IFT 615 : Devoir 2
Travail individuel

Remise : 2 mars 2012, 12h20 (**au plus tard**)

1. **[2 points]** Soit le prédicat p, les fonctions f, g et h, les variables x, y et z puis les constantes a, b et c. Trouvez, s'il existe, l'unificateur le plus général (UPG) des clauses suivantes :
   (a) $\{p(f(y), a), p(y, x)\}$
   (b) $\{p(h(f(x), z), z), p(y, g(x))\}$
   (c) $\{p(a, b), p(c, x)\}$
   (d) $\{p(a, f(x), z, b), p(x, f(z), a, y)\}$

2. **[2 points]** Soit les symboles suivants :
   - $Emploi(p, o)$: prédicat indiquant que la personne p a comme profession o
   - $Client(p1, p2)$: prédicat indiquant que la personne $p1$ est un client de la personne $p2$
   - $Patron(p1, p2)$ prédicat indiquant que la personne $p1$ est le patron de $p2$
   - $Docteur$, $Chirurgien$, $Avocat$, $Acteur$: constantes désignant des professions
   - $Anne$, $Jean$: constantes désignant des personnes

   Supposez également l'existence d'un prédicat $Egale(x, y)$ qui est vrai seulement si les variables x et y correspondent à la même constante.

   Donnez une formule de logique du premier ordre décrivant chacune des assertions suivantes :
   (a) Anne est chirurgienne ou avocate.
   (b) Jean est un acteur, mais il a également un autre emploi.
   (c) Tous les chirurgiens sont également des docteurs.
   (d) Jean n'a pas d'avocat.
   (e) Le patron d'Anne est avocat.
   (f) Il y a un avocat dont tous les clients sont des docteurs.
   (g) Tous les chirurgiens ont un avocat.

3. **[2 points]** Soit les prédicats p, q et r, les variables x et y, puis les constantes a et b. Soit la base de connaissances suivante :
   (a) $\neg p(a, b)$
   (b) $\neg p(a, b) \vee q(b) \vee r(a)$
   (c) $\forall x \; \forall y \; q(x) \vee p(y, b) \vee r(a)$
   (d) $q(b) \vee p(a, b)$
   (e) $\neg q(a)$
   (f) $q(b) \vee p(a, a)$

   Utilisez la preuve par résolution pour prouver $\exists x \; r(x)$, et trouvez la valeur de x satisfaisant cette formule.

4. **[2 points]** Soit la distribution conjointe sur les variables aléatoires booléennes A, B, C et D :

		$A = false$		$A = true$	
		$B = false$	$B = true$	$B = false$	$B = true$
$C = false$	$D = false$	0.0054	0.0126	0.0216	0.0504
	$D = true$	0.0003	0.0007	0.0027	0.0063
$C = true$	$D = false$	0.3888	0.0972	0.2592	0.0648
	$D = true$	0.0360	0.0090	0.0360	0.0090

   (a) Calculez la probabilité marginale $P(B = true)$.

   (b) Calculez la probabilité conditionnelle $P(C = false|B = true)$.

   (c) Calculez la probabilité $P(C = true \vee D = true \vee A = false)$.

   (d) Est-ce que D et C sont indépendantes ?

   (e) Est-ce que B et A sont conditionnellement indépendantes, sachant C ?

   Suggestion : utilisez Python pour faire vos calculs.

5. **[2 points]** Soit la classe Python suivante, représentant un noeud dans un réseau bayésien :

```
class Noeud:

    def __init__(self, variable, domaine, parents,
                 p_variable_etant_donnes_parents):
        self.variable = variable
        self.domaine = domaine
        self.parents = parents
        self.p_variable_etant_donnes_parents = p_variable_etant_donnes_parents

    def sample(self, valeur_parents, rng):
        p = np.array([
                self.p_variable_etant_donnes_parents[(i,) + valeur_parents]
                for i in self.domaine ])
        return rng.multinomial(1,p).argmax()
```

   Dans cette classe, `__init__(self, variable, domaine, parents, p_variable_etant_donnes_parents)` est le constructeur. Il nécessite :

   - Le nom de la variable aléatoire : `variable` est une *string*.
   - Le domaine de cette variable : `domaine` est une liste d'entiers de 0 à `len(domaine) - 1`.
   - Les parents du noeud : `parents` est une liste de *string* correspondant aux parents.
   - La table de probabilités de la variable : `p_variable_etant_donnes_parents` est un tableau Numpy à `1 + len(parents)` dimensions, où la première correspond à la variable du noeud et les suivantes correspondent aux variables parents, dans l'ordre qu'elles apparaissent dans `parents`. Par exemple, pour le noeud ayant `'A'` comme variable et ayant les parents `['B','C']`, la probabilité $p(A = 2|B = 3, C = 0)$ est donnée par `p_variable_etant_donnes_parents[(2,3,0)]`.

   Le méthode `sample(self, valeur_parents, rng)` retourne une valeur échantillonnée de la variable du noeud (donc qui appartient au domaine de la variable), étant données les valeurs des parents. `valeur_parents` doit être un tuple, contenant la valeur des parents, dans l'ordre qu'elles apparaissent dans `parents`. `rng` est un objet générateur de nombres aléatoires, utilisé par la méthode `sample` (il n'est pas nécessaire de comprendre cet objet, il vous sera fourni).

Programmez une classe `ReseauBayesien` :

```
class ReseauBayesien:

    def __init__(self, noeuds):
        #Mettre code ici
        pass

    def p_exacte(self, variable_requete, observations):
        # Mettre code ici
        pass

    def p_echantillonnage_direct(self, variable_requete, observations,
                                 n_echantillons, rng):
        # Mettre code ici
        pass
```

définissant les méthodes suivantes :

- `__init__(self, noeuds)` : le constructeur prenant une liste d'objets `Noeud` qui correspondent aux noeuds du réseau bayésien.
- `p_exacte(self, variable_requete, observations)` : retourne le calcul exacte de la **distribution** de la variable aléatoire `variable_requete` étant données les observations dans le dictionnaire `observations`. Ce dictionnaire a comme clés des noms de variables, et comme valeur la valeur de chaque variable. Cette méthode doit donc retourner un vecteur sous forme de tableau Numpy contenant la probabilité de chaque valeur possible de la variable `variable_requete` (par exemple, $p(A = 0|B = 3, C = 0)$, $P(A = 1|B = 3, C = 0)$, $P(A = 2|B = 3, C = 0)$, etc.). Vous pouvez utiliser la méthode d'inférence par énumération, ou toute autre méthode exacte.
- `p_echantillonnage_direct(self, variable_requete, observations, n_echantillons, rng)` : retourne une estimation approximative de cette même distribution. Vous devez absolument utiliser la **méthode de rejet** vue en classe. `n_echantillons` donne le nombre d'échantillons à générer pour l'estimation, `rng` est le générateur de nombres aléatoires à utiliser.

Votre implémentation de cette classe doit être placée dans un fichier `solution.py`, qui sera importé lors de la procédure automatique de correction.

Le fichier `devoir_2.py` contient un exemple d'exécution de votre code. Vous pouvez l'appeler comme un script. Les probabilités calculées par le script correspondent aux exemples vus dans le cours. À noter que l'évaluation ne se fera pas sur la base de ces exemples. Ainsi, n'hésitez pas à générer vous-mêmes d'autres exemples de réseaux bayésiens, pour tester votre code.

La correction se fera de façon automatique. Ainsi, dans le cas de la méthode de rejet, il est important d'utiliser le générateur de nombres aléatoires passé en argument. Veuillez soumettre votre fichier `solution.py` via l'outil **turnin**, avant la date de remise. À noter que **tout autre fichier soumis sera ignoré** : votre implémentation doit être entièrement comprise dans le fichier `solution.py`.

6. [**BONUS**] Les 3 étudiants ayant remis une implémentation de l'inférence exacte **correcte** et **plus rapide** que celle des autres étudiants recevra **1 point bonus**. L'évaluation du temps de calcul incluera la construction du réseau bayésien ainsi que l'appel de la fonction `p_exacte` à quelques reprises.