POLYTECHNIQUE
MONTRÉAL

LE GÉNIE
EN PREMIÈRE CLASSE

Polytechnique Montréal

INF8225

Intelligence artificielle : modèles probabilistes et apprentissage

TP 1

Auteur:
Zins Pierre 1863527

Janvier 16, 2017

Contents

1 Question 1

Pour expliquer, les trois phénomènes *Explaining away*, *Serial blocking* et *Divergent blocking*, je me suis basé sur le réseau bayésien suivant :

- Pollution (trop importante): P
- Smoker : S
- Cancer : C
- XRay : X
- Dispnea : D

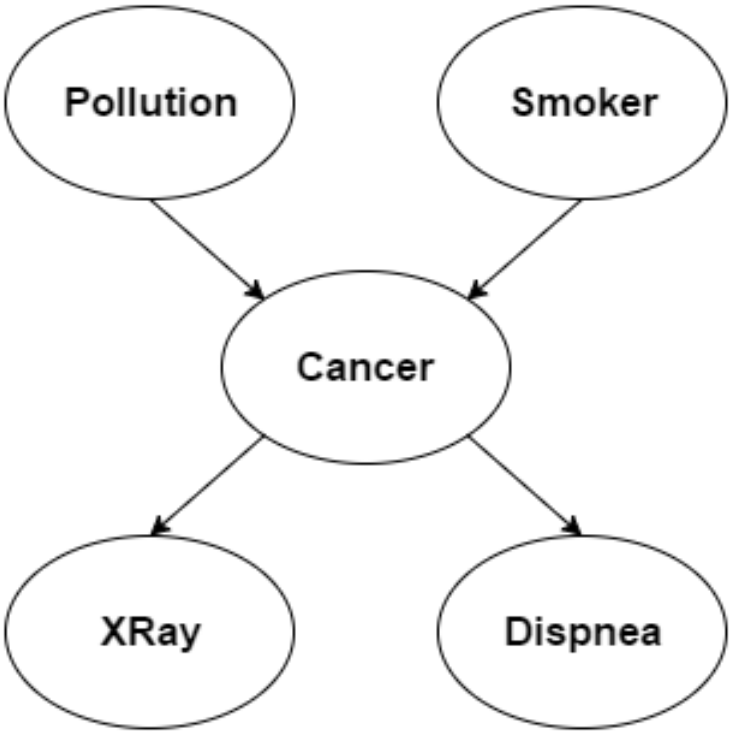

Figure 1: Réseau bayésien

Voici les tables de probabilités associées :

Pollution :

P(P=F)	P(P=V)
0.9	0.1

Smoker :

P(S=F)	P(S=V)
0.7	0.3

Cancer :

P	S	P(C=F)	P(C=V)
V	V	0.95	0.05
V	F	0.98	0.02
F	V	0.97	0.03
F	F	0.999	0.0001

XRay :

C	P(X=F)	P(X=V)
V	0.1	0.9
F	0.8	0.2

Dispnea :

C	P(D=F)	P(D=V)
V	0.35	0.65
F	0.7	0.3

1.1 Explaining away

1.1.1 Explication théorique

Dans un réseau bayésien, le phénomène de "explaining away" est présent lorsque l'on a au moins trois nœuds : les nœuds A et C parents d'un nœud B. Au départ, A et C sont indépendants, mais à partir du moment où l'on a une information sur B (B est observé) ou un de ses enfants, A et C deviennent dépendants. En effet, un changement sur la probabilité de A entraînera l'effet inverse sur la probabilité de C. Par exemple, si la probabilité de augmente et approche 1, cela suffira a expliquer B et donc la probabilité de C va baisser.

1.1.2 Explication pratique

A partir de mon réseau bayésien, on peut remarquer ce phénomènes entre les nœuds S, P et C. Au départ, S et P sont indépendants comme on peut le voir à partir des égalités suvantes : $P(S = 1) == P(S = 1|P = 1) == P(S = 1|P = 0) == 0.3$
Cependant, si C est observé, alors S et P deviennent dépendants :

- Si l'on a l'information est que P est vrai alors c'est lui qui va expliquer le fait que C soit vrai et ainsi la probabilité de S va baisser :

$P(S = 1|C = 1, P = 1) = 0.517 \quad < \quad P(S = 1|C = 1) = 0.825$

- à l'inverse si l'information est que P est faux alors c'est S qui va expliquer le fait que C soit vrai et ainsi la probabilité de S va augmenter :
  $P(S = 1|C = 1, P = 0) = 0.928 \quad > \quad P(S = 1|C = 1) = 0.825$

1.2 Serial blocking

1.2.1 Explication théorique

Pour ce phénomène nous avons la situation suivante : un nœud A parent d'un nœud B, lui même parent d'un nœud C. Au départ, les trois nœuds sont dépendants, un indice sur le fait que C est vrai, va augmenter la probabilité de B qui va ensuite faire augmenter la probabilité de A. Cependant, si B est observé, A et C vont devenir indépendants. En effet, le fait d'observer B va couper le lien de dépendance entre A et C. Le fait que la probabilité de C augmente sera uniquement expliqué par le fait que B soit observé et ainsi, il n'y aura aucun effet sur A. De même dans le sens inverse, tout changement sur A sera "absorbé" par B et n'affectera pas C.

1.2.2 Explication pratique

Ce phénomène est visible au travers des nœuds S, C et X qui sont dépendants entre eux au départ. S et X ne sont donc pas indépendants comme on peut le vérifier puisque : $P(X = 1) = 0.202 \quad != \quad P(X = 1|S = 0) = 0.222 \quad != \quad P(X = 1|S = 1) = 0.208$
Cependant, si l'on observe C, alors X et S vont devenir indépendants et on a les égalités suivantes :
$P(X = 1|C = 1) == P(X = 1|C = 1, S = 0) == P(X = 1|C = 1, S = 1) == 0.9$

1.3 Divergent blocking

1.3.1 Explication théorique

Pour ce phénomène, nous devons considérer la situation suivante : un nœud B parent des nœuds A et C. Au départ, A et C dépendants car une augmentation de probabilité sur A va entraîner une augmentation sur B qui ensuite fera augmenter

celle de C. Cependant, si B est observé A et C vont devenir indépendants. En effet, comme dans le cas du *Serial blocking*, le fait d'observer B va couper le lien de dépendance entre A et C. Par exemple, une augmentation de la probabilité de A sera uniquement expliquée par B et n'aura aucun impact sur C.

1.3.2 Explication pratique

Ce phénomène est visible au travers des nœuds C, D et X qui sont dépendants entre eux au départ. D et X ne sont donc pas indépendants comme on peut le vérifier puisque : $P(X = 1) = 0.204 \quad != \quad P(X = 1|D = 0) = 0.217 \quad != \quad P(X = 1|D = 1) = 0.208$
Cependant, si l'on observe C, alors X et S vont devenir indépendants et on a les égalités suivantes :
$P(X = 1|C = 1) == P(X = 1|C = 1, D = 0) == P(X = 1|C = 1, D = 1) == 0.9$

2 Question 2

2.1 a)

J'ai crée le réseau bayésien *Alarme* dans Matlab à l'aide de *PMTK*. Le code est disponible dans le fichier **question2.m**

2.2 b)

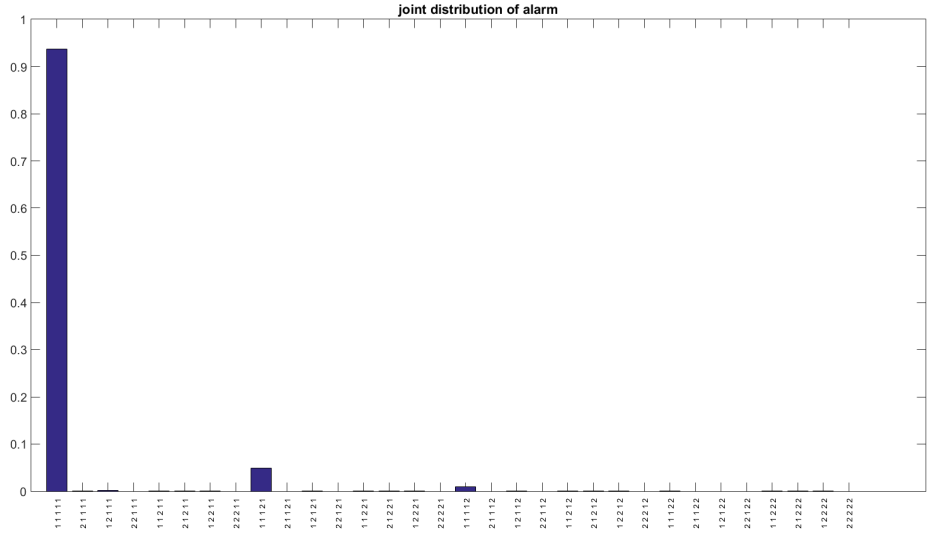

Figure 2: Réseau bayésien

Voici l'histogramme de la probabilité jointe du modèle.
Pour les abscisses, on a:

- Cambriolage Tremblement Alarme MarieAppelle JeanAppelle
- avec 1 = F et 2 = V

2.3 c)

Avec PMTK, j'ai pu calculer les différentes probabilités.

- $P(Cambriolage = V | MarieAppelle = V, JeanAppelle = F) = 0.005130$
- $P(Cambriolage = V | MarieAppelle = F, JeanAppelle = V) = 0.006876$
- $P(Cambriolage = V | MarieAppelle = V, JeanAppelle = V) = 0.284172$
- $P(Cambriolage = V | MarieAppelle = F, JeanAppelle = F) = 0.000090$
- $P(Cambriolage = V | MarieAppelle = V) = 0.016284$
- $P(Cambriolage = V | JeanAppelle = V) = 0.056117$

2.4 d)

- $P(Cambriolage = V) = 0.001000$
- $P(Tremblement = V) = 0.002000$
- $P(Alarme = V) = 0.002516$
- $P(MarieAppelle = V) = 0.052139$
- $P(JeanAppelle = V) = 0.011736$

2.5 e)

Voici les équations permettant de cacluler les probabilités.

$$
\begin{aligned}
P(J) &= \sum_C \sum_T \sum_A \sum_M P(C,T,A,M,J) \\
&= \sum_C \sum_T \sum_A \sum_M P(C) * P(T) * P(A|C,T) * P(M|A) * P(J|A) \\
&= \sum_C \sum_T \sum_A P(C) * P(T) * P(A|C,T) * P(J|A) * \underbrace{\sum_M * P(M|A)}_{=1} \\
&= \sum_C \sum_T \sum_A P(C) * P(T) * P(A|C,T) * P(J|A) \\
&= \sum_C P(C) * \sum_T P(T) \sum_A P(A|C,T) * P(J|A) \\
&= P(C=V) * (P(T=V) * (P(A=V|C=V,T=V) * P(J=V|A=V) + \\
&P(A=F|C=V,T=V) * P(J=V|A=F)) + P(T=F) * \\
&(P(A=V|C=V,T=F) * P(J=V|A=V) + P(A=F|C=V,T=F) * \\
&P(J=V|A=F))) + P(C=F) * (P(T=V) * (P(A=V|C=F,T=V) * \\
&P(J=V|A=V) + P(A=F|C=F,T=V) * P(J=V|A=F)) + \\
&P(T=F) * (P(A=V|C=F,T=F) * P(J=V|A=V) + \\
&P(A=F|C=F,T=F) * P(J=V|A=F))) \\
&= 0.001 * (0.002 * (0.950.7 + 0.05 * 0.01) + 0.98 * (0.94 * 0.7 + 0.06 * 0.01)) + \\
&0.999 * (0.002 * (0.29 * 0.7 + 0.71 * 0.01) + 0.998 * (0.001 * 0.7 + 0.999 * 0.01)) \\
&= 0.0117
\end{aligned}
\tag{1}
$$

$$
\begin{aligned}
P(C|J=V) &= \frac{P(C, J=V)}{P(J=V)} \\
P(C, J=V) &= \sum_T \sum_A \sum_M P(C, T, A, M, J=V) \\
&= \sum_T \sum_A \sum_M P(C) * P(T) * P(A|C,T) * P(M|A) * P(J|A) \\
&= \sum_T \sum_A P(C) * P(T) * P(A|C,T) * P(J=V|A) * \underbrace{\sum_M * P(M|A)}_{=1} \\
&= \sum_T \sum_A P(C) * P(T) * P(A|C,T) * P(J=V|A) \\
&= \sum_T P(C) * P(T) \sum_A P(A|C,T) * P(J=V|A) \\
&= P(C=V) * (P(T=V) * (P(A=V|C=V,T=V) * \\
&P(J=V|A=V) + P(A=F|C=V,T=V) * P(J=V|A=F)) + \\
&P(T=F) * (P(A=V|C=V,T=F) * P(J=V|A=V) + \\
&P(A=F|C=V,T=F) * P(J=V|A=F))) \\
&= 0.001 * (0.002 * (0.950.7 + 0.05 * 0.01) + 0.98 * (0.94 * 0.7 + 0.06 * 0.01)) \\
&= 0.000658 \\
P(C|J=V) &= \frac{0.000658}{0.0117} = 0.056
\end{aligned}
\tag{2}
$$

Pour les deux calculs, on retrouve les mêmes résultats que ceux obtenus avec *Matlab* et *PMTK*

3 Question 3

J'ai implémenté l'algorithme **sum-product** en Python. Ce dernier va donc renvoyer les formules littérales qui permettront de calculer des probabilités dans un réseau bayésien.

1. L'utilisateur devra déclarer des noeuds *"Function"* en précisant un nom, ainsi que la valeur de f(x) et des noeuds *"Variable"* en précisant un nom.
2. Ensuite pour créer le graphe des facteurs, il devra utiliser la méthode *addNeighbours* afin de créer les liens entre les noeuds.

3. Enfin, la fonction *getProbabilty(variable, observed_variables)* permettra d'obtenir un résultat.
   - **variable** : variable pour laquelle on veut calculer une probabilité
   - **observed_variables** : liste contenant toutes les variables observées

Exemple

```
# declaration of the factor graph
fa = Function("fa", "P(x1)")
fb = Function("fb", "P(x2)")
fc = Function("fc", "P(x3|x1)")
fd = Function("fd", "P(x4|x1, x2)")
fe = Function("fe", "P(x5|x2)")

x1 = Variable("x1")
x2 = Variable("x2")
x3 = Variable("x3")
x4 = Variable("x4")
x5 = Variable("x5")

# declaration of the edges
x5.addNeighbours([fe])
fe.addNeighbours([x2, x5])
fb.addNeighbours([x2])
x2.addNeighbours([fd, fe, fb])
fd.addNeighbours([x1, x2, x4])
x4.addNeighbours([fd])
x1.addNeighbours([fc, fa, fd])
fa.addNeighbours([x1])
fc.addNeighbours([x1, x3])
x3.addNeighbours([fc])

# compute probability
print(getProbability(x3, [x4]))
```

Figure 3: Création du graphe de facteurs, calcul $P(x3|x4)$

En retour, on obtiendra une chaîne de caractères représentant la formule pour obtenir la probabilité souhaitée. Il suffira ensuite de remplacer les probabilités par celles présentes dans les tables de probabilités du réseau bayésien et de dérouler les différentes sommes, afin d'obtenir une valeur numérique.

Pour obtenir ce résultat, j'ai créer 3 classes : `Node` la classe de base et 2 classes qui en dérivent : `Function` et `Variable`. Cela permet de facilement créer un graphe de facteurs. Chaque nœuds (Function ou Variables) pourront déclarer des voisins avec la méthode *addNeighbours()*.
Ensuite, les classes `Function` et `Variable` implémentent chacune une méthode *getProbability()* qui va simuler l'envoi des messages.
Chaque nœud appellera la méthode *getProbability()* de ces voisins et ainsi le comportement de l'algorithme **sum-produit** sera simulé. Cela revient plus ou moins à faire "remonter" des informations (ici des probabilités) des feuilles de l'arbre, jusqu'à un certain nœud racine.

Exemple 1
Pour le premier exemple, j'ai repris le graphe des facteurs du cours.

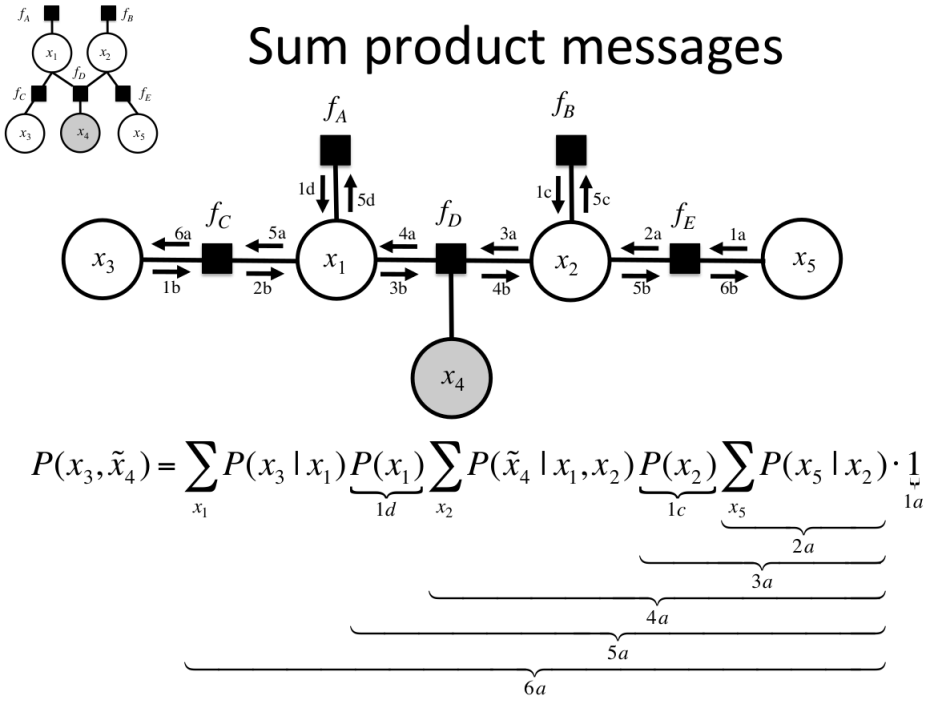

Figure 4: graphe de facteurs

Voici la déclarations du graphe de facteurs :

```python
# declaration of the factor graph
fa = Function("fa", "P(x1)")
fb = Function("fb", "P(x2)")
fc = Function("fc", "P(x3|x1)")
fd = Function("fd", "P(x4|x1, x2)")
fe = Function("fe", "P(x5|x2)")

x1 = Variable("x1")
x2 = Variable("x2")
x3 = Variable("x3")
x4 = Variable("x4")
x5 = Variable("x5")

# declaration of the edges
x5.addNeighbours([fe])
fe.addNeighbours([x2, x5])
fb.addNeighbours([x2])
x2.addNeighbours([fd, fe, fb])
fd.addNeighbours([x1, x2, x4])
x4.addNeighbours([fd])
x1.addNeighbours([fc, fa, fd])
fa.addNeighbours([x1])
fc.addNeighbours([x1, x3])
x3.addNeighbours([fc])

# compute probability
print(getProbability(x3, [x4]))
```

Figure 5: Création du graphe de facteurs

Je souhaite donc calculer la probabilité $P(x3|x4)$
Voici la formule renvoyée par mon programme, qui va permettre de calculer la probabilité de $P(x3|x4)$:

```
pierre@700-151SK:~/Dev/INF8225/TP1$ python3 main.py
 SOMME x1 P(x3|x1) P(x1) SOMME x2 P(x4|x1, x2) P(x2) SOMME x5 P(x5|x2) 1
```

Figure 6: résultat

Cela correspond à la formule suivante :
$P(x3|x4) = \sum_{x1} P(x3|x1)P(x1) \sum_{x2} P(x4|x1, x2)P(x2) \sum_{x5} P(x5|x2)1$

On retrouve donc bien la formule permettant le calcul de $P(x3|x4)$

Exemple 2

Dans un second temps, j'ai repris le réseau bayésien de la question 2.

Voici la déclaration du graphe de facteurs :

```
# declaration of the factor graph
fa = Function("fc", "P(C)")
fb = Function("ft", "P(T)")
fc = Function("fa", "P(A|C, T)")
fd = Function("fm", "P(M|A)")
fe = Function("fj", "P(J|A)")

x1 = Variable("C")
x2 = Variable("T")
x3 = Variable("A")
x4 = Variable("M")
x5 = Variable("J")

# declaration of the edges
x5.addNeighbours([fe])
fe.addNeighbours([x3, x5])
fb.addNeighbours([x2])
x2.addNeighbours([fb, fc])
fd.addNeighbours([x3, x4])
x4.addNeighbours([fd])
x1.addNeighbours([fc, fa])
fa.addNeighbours([x1])
fc.addNeighbours([x1, x2, x3])
x3.addNeighbours([fc, fd, fe])

# compute probability
print(getProbability(x5, []))
```

Figure 7: Création du graphe de facteurs

Pour cet exemple,

- x1 = Cambriolage
- x2 = Tremblement
- x3 = Alarme
- x4 = MarieAppelle
- x5 = JeanAppelle

Comme pour la question 2.d), je souhaite calculer $P(J)$
Voici le résultat :

```
pierre@700-15ISK:~/Dev/INF8225/TP1$ python3 main.py
 SOMME_A P(J|A)  SOMME_M P(M|A) 1 SOMME_C_T P(A|C, T) P(C)P(T)
```

Figure 8: résultat

Cela correspond à la formule suivante :
$P(J) = \sum_A P(J|A) \sum_M P(M|A) \sum_{CT} P(A|C,T)P(C)P(T)$
Cette formule est équivalente à celle trouvée à la question 2.d) :
$P(J) = \sum_C P(C) \sum_T P(T) \sum_A P(A|C,T)P(J|A)$

Il y a juste eu des simplifications ($\sum_M P(M|A) = 1$) et les sommes sont ordonnées différemment dans la seconde formule, mais les deux sont équivalentes. Dans les deux cas, on trouve 0.0117.
Il suffit ensuite de développer les sommes et de remplacer les probabilités par les valeurs présentes dans les tables de probabilités du réseau.

Si j'essaie de calculer la probabilité de $P(C|J = V)$, cela marche également. Il suffit d'utiliser la fonction *getProbability()* avec **x1** et **x5**, puisqu'il correspondent respectivement à C et J.

```
# compute probability
print(getProbability(x1, [x5]))
```

Figure 9: Appel à getProbability()

Voici le résultat :

```
pierre@700-15ISK:~/Dev/INF8225/TP1$ python3 main.py
 SOMME_A_T P(A|C, T)  SOMME_M P(M|A) 1 P(J|A) P(T)P(C)
```

Figure 10: résultat

Cela correspond à la formule suivante :
$P(C|J) = \sum_{AT} P(A|CT) \sum_M P(M|A)P(J|A)P(T)P(C)$
Cette formule est équivalente à celle trouvée à la question 2.d) :
$P(C|J) = \sum_T P(C)P(T) \sum_A P(A|C,T)P(J|A)$
A nouveau, il y a simplement eu des simplifications et les sommes sont ordonnées différemment dans la seconde formule. Cependant, les deux formules restent équivalentes. Dans les deux cas, on trouve 0.056.

Ainsi mon programme va simuler l'algorithme **sum-product** et l'envoi des différents

messages, ce qui permettra de calculer n'importe quelle probabilité dans un réseau bayésien. On obtiendra en sortie une formule que l'on peut facilement calculer, puisque toutes les probabilités présentes dans la formules sont présentes dans les tables de probabilités associées au réseau bayésien.