\mathcal{C}' = \mathcal{C}$.
- Les arêtes vérifient la propriété : $(C, D) \in \mathcal{E} \Rightarrow C \cap D \neq \emptyset$

Définition 6.2 (Arbre de jonction) Un arbre de jonction est un arbre de cliques $\mathcal{T} = (\mathcal{C}, \mathcal{E})$ vérifiant l'une des deux propriétés équivalentes suivantes :

1. $\forall v \in V$, le sous-arbre des cliques contenant v est connexe.
2. $\forall (C, D) \in \mathcal{C} \times \mathcal{C}$, $C \cap D$ est inclus dans chaque clique de l'unique chemin de C à D .

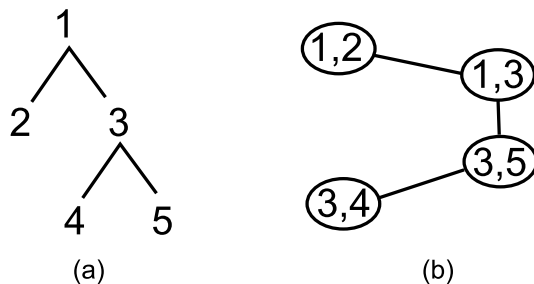


FIGURE 6.1 : (a) Graphe. (b) Arbre de jonction



Tout graphe G admet au moins un (et souvent plusieurs) arbre de cliques, mais pas forcément un arbre de jonctions (cf sections suivantes sur la triangulation).

Définition 6.3 (Séparateur) L'intersection de deux cliques voisines est appelée un séparateur.

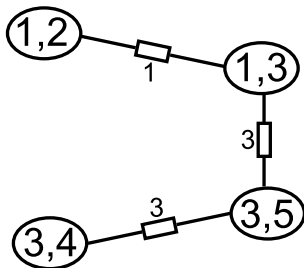


FIGURE 6.2 : Séparateurs

6.3 Algorithme Somme-Produit

L'objet de cette section est de montrer qu'on sait faire de l'inférence sur un arbre de jonction. Il suffira alors de savoir s'y ramener à partir de n'importe quel MG. L'algorithme somme-produit a été défini précédemment sur un arbre, il s'étend comme suit sur un arbre de jonctions.

6.3.1 L'algorithme

On suppose que l'on a un arbre de jonction $\mathcal{T} = (\mathcal{C}, \mathcal{E})$.

Définition 6.4 Pour un couple $(C_i, C_j) \in \mathcal{E}$, le message $m_{C_i C_j}$ de $i \rightarrow j$ est défini par :

$$m_{C_i C_j}(x_{C_i \cap C_j}) := \sum_{x_{C_i \setminus (C_i \cap C_j)}} \psi_{C_i}(x_{C_i}) \prod_{C_k \in \mathcal{N}(C_i)} m_{C_k C_i}(x_{C_k \cap C_i})$$

Le protocole de passage des messages consiste à transmettre le message de $i \rightarrow j$ seulement après avoir reçu les messages de tous les voisins de i .

Théorème 6.5 Après avoir passé les $2(|\mathcal{C}| - 1)$ messages en respectant le "protocole", alors :

$$\forall C_i, \quad u(x_{C_i}) = \psi_{C_i}(x_{C_i}) \prod_{C_k \in \mathcal{N}(C_i)} m_{C_k C_i}(x_{C_i})$$

6.3.2 Preuve de l'algorithme

Soit $\mathcal{T} = (\mathcal{C}, \mathcal{E})$ un arbre de jonction. On va montrer qu'on peut se ramener au cas de l'algorithme somme-produit pour un arbre que nous avons déjà traité, en construisant les potentiels ad-hoc.

On introduit des copies locales et un potentiel :

$$\forall C \in \mathcal{C} \quad \tilde{x}_C = \{\tilde{x}_{C,i}, \quad i \in C\}$$

$$\forall (C, D) \in \mathcal{C}^2 \quad \psi_{C,D}(\tilde{x}_C, \tilde{x}_D) = \prod_{i \in C \cap D} \delta(\tilde{x}_{C,i} = \tilde{x}_{D,i})$$

On a alors :

$$u(x) = \prod_{C \in \mathcal{C}} \psi_C(x_C)$$

$$\tilde{u}(\tilde{x}) = \prod_{C \in \mathcal{C}} \psi_C(\tilde{x}_C) \prod_{(C,D) \in \mathcal{E}} \underbrace{\prod_{i \in C \cap D} \delta(\tilde{x}_{C,i} = \tilde{x}_{D,i})}_{\psi_{C,D}(\tilde{x}_C, \tilde{x}_D)}$$

Lemme 6.6 Si toutes les copies locales de chaque x_i sont cohérentes ($\tilde{x}_{C,i} = \tilde{x}_{D,i} = x_i$) alors $\tilde{u}(\tilde{x}) = u(x)$ sinon $\tilde{u}(\tilde{x}) = 0$

Démonstration

- Si toutes les copies locales de chaque x_i sont cohérentes, le résultat est trivial.
- Sinon : Supposons qu'il existe i, C, D tels que $i \in C \cap D$ et $\tilde{x}_{C,i} \neq \tilde{x}_{D,i}$. Le résultat est immédiat si $(C, D) \in \mathcal{E}$. Le fait que nous ayons affaire à un arbre de jonction permet de traiter le cas $(C, D) \notin \mathcal{E}$, en utilisant la deuxième propriété définissant les arbres de jonctions :

$$\forall C_1 = C, C_2, \dots, C_K = D / (\forall k) \quad C_k \in \mathcal{N}(C_{k-1}), \text{ on a } (\forall k) \quad i \in C_k$$

$$\text{Ainsi } \exists k \leq K, \quad \tilde{x}_{C_k,i} \neq \tilde{x}_{C_{k+1},i}$$

$$\text{Et donc } \tilde{u}(\tilde{x}) = 0$$

■

Fin de la preuve de l'algorithme :

$$m_{C_i, C_j}(\tilde{x}_{C_j}) = \sum_{\tilde{x}_{C_i}} \psi_{C_i, C_j}(\tilde{x}_{C_i}, \tilde{x}_{C_j}) \underbrace{\psi_{C_i}(\tilde{x}_{C_i}) \prod_{C_k \in \mathcal{N}(C_i)} m_{k,i}(\tilde{x}_{C_i})}_{=F(\tilde{x}_{C_i})}$$

$$\begin{aligned} m_{C_i, C_j}(\tilde{x}_{C_i \cap C_j, C_j}, \tilde{x}_{C_i \setminus (C_i \cap C_j), C_j}) &= \sum_{\tilde{x}_{C_i}} F(\tilde{x}_{C_i}) \psi_{C_i, C_j}(\tilde{x}_{C_i}, \tilde{x}_{C_j}) \\ &= \sum_{\tilde{x}_{C_i \cap C_j, C_j}} F\left(\begin{array}{c} \tilde{x}_{C_i \cap C_j, C_j} \\ \tilde{x}_{C_i \setminus (C_i \cap C_j), C_i} \end{array}\right) \end{aligned}$$

Comme on a un arbre de jonction, tous les \tilde{x} sont consistants et donc :

$$\begin{aligned} m_{C_i, C_j}(x_{C_i \cap C_j}) &= \sum_{\tilde{x}_{C_i}} F(\tilde{x}_{C_i}) \psi_{C_i, C_j}(\tilde{x}_{C_i}, \tilde{x}_{C_j}) \\ &= \sum_{x_{C_i \setminus (C_i \cap C_j)}} F\left(\begin{array}{c} \tilde{x}_{C_i \cap C_j} \\ x_{C_i \setminus (C_i \cap C_j)} \end{array}\right) \end{aligned}$$

Faire passer les messages pour $\tilde{u}(\tilde{x})$ revient à faire passer les messages dans l'arbre de jonction. L'algorithme Somme-Produit est donc exact pour $\tilde{u}(\tilde{x})$ si et seulement si il l'est pour $u(x)$: on s'est bien ramené au cas connu.

Proposition 6.7 Une fois les messages passés :

$$u(x) = \prod_{C_i \in \mathcal{C}} \psi_{C_i}(x_{C_i}) = \frac{\prod_{\text{Cliques}} u(x_C)}{\prod_{\text{Separateurs}} u(x_S)}$$

Exemple

Soit G le graphe suivant :

$$\text{D'après la proposition précédente : } u(x) = \frac{u(x_1, x_2)u(x_2, x_3)}{u(x_2)}.$$

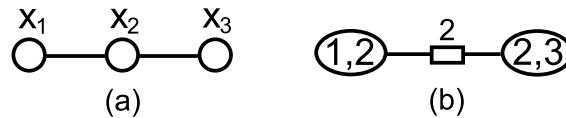


FIGURE 6.3 : Exemple (a) Graphe. (b) Arbre de jonction.

6.3.3 Complexité

Soit r le nombre de valeurs que peut prendre chaque variable x_i .

1. Algorithme naïf : $u(x_C) = \sum_{x_{V \setminus C}} u(x) \rightarrow$ Complexité en $O(r^{|V|})$
2. Algorithme Somme-Produit :
 - Complexité du calcul de $m_{C_i C_j}$ en $O(r^{|C_i|})$.
 - Complexité totale en $O(|\mathcal{C}| r^{\max_{C \in \mathcal{C}} |C|})$

L'algorithme Somme-Produit est donc exponentiel en la taille de la clique maximale, mais seulement linéaire en le nombre de cliques.

6.4 Triangulation

Comme vu précédemment, tout graphe n'admet pas forcément un arbre de jonction. Néanmoins, en rajoutant potentiellement des arêtes pour le rendre triangulé, on se ramène à un graphe qui admet un arbre de jonction.

Définition 6.8 Soit $G = (V, E)$ un graphe non orienté.

Un cycle sans corde (Figure 6.4) est défini comme un cycle C d'ordre ≥ 4 tel que :

$$\forall (u, v) \in C, (u, v) \in E \Rightarrow (u, v) \text{ adjacents dans } C.$$

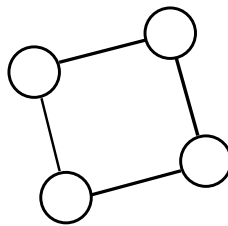


FIGURE 6.4 : Cycle sans corde.

Définition 6.9 Soit $G = (V, E)$ un graphe non orienté. G est triangulé s'il ne contient pas de cycle sans corde (Figure 6.5).

Théorème 6.10 Soit G non orienté, I un ordre d'élimination. Le graphe reconstitué après élimination est triangulé.

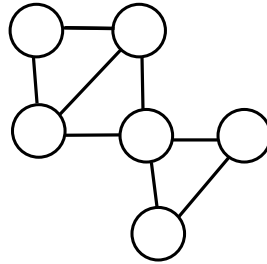


FIGURE 6.5 : Graphe triangulé.

Démonstration La preuve se fait par récurrence sur le nombre de sommets $n := \#V$. Pour le cas de base $n = 1$, le graphe constitué d'un nœud unique est nécessairement triangulé. On suppose l'hypothèse vraie pour un graphe à n sommets et on montre que c'est toujours vrai pour un graphe à $n + 1$ sommets.

Prenons un graphe $G = (V, E)$ à $n + 1$ sommets, la première itération de l'algorithme d'élimination élimine le sommet $\{n + 1\}$ et relie tous ses voisins. Le résultat est un graphe à n sommets $\tilde{G} = (\tilde{V}, \tilde{E})$ où $\tilde{V} = V \setminus \{n + 1\}$ et $\tilde{E} = (E \cap (\tilde{V} \times \tilde{V})) \cup (N(n + 1) \times N(n + 1))$.

D'après l'hypothèse de récurrence, en appliquant l'algorithme d'élimination sur \tilde{G} , on obtient un graphe reconstitué $rec(\tilde{G})$ triangulé.

En ajoutant le sommet $\{n + 1\}$ au graphe $rec(\tilde{G})$, on obtient le graphe $rec(G)$ qui ne contient aucun cycle sans corde car tous les voisins du nœud $\{n + 1\}$ ont été connectés. Par conséquent, le graphe reconstitué $rec(G)$ est triangulé (Figure 6.6). ■

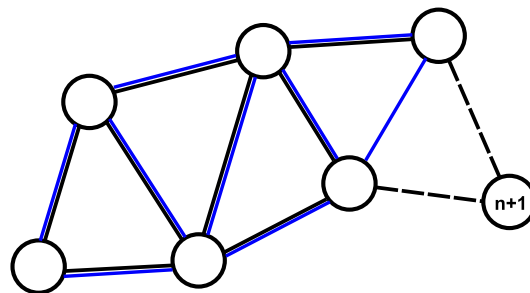


FIGURE 6.6 : Graphe reconstitué triangulé.

Théorème 6.11 G est triangulé \Leftrightarrow Il existe un ordre d'élimination laissant G invariant.

- G est triangulé \Rightarrow il existe un ordre d'élimination laissant G invariant est un corollaire du théorème précédent.
- Il existe un ordre d'élimination laissant G invariant $\Rightarrow G$ est triangulé est admis.

Théorème 6.12 G est triangulé $\Leftrightarrow G$ admet un arbre de jonction JT (Junction Tree).

Démonstration

1. G est triangulé $\Rightarrow G$ admet un arbre de jonction (preuve par récurrence) :

Le cas de base où G est un graphe à un seul nœud est trivial.

On considère que le graphe $G = (V, E)$ à $n + 1$ nœuds est triangulé. Selon le théorème précédent, il existe un ordre d'élimination qui le laisse invariant.

Soit $\{\alpha\}$ le premier nœud éliminé. Le graphe $G \setminus \{\alpha\}$ obtenu est triangulé. Ainsi, par hypothèse de récurrence, le graphe triangulé $G \setminus \{\alpha\}$ admet un arbre de jonction T .

Soit C_α la clique contenant α et ses voisins.

Si $C = C_\alpha \setminus \{\alpha\}$ est une clique dans \mathcal{T} , alors en ajoutant $\{\alpha\}$ à cette clique, l'arbre de jonction T contenant la clique augmentée est un arbre de jonction pour G .

Si $C = C_\alpha \setminus \{\alpha\}$ n'est pas une clique dans \mathcal{T} , alors c'est un sous ensemble d'une clique D dans T . Dans ce cas C_α peut être ajoutée comme une feuille de T , relié à D et un ensemble séparateur $S = C$. Le résultat est un arbre de jonction.

2. G admet un arbre de jonction $\Rightarrow G$ est triangulé (pour cette preuve se référer au livre de M. Jordan : 17.14.2).

■

En conclusion, lorsqu'on veut faire de l'inférence sur un graphe non triangulé, on le triangule par élimination, puis on utilise l'algorithme somme-produit sur son arbre de jonction. Il nous est donc nécessaire de savoir construire ce dernier.

6.5 Construction de l'arbre de jonction

Il est clair à présent que tout graphe triangulé admet un arbre de jonction. Néanmoins, tout arbre de cliques obtenu à partir d'un graphe triangulé n'est pas un arbre de jonction. Si l'on considère le graphe de la Figure 6.7(a), l'arbre de cliques de la Figure 6.7(b) n'est pas un arbre de jonction. Ce graphe admet comme arbre de jonction l'arbre de la Figure 6.7(c).

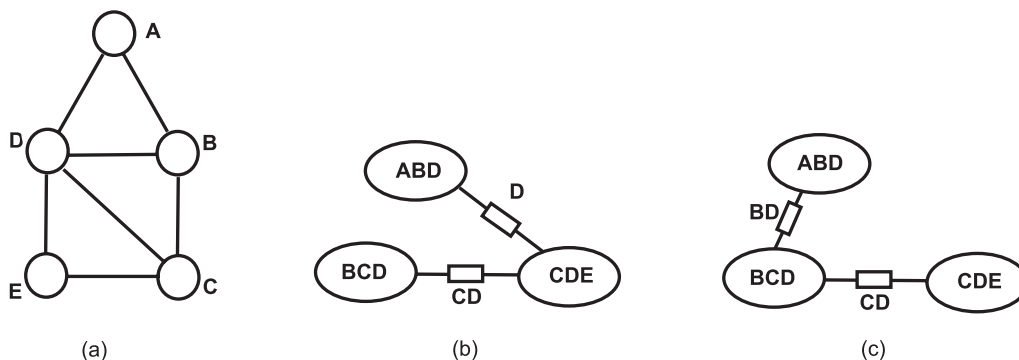


FIGURE 6.7 : (a) Graphe triangulé. (b) Arbre de cliques. (c) Arbre de jonction.

Chaque arbre de cliques associé à un graphe triangulé est caractérisé par un poids $w(\mathcal{T})$ relatif à la somme des cardinaux des ensembles séparateurs. Un arbre de cliques sera un arbre de jonction si et seulement s'il est de poids maximal.

Soit le graphe $G = (V, E)$ associé à un arbre de cliques $\mathcal{T} = (\mathcal{C}, \mathcal{E})$.
Notons \mathcal{W} la fonction qui compte le nombre de séparateurs entre deux cliques. Elle sera appliquée aux nœuds de l'arbre de cliques qui ne sont rien d'autre que des cliques.

$$\begin{aligned} \mathcal{W} : \mathcal{C} \times \mathcal{C} &\rightarrow \mathbf{R}_+ \\ (C, D) &\rightarrow \text{Card}(C \cap D) \end{aligned}$$

Lemme 6.13 $\forall \mathcal{T}$ arbre de cliques :

$$\forall v \in V, \quad \sum_{(C,D) \in \mathcal{E}} \delta(v \in C \cap D) \leq \sum_{C \in \mathcal{C}} \delta(x_v \in C) - 1. \quad (6.1)$$

Le nombre d'apparitions d'un séparateur dans l'arbre de cliques est inférieur au nombre de cliques qu'il sépare moins un.

Il y a donc égalité pour $v \in V$ ssi le sous-arbre formé des cliques auxquelles v appartient est connexe.

Démonstration Considérons le sous arbre \mathcal{T}_v (de cliques) des cliques contenant v . Il est clair que : (nombre d'arêtes de \mathcal{T}_v) \leq (nombre de sommets de \mathcal{T}_v) $- 1$. ■

Lemme 6.14 $\forall \mathcal{T}$ arbre de cliques :

$$\sum_{(C,D) \in \mathcal{E}} \mathcal{W}(C, D) \leq \sum_{C \in \mathcal{C}} \text{Card}(C) - \text{Card}(V). \quad (6.2)$$

avec égalité ssi l'arbre \mathcal{T} est un arbre de jonction.

Démonstration

$$\sum_{(C,D) \in \mathcal{E}} \text{Card}(C \cap D) = \sum_{v \in V} \sum_{(C,D) \in \mathcal{E}} \delta(v \in C \cap D) \quad (6.3)$$

$$\leq \sum_{v \in V} \left(\sum_{C \in \mathcal{C}} \delta(v \in C) - 1 \right) \quad (6.4)$$

$$= \sum_{C \in \mathcal{C}} \text{Card}(C) - \text{Card}(V). \quad (6.5)$$

Il y a égalité ssi pour tout v , le sous-arbre \mathcal{T}_v est connexe, i.e., exactement si on a un arbre de jonction. ■

Le résultat précédent est indépendant de \mathcal{T} . Ceci signifie que, pour n'importe quel arbre, on ne peut obtenir un poids $w(\mathcal{T})$ supérieur à $\sum_{C \in \mathcal{C}} \text{Card}(C) - \text{Card}(V)$.

Si l'inégalité devient une égalité, c'est que \mathcal{T} est un arbre de jonction. Ainsi, obtenir l'arbre de jonction revient à trouver l'arbre couvrant de poids maximum pour l'arbre de cliques (Problème de l'arbre couvrant de poids maximum). Il s'agit donc de trouver \mathcal{T} tel que : $\mathcal{T} \in \arg \max \left(\sum_{(C,D) \in \mathcal{C}} \mathcal{W}(C,D) \right)$.

Le problème de l'arbre couvrant de poids maximum peut être résolu de manière exacte par l'algorithme de Kruskal (algorithme glouton). Cependant la solution optimale n'est pas unique.

Algorithme de Kruskal

Pour $i = 1, \dots, \text{Card}(V) - 1$:

Choisir l'arête de poids maximum ne créant pas de cycle (Figure 6.8).

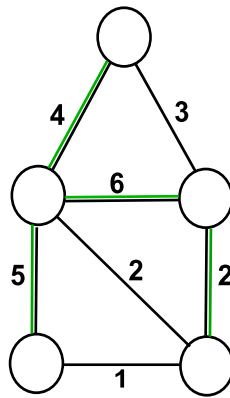


FIGURE 6.8 : Arbre couvrant de poids maximum (6,5,4,2).

Complexité $O(n^2 \log n)$, avec $n = \text{Card}(V)$.

6.6 Inférence exacte

Résumé des étapes de l'algorithme :

1. Moralisation (si le graphe est orienté).
2. Introduction des observations (évidence).
3. Construction de l'arbre de jonction :
 - Sélectionner un ordre d'élimination.
 - Triangulation par élimination et extraction des cliques qui formeront les nœuds du futur arbre.

– Création d'un arbre couvrant de poids maximum (Algorithme de Kruskal).

4. Propagation des messages sur l'arbre de jonction.

Cette procédure est une procédure imbattable pour calculer l'inférence exacte. Cependant sa mise en œuvre peut s'avérer trop longue. On lui préférera un calcul approché. Le véritable problème bloquant est l'étape 4 dont la complexité est exponentielle en $\#C_{max}$ (taille de la clique maximale après triangulation). Le cas le plus favorable est celui d'un arbre ($tw = 1$), un des cas les plus défavorables est celui de la grille ($tw \geq \sqrt{n}$).

L'arbre couvrant obtenu à la fin de l'étape 3 donne donc une information sur l'ordre de grandeur de la complexité de l'étape 4 au travers du calcul de la largeur arborescente de l'arbre de jonction. Selon la valeur de cette complexité, on peut alors décider de lancer ou non la dernière étape avant même d'avoir vu les données.

De plus, au début de l'étape 3, trouver l'ordre d'élimination (ou la triangulation) qui permet d'obtenir la plus petite clique de taille maximale est un problème NP-difficile. Il peut être résolu de manière approchée par des algorithmes moins gloutons et non optimaux.

6.7 Entropie

On suppose que la variable aléatoire X prend ses valeurs dans un ensemble fini \mathcal{X} .

Définition 6.15 (Entropie H)

$$H(X) = - \sum_{x \in \mathcal{X}} p(x) \log p(x) = \mathbf{E}(\log \frac{1}{p(x)})$$

Proposition 1 On a

1. $H(X) \geq 0$
2. $H(X) \leq \log(\text{Card}(\mathcal{X}))$ avec égalité si $p(x)$ est uniforme.

Démonstration

Pour 1, $(\frac{1}{p(x)} \geq 1, \forall x \in \mathcal{X})$ implique le résultat.

Pour le 2, on peut appliquer l'inégalité de Jensen à la fonction $f : p \mapsto p \log p$.

Montrons que celle ci est convexe :

$$\begin{aligned} f'(p) &= \log(p) + 1 \\ f''(p) &= \frac{1}{p} > 0 \end{aligned}$$

On a alors,

$$H(X) = \mathbf{E}(\log(\frac{1}{p(X)})) \leq \log \mathbf{E}(\frac{1}{p(X)}) = \log(\text{Card}(\mathcal{X}))$$

■

Lecture : Elements of Information Theory, par Thomas M. Cover, Joy A. Thomas, Wiley.

6.8 KL-divergence (Kullback–Leibler)

6.8.1 Définition et propriétés de base

Définition 6.16 (KL-divergence D)

$$D(p||q) = \sum_{x \in \mathcal{X}} p(x) \log\left(\frac{p(x)}{q(x)}\right)$$

avec $p(x)$ et $q(x)$ deux lois.

Proposition 2 On a

1. $D(p||q) \geq 0$ avec égalité ssi $p=q$,
2. D n'est pas symétrique : $D(p||q) \neq D(q||p)$.

Démonstration On a par Jensen :

$$D(p||q) = - \sum_{x \in \mathcal{X}} p(x) \log\left(\frac{q(x)}{p(x)}\right) \geq - \log\left(\sum_{x \in \mathcal{X}} p(x) \cdot \frac{q(x)}{p(x)}\right) = 0$$

avec égalité ssi $p = q$. ■

6.8.2 Lien entre KL-divergence et Maximum Likelihood

Soit $x \in \mathcal{X}$ ensemble fini, x_1, \dots, x_n des observations i.i.d.

Définition 6.17 (Loi empirique)

$$\hat{p}(x) = \frac{1}{N} \sum_{n=1}^N \delta(x = x_n)$$

Proposition 3

$$\begin{aligned} D(\hat{p}||q) &= \sum_{x \in \mathcal{X}} \hat{p}(x) \log\left(\frac{\hat{p}(x)}{q(x)}\right) \\ &= -H(\hat{p}(x)) - \sum_{x \in \mathcal{X}} \hat{p}(x) \log(q(x)) \\ &= -H(\hat{p}(x)) - \sum_{n=1}^N \log(p(x_n)) \end{aligned}$$

Lien : maximiser $\sum_{n=1}^N \log q(x_n)$ équivaut à minimiser $D(\hat{p}||q)$.



Ceci n'est pas la même chose que $D(q||\hat{p})!$

6.8.3 Lien entre KL-divergence et entropie

$$D(p||\text{loi uniforme}) = \sum_{x \in \mathcal{X}} p(x) \log \frac{p(x)}{1/\text{Card}\mathcal{X}} = -H_{p(x)}(X) + \log \text{Card}\mathcal{X}$$