3i005 - projet 2 - 2019fev - *version définitive*

CLASSIFICATIONS PROBABILISTES

Le but de ce projet est d'étudier les méthodes de classifications auxquelles les statistiques donnent accès naturellement. Il s'agira donc d'étudier les outils de classifications probabilistes. C'est bien évidemment une petite partie de l'ensemble des méthodes de classification existant.

Evaluation du projet

L'ensemble des codes que vous réaliserez seront accessible à partir du ficher `projet.py` . L'évaluation de votre code se fera à l'aide de données autres que celles du projet dans un programme qui commencera par importer votre `projet.py` . **il faudra donc une vigilance particulière à respecter les noms et la signature des classes, fonctions et méthodes ! Le code dans ce notebook ne doit pas être modifié, votre code dans `projet.py` doit permettre d'exécuter ce notebook et d'avoir les mêmes résultats que ceux de la version originale**.

Si des questions méritent des réponses ouvertes, ces réponses seront insérées dans ce notebook dans une cellule immédiatement en dessous de celle contenant la question et nulle part ailleurs (la cellule existe et contient le texte `(votre réponse ici)`).

WARNING: le notebook n'est pas statique et évoluera avec des questions supplémentaires durant le projet. Ne répondez donc pas directement aux questions ouvertes dans le notebook qui sera à écraser chaque semaine avec la nouvelle version (incrémentale). Répondez temporairement dans un fichier à part (par exemple dans des copies du notebook). Vous compléterez le notebook à la fin de la troisième semaine avant le rendu final.

Une attention soutenue sera demandée à la documentation de votre code et à sa qualité ainsi qu'à la qualité des réponses ouvertes dans ce notebook.

Base utilisé : heart disease (Cleveland database)

```
    This database contains 76 attributes, but all published experiments refer to using a subset of
14 of them. In particular, the Cleveland database is the only one that has been used by ML researc
hers to this date.
    The `target` field refers to the presence of heart disease in the patient. It is integer value
d from 0 (no presence) to 4. Experiments with the Cleveland database have concentrated on simply a
ttempting to distinguish presence (values 1) from absence (value 0).
```

champs	definition
age	age in years
sex	(1 = male; 0 = female)
cp	chest pain type
trestbps	resting blood pressure (in mm Hg on admission to the hospital)
chol	serum cholestoral in mg/dl
fbs	(fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
restecg	resting electrocardiographic results
thalach	maximum heart rate achieved
exang	exercise induced angina (1 = yes; 0 = no)
oldpeak	ST depression induced by exercise relative to rest
slope	the slope of the peak exercise ST segment
ca	number of major vessels (0-3) colored by flourosopy
thal	3 = normal; 6 = fixed defect; 7 = reversable defect
target	1 or 0

Notre but est donc de proposer des classifieurs qui tentent de prédire la valeur de `target` à partir des autres champs en utilisant des arguments probabilistes.

Simplification de la base (prélude au projet)

In [1]:
```
import pandas as pd # package for high-performance, easy-to-use data structures and data ana
lysis
import numpy as np # fundamental package for scientific computing with Python
```

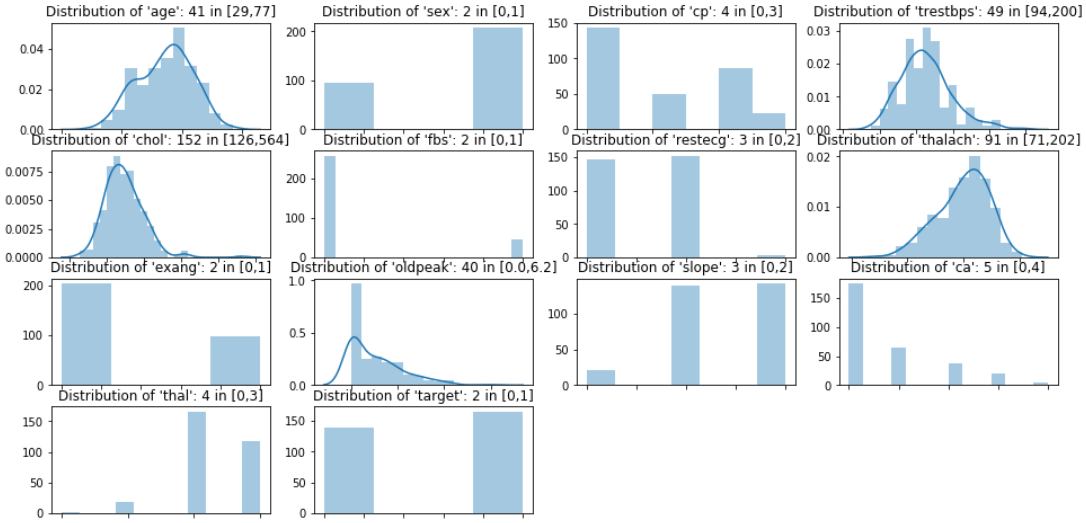
In [2]:
```
# to allow autoreload of projet.py
%load_ext autoreload
%autoreload 2

import utils # deux fonctions dans le fichier utils.py
import projet # votre code
```

In [3]:
```
data=pd.read_csv("heart.csv")
data.head()
```

Out[3]:

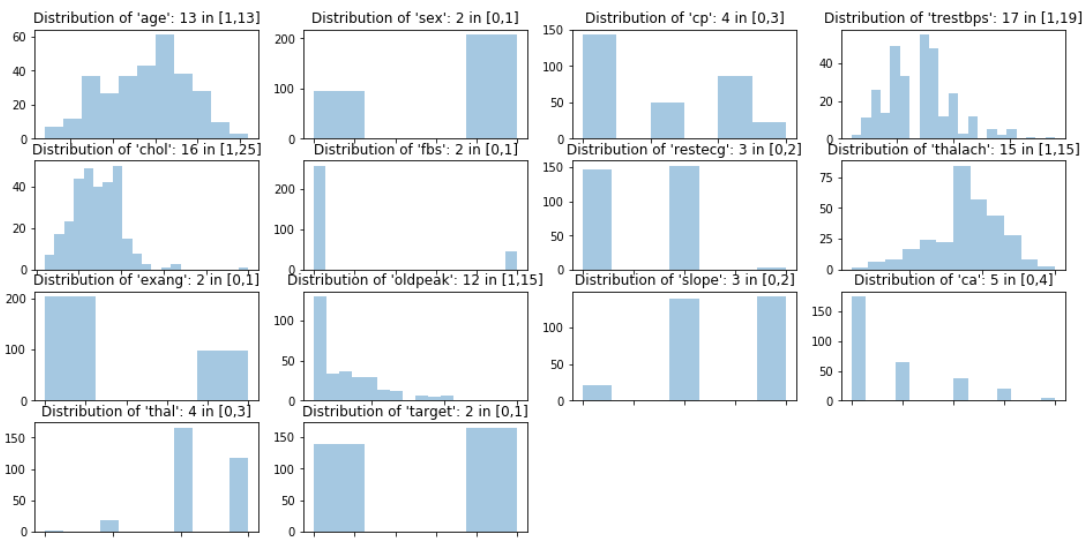
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

In [4]:
```
utils.viewData(data)
```



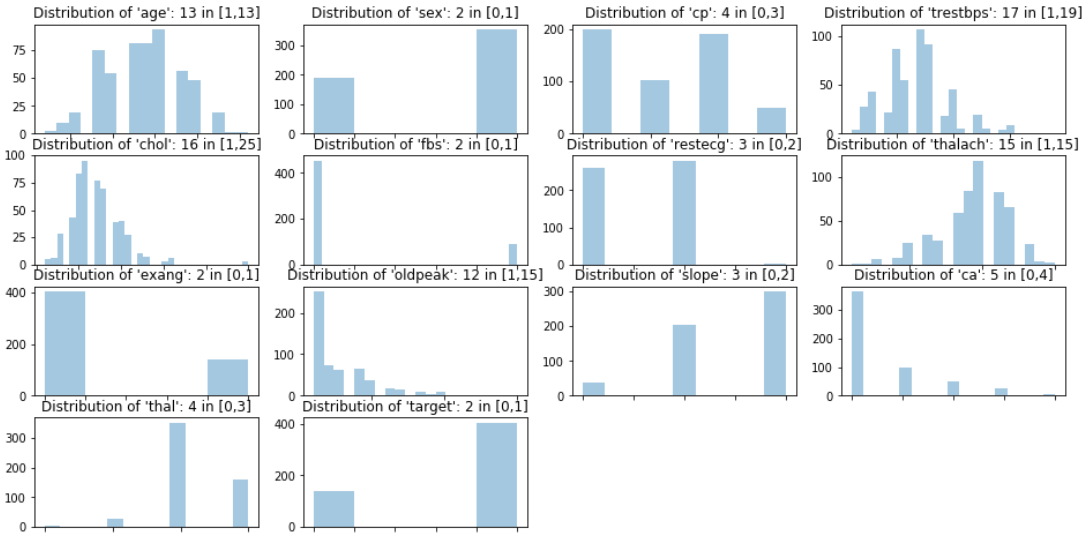
Certaines variabless comme `age` , `thalach` , etc. possèdent un grand nombre de modalités, rendant difficile le traitement. Nous simplifions donc la base en discrétisant au mieux toutes les variables qui ont plus de 5 valeurs.

In [5]:
```
discretise=utils.discretizeData(data)
utils.viewData(discretise,kde=False)
```



Nous utilisons maintenant 2 fichiers csv préparés à partir de cette base afin de rendre les résultats plus intéressants (en particulier, les 2 classes sont un peu plus déséquilibrées).

In [6]:
```
train=pd.read_csv("train.csv")
test=pd.read_csv("test.csv")
utils.viewData(train,kde=False)
```



À partir de maintenant, nous utilisons le dataframe `train` qui contient les données pour l'apprentissage et `test` qui contient les données pour la validation.

Classification a priori

Question 1

Dans une fonction `getPrior` , calculer la probabilité a priori de la classe 1 ainsi que l'intervalle de confiance à 95% pour l'estimation de cette probabilité.

In [7]:
```
projet.getPrior(train) # ou project.getPrior("train.csv")
```

Out[7]:
```
{'estimation': 0.7453874538745388,
 'min5pourcent': 0.7087109975695709,
 'max5pourcent': 0.7820639101795066}
```

Question 2

On propose une classe permettant de représenter un classifieur. Un classifieur répond à une question principale : étant donné un individu, connu par ses attributs, quelle est sa classe ? Nous proposons donc une classe simple qu'il s'agira d'améliorer : `AbstractClassifier` dans le fichier `utils.py`

Question 2a

Ecrire dans `projet.py` un classifieur `APrioriClassifier` (enfant de `AbstractClassifier`) qui utilise le résultat de la question 1 pour estimer très simplement la classe de chaque individu par la classe majoritaire.

In [8]:
```
cl=projet.APrioriClassifier()
clpredite=cl.estimClass(None) # n'importe quoi donne la même classe pour un classifieur a pr
iori
# la valeur prédite n'est pas affichée sciemment
```

Question 2b : évaluation de classifieurs

Implémenter également la méthode `statsOnDF` qui rendra les valeurs suivantes :

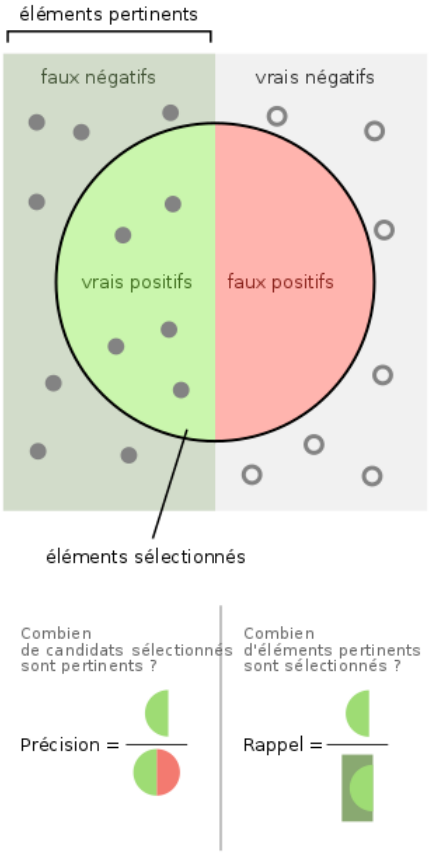
- VP : nombre d'individus avec target=1 et classe prévue=1
- VN : nombre d'individus avec target=0 et classe prévue=0
- FP : nombre d'individus avec target=0 et classe prévue=1
- FN : nombre d'individus avec target=1 et classe prévue=0
- précision
- rappel

- *Petite aide : comment itérer sur un dataframe*

```
for t in train.itertuples():
    dic=t._asdict()
    print("ca={} oldpeak={} target=
{}".format(dic['ca'],dic['oldpeak'],dic['target']))
```

- *Par ailleurs, dans utils, il y a une fonction `getNthDict(df,n)` qui rend le dictionnaire des attributs de la $n$ième ligne dans `df`.*

```
getNthDict(train,0)
{'age': 9, 'sex': 1, 'cp': 3, 'trestbps': 9, 'chol': 6,
'fbs': 1, 'restecg': 0, 'thalach': 9, 'exang': 0, 'oldpeak':
6, 'slope': 0, 'ca': 0, 'thal': 1, 'target': 1}
```



In [9]:
```
cl=projet.APrioriClassifier()
print("test en apprentissage : {}".format(cl.statsOnDF(train)))
print("test en validation: {}".format(cl.statsOnDF(test)))
```

```
test en apprentissage : {'VP': 404, 'VN': 0, 'FP': 138, 'FN': 0, 'Précision': 0.745387453874
5388, 'Rappel': 1.0}
test en validation: {'VP': 138, 'VN': 0, 'FP': 62, 'FN': 0, 'Précision': 0.69, 'Rappel': 1.0
}
```

Question 3 : classification probabiliste à 2 dimensions

La classification a priori ne donne pas d'excellents résultats puiqu'elle se contente de la règle majoritaire. On se propose donc maintenant essayer d'enrichir notre processus de décision en tenant compte d'une caractéristique de la base de données.

Question 3.a : probabilités conditionelles

Écrire une fonction `P2D_l(df,attr)` qui calcule dans le dataframe la probabilité $P(attr|target)$ sous la forme d'un dictionnaire asssociant à la valeur t un dictionnaire associant à la valeur a la probabilité $P(attr = a|target = t)$.

In [10]:
```
projet.P2D_l(train,'thal')
```

Out[10]:
```
{1: {1: 0.03217821782178218,
  2: 0.7821782178217822,
  3: 0.1782178217821782,
  0: 0.007425742574257425},
 0: {1: 0.08695652173913043,
  2: 0.2608695652173913,
  3: 0.644927536231884,
  0: 0.007246376811594203}}
```

Écrire une fonction `P2D_p(df,attr)` qui calcule dans le dataframe la probabilité $P(target|attr)$ sous la forme d'un dictionnaire associant à la valeur a un dictionnaire asssociant à la valeur t la probabilité $P(target = t|attr = a)$.

In [11]:
```
projet.P2D_p(train,'thal')
```

Out[11]:
```
{1: {1: 0.52, 0: 0.48},
 2: {1: 0.8977272727272727, 0: 0.10227272727272728},
 3: {1: 0.4472049689440994, 0: 0.5527950310559007},
 0: {1: 0.75, 0: 0.25}}
```

Question 3.b : classifieurs 2D par maximum de vraisemblance

Supposons qu'un individu ait la valeur a pour l'$attr$, un classifieur du type $P2D_l$ pourrrait donc utiliser $P(attr = a|target = t)$ et sélectionner comme estimation de la classe de l'individu la valeur $t = 0$ ou $t = 1$ maximisant cette probabilité. $P(attr = a|target)$ est la vraisemblance d'observer $attr = a$ quand $target = 0$ ou $target = 1$. Un tel classifieur utilise donc le principe du **maximum de vraisemblance** (ML=Max Likelihood).

Pour construire un tel classifieur, il faut initialiser l'attribut utilisé puis construire la table `P2Dl` . La fonction `estimClass` rendra la position du maximum trouvé dans cette table.

> Supposons un individu dont $thal = 3$, alors dans la table P2Dl, on trouve 0.178 pour $target = 1$ et 0.644 pour $target = 0$, la bonne classe d'après le critère du ML est donc 0

Écrire une classe `ML2DClassifier` qui utilise une telle procédure de maximum de vraisemblance pour estimer la classe d'un individu. Afin de ne pas avoir à réécrire la méthode `statsOnDF` qui ne devrait pas changer, `ML2DClassifier` aura pour parent la classe `APrioriClassifier` .

PS- penser bien à calculer une seule fois la table `P2Dl` dans le constructeur de la classe afin de ne pas itérer sur toute la base à chaque fois que vous appelez la méthode `estimClass` .

PS2- Dans les cas d'égalité des 2 probabilités, on choisira la classe 0.

In [12]:
```
cl=projet.ML2DClassifier(train,"thal") # cette ligne appelle projet.P2Dl(train,"thal")
for i in [0,1,2]:
    print("Estimation de la classe de l'individu {} par ML2DClassifier : {}".format(i,cl.est
imClass(utils.getNthDict(train,i))))
```

```
Estimation de la classe de l'individu 0 par ML2DClassifier : 0
Estimation de la classe de l'individu 1 par ML2DClassifier : 1
Estimation de la classe de l'individu 2 par ML2DClassifier : 1
```

In [13]:
```
print("test en apprentissage : {}".format(cl.statsOnDF(train)))
print("test en validation: {}".format(cl.statsOnDF(test)))
```

```
test en apprentissage : {'VP': 319, 'VN': 101, 'FP': 37, 'FN': 85, 'Précision': 0.8960674157
303371, 'Rappel': 0.7896039603960396}
test en validation: {'VP': 113, 'VN': 48, 'FP': 14, 'FN': 25, 'Précision': 0.889763779527559
, 'Rappel': 0.8188405797101449}
```

Question 3.c : classifieurs 2D par maximum a posteriori

Supposons qu'un individu ait la valeur a pour l'$attr$, un classifieur du type $P2D_p$ pourrrait donc utiliser $P(target = t|attr = a)$ et sélectionner comme estimation de la classe de l'individu la valeur $t = 0$ ou $t = 1$ maximisant cette probabilité. $P(target|attr = a)$ est la distribution a posteriori de $target$ après avoir observé $attr = a$. Un tel classifieur utilise donc le principe du **maximum a posteriori** (MAP).

Pour construire un tel classifieur, il faut initialiser l'attribut utilisé puis construire la table `P2Dp` . La fonction `estimClass` rendra la position du maximum trouvé dans cette table.

> Supposons un individu dont $thal = 3$, alors dans la table P2Dp, on trouve 0.447 pour $target = 1$ et 0.552 pour $target = 0$, la bonne classe d'après le critère du ML est donc 0

Écrire une classe `MAP2DClassifier` qui utilise une telle procédure de maximum de vraisemblance pour estimer la classe d'un individu. Afin de ne pas avoir à réécrire la méthode `statsOnDF` qui ne devrait pas changer, `MAP2DClassifier` héritera de `AprioriClassifier` .

PS- penser bien à calculer une seule fois la table `P2Dp` dans le constructeur afin de ne pas itérer sur toute la base à chaque fois que vous appelez la méthode `estimClass` .

PS2- Dans les cas d'égalité des 2 probabilités, on choisira la classe 0.

In [14]:
```
cl=projet.MAP2DClassifier(train,"thal") # cette ligne appelle projet.P2Dp(train,"thal")
for i in [0,1,2]:
    print("Estimation de la classe de l'individu {} par MAP2DClasssifer) : {}".format(i,cl.e
stimClass(utils.getNthDict(train,i))))
```

```
Estimation de la classe de l'individu 0 par MAP2DClasssifer) : 1
Estimation de la classe de l'individu 1 par MAP2DClasssifer) : 1
Estimation de la classe de l'individu 2 par MAP2DClasssifer) : 1
```

In [15]:
```
print("test en apprentissage : {}".format(cl.statsOnDF(train)))
print("test en validation: {}".format(cl.statsOnDF(test)))
```

```
test en apprentissage : {'VP': 332, 'VN': 89, 'FP': 49, 'FN': 72, 'Précision': 0.87139107611
54856, 'Rappel': 0.8217821782178217}
test en validation: {'VP': 114, 'VN': 43, 'FP': 19, 'FN': 24, 'Précision': 0.857142857142857
1, 'Rappel': 0.8260869565217391}
```

Question 3.d - comparaison

Quelle classifieur préférez-vous en théorie entre `APrioriClassifier` , `ML2DClassifier` et `MAP2DClassifier` ? Quels résultats vous semble-les plus intéressants ?

(ici votre réponse)

Question 4

On peut bien sûr se dire que les classifieurs ont intérêt à utiliser le plus d'information possible. Il serait donc pertinent de construire les classifieurs `ML3DClassifier` , `MAP3DClassifier` , ..., `ML14DClassifier` et `MAP14DClassifier` où les "x `DClassifier` " prendrait $x-1$ attributs pour construire la prédiction de `target` . Toutefois, un problème va nous arrêter : les tables $P14Da$ et $P14Db$ sont de bien trop grande taille.

Question 4.1 : complexité en mémoire

Écrire une fonction `nbrParams` qui calcule la taille mémoire de ces tables $P(target|attr_1,..,attr_k)$ étant donné un dataframe et la liste $[target, attr_1, \ldots, attr_l]$ en supposant qu'un float est représenté sur 8octets.

In [16]:
```
projet.nbParams(train,['target'])
projet.nbParams(train,['target','thal'])
projet.nbParams(train,['target','age'])
projet.nbParams(train,['target','age','thal','sex','exang'])
projet.nbParams(train,['target','age','thal','sex','exang','slope','ca','chol'])
projet.nbParams(train)
```

```
1 variable(s) : 16 octets
2 variable(s) : 64 octets
2 variable(s) : 208 octets
5 variable(s) : 3328 octets = 3ko 256o
8 variable(s) : 798720 octets = 780ko 0o
14 variable(s) : 58657996800 octets = 54go 644mo 640ko 0o
```

On ne peut donc pas manipuler de telles tables et il faut trouver de nouvelles façon de représenter les distributions de probabilités, quitte à en faire des approximations.

La meilleure façon de simplifier la représentation d'une distribution de probabilité est d'utiliser des hypothèses d'indépendances. Ainsi, dans une loi jointe des variables A, B, C, D, E, si on suppose l'indépendance de ces 5 variables, on sait qu'on pourra écrire que

$$P(A,B,C,D,E) = P(A) * P(B) * P(C) * P(D) * P(E)$$

et donc remplacer un tableau à 5 dimensions par 5 tableaux monodimensionnels.

Question 4.2 : complexité en mémoire sous hypothèse d'indépendance complète

Ecrire une fonction `nbrParamsIndep` qui calcule la taille mémoire nécessaire pour représenter les tables de probabilité étant donné un dataframe, en supposant qu'un float est représenté sur 8octets et **en supposant l'indépendance des variables**.

In [17]:
```
projet.nbParamsIndep(train[['target']])
projet.nbParamsIndep(train[['target','thal']])
projet.nbParamsIndep(train[['target','age']])
projet.nbParamsIndep(train[['target','age','thal','sex','exang']])
projet.nbParamsIndep(train[['target','age','thal','sex','exang','slope','ca','chol']])
projet.nbParamsIndep(train)
```

```
1 variable(s) : 16 octets
2 variable(s) : 48 octets
2 variable(s) : 120 octets
5 variable(s) : 184 octets
8 variable(s) : 376 octets
14 variable(s) : 800 octets
```

Question 4.3

L'indépendance complète comme ci-dessus amène forcément à un classifier a priori (aucun attribut n'apporte d'information sur `target`).

Nous allons donc essayer de trouver des modèles supposant une certaine forme d'indépendance partielle qui permettra d'alléger quand même la représentation en mémoire de la distribution de probabilités. Ce sont les indépendances conditionnelles. Si l'on sait par exemple que A est indépendant de C sachant B, on peut écrire la loi jointe :

$$P(A,B,C) = P(A) * P(B|A) * P(C|B)$$

Pouvez vous le prouvez ?

(votre réponse ici)

Si les 3 variables A, B et C ont 5 valeurs, quelle est la taille mémoire en octet nécessaire pour représenter cette distribution avec et sans l'utilisation de l'indépendance conditionnelle ?

(votre réponse ici)

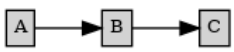
Question 5 : Modèles graphiques

Afin de représenter efficacement les indépendances conditionnelles utilisées pour représenter une distribution jointe de grande taille, on peut utiliser un graphe orienté qui se lit ainsi : dans la décomposition de la loi jointe, chaque variable X apparaitra dans un facteur de la forme $P(X|Parents_X)$. On note que cette factorisation n'a de sens que si le graphe n'a pas de circuit (c'est un DAG).

Ainsi, on représente la factorisation $P(A,B,C) = P(A) * P(B|A) * P(C|B)$ par le graphe suivant : A n'a pas de parent, B a pour parent A et C a pour parent B.

In [18]:
```
utils.drawGraphHorizontal("A->B;B->C")
```

Out[18]:

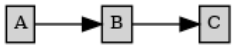


Question 5.1

Dans les 2 cellules suivantes, dessiner les graphes pour 5 variables A, B, C, D, E complètement indépendantes puis pour ces 5 même variables sans aucune indépendance.

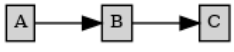
In [19]:
```
utils.drawGraphHorizontal("A->B->C") # changer la chaîne pour représenter une indépendance complète entre A,B,C,D,E
```

Out[19]:



In [20]:
```
utils.drawGraphHorizontal("A->B->C") # changer la chaîne pour représenter une dépendance complète entre A,B,C,D,E-
```

Out[20]:



Question 5.2 : naïve Bayes

Un modèle simple souvent utilisée est le **Naïve Bayes**. Il suppose que 2 attributs sont toujours indépendants conditionnellement à `target` .

Ce modèle est évidemment très simpliste et certainement faux. Toutefois, en classification, il donne souvent de bon résultats.

Écrire comment se décompose la vraisemblance $P(attr1, attr2, attr3, \cdots | target)$.

Écrire comment se décompole la distribution a posteriori $P(target|attr1, attr2, attr3, \cdots)$ *(ou du moins une fonction proportionnelle à cette distribution a posteriori).*

(votre réponse ici)

$$P(attr1, attr2, attr3, \cdots | target) = \ldots$$
$$P(target|attr1, attr2, attr3, \cdots) = \ldots$$

Question 5.3 : modèle graphique et naïve bayes

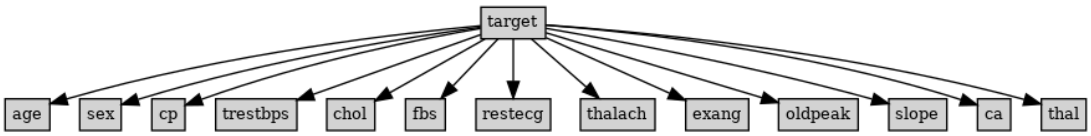
Un modèle naïve bayes se représente sous la forme d'un graphe où le noeud `target` est l'unique parent de tous les attributs. Construire une fonction `drawNaiveBayes` qui a partir d'un dataframe et du nom de la colonne qui est la classe, dessine le graphe.

Note : une fonction qui dessine un graphe retourne un appel à `utils.drawGraph` , par exemple :

```
def dessin_A_vers_B():
    return utils.drawGraph("A->B")
```

In [21]:
```
projet.drawNaiveBayes(train,"target")
```

Out[21]:



Écrire une fonction `nbrParamsNaiveBayes` qui calcule la taille mémoire nécessaire pour représenter les tables de probabilité étant donné un dataframe, en supposant qu'un float est représenté sur 8octets et **en utilisant l'hypothèse du Naive Bayes**.

In [22]:
```
projet.nbParamsNaiveBayes(train,'target',[])
projet.nbParamsNaiveBayes(train,'target',['target','thal'])
projet.nbParamsNaiveBayes(train,'target',['target','age'])
projet.nbParamsNaiveBayes(train,'target',['target','age','thal','sex','exang'])
projet.nbParamsNaiveBayes(train,'target',['target','age','thal','sex','exang','slope','ca','
chol'])
projet.nbParamsNaiveBayes(train,'target')
```

```
0 variable(s) : 16 octets
2 variable(s) : 80 octets
2 variable(s) : 224 octets
5 variable(s) : 352 octets
8 variable(s) : 736 octets
14 variable(s) : 1584 octets = 1ko 560o
```

On voit que l'augmentation de la mémoire nécessaire est raisonnable.

Question 5.4 : classifier naïve bayes

Afin de ne pas avoir à réécrire la méthode `statsOnDF` qui ne devrait pas changer, écrire les classes `MLNaiveBayesClassifier` et `MAPNaiveBayesClassifier` qui hérite de `AprioriClassifier` et qui utilise le maximum de vraisemblance (ML) et le maximum a posteriori (MAP) pour estimer la classe d'un individu en utilisant l'hypothèse du Naïve Bayes.

De la même façon que plus haut, penser à calculer tous les paramètres du Naïve Bayes dans le constructeur de la classe afin de ne pas les recalculer pour chaque classification.

Décomposer la méthodes `estimClass` en 2 parties : `estimProbas` qui calcule la vraisemblance et `estimClass` qui utilise `estimProbas` pour choisir la classe comme dans les classifieurs précédents.

In [23]:
```
cl=projet.MLNaiveBayesClassifier(train)
for i in [0,1,2]:
    print("Estimation de la proba de l'individu {} par MLNaiveBayesClassifier : {}".format(i
,cl.estimProbas(utils.getNthDict(train,i))))
    print("Estimation de la classe de l'individu {} par MLNaiveBayesClassifier : {}".format(
i,cl.estimClass(utils.getNthDict(train,i))))
print("test en apprentissage : {}".format(cl.statsOnDF(train)))
print("test en validation: {}".format(cl.statsOnDF(test)))
```

```
Estimation de la proba de l'individu 0 par MLNaiveBayesClassifier : {0: 5.265474022893807e-1
1, 1: 8.7794388463561 87e-12}
Estimation de la classe de l'individu 0 par MLNaiveBayesClassifier : 0
Estimation de la proba de l'individu 1 par MLNaiveBayesClassifier : {0: 0.0, 1: 1.9903404816
168002e-09}
Estimation de la classe de l'individu 1 par MLNaiveBayesClassifier : 1
Estimation de la proba de l'individu 2 par MLNaiveBayesClassifier : {0: 3.6835223975945704e-
10, 1: 1.5920340255297033e-06}
Estimation de la classe de l'individu 2 par MLNaiveBayesClassifier : 1
test en apprentissage : {'VP': 350, 'VN': 116, 'FP': 22, 'FN': 54, 'Précision': 0.9408602150
537635, 'Rappel': 0.8663366336633663}
test en validation: {'VP': 49, 'VN': 60, 'FP': 2, 'FN': 89, 'Précision': 0.9607843137254902,
'Rappel': 0.35507246376811596}
```

In [24]:
```
cl=projet.MAPNaiveBayesClassifier(train)
for i in [0,1,2]:
    print("Estimation de la proba de l'individu {} par MAPNaiveBayesClassifier : {}".format(
i,cl.estimProbas(utils.getNthDict(train,i))))
    print("Estimation de la classe de l'individu {} par MAPNaiveBayesClassifier : {}".format
(i,cl.estimClass(utils.getNthDict(train,i))))
print("test en apprentissage : {}".format(cl.statsOnDF(train)))
print("test en validation: {}".format(cl.statsOnDF(test)))
```

```
Estimation de la proba de l'individu 0 par MAPNaiveBayesClassifier : {0: 0.6719863008964105,
1: 0.32801369910358946}
Estimation de la classe de l'individu 0 par MAPNaiveBayesClassifier : 0
Estimation de la proba de l'individu 1 par MAPNaiveBayesClassifier : {0: 0.0, 1: 1.0}
Estimation de la classe de l'individu 1 par MAPNaiveBayesClassifier : 1
Estimation de la proba de l'individu 2 par MAPNaiveBayesClassifier : {0: 7.90267948988375e-0
5, 1: 0.9999209732051012}
Estimation de la classe de l'individu 2 par MAPNaiveBayesClassifier : 1
test en apprentissage : {'VP': 382, 'VN': 111, 'FP': 27, 'FN': 22, 'Précision': 0.9339853300
733496, 'Rappel': 0.9455445544554455}
test en validation: {'VP': 53, 'VN': 57, 'FP': 5, 'FN': 85, 'Précision': 0.9137931034482759,
'Rappel': 0.38405797101449274}
```

Question 6 : *feature selection* dans le cadre du classifier naive bayes

Il est possible qu'un attribut de la base ne soit pas important pour estimer la classe d'un individu. Dans le cadre du Naïve Bayes, un tel noeud se reconnaît car il est indépendant de `target`. Un tel noeud peut être supprimé du Naïve Bayes.

Écrire une fonction `isIndepFromTarget(df,attr,x)` qui vérifie si `attr` est indépendant de `target` au seuil de x%.

Note : vous avez le droit d'utiliser `scipy.stats.chi2_contingency` dans cette fonction.

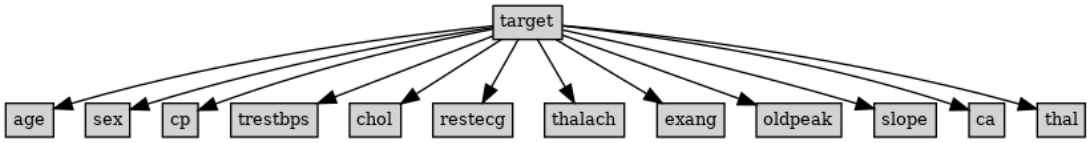
In [25]:
```
for attr in train.keys():
    if attr!='target':
        print("target independant de {} ? {}".format(attr,"**YES**" if projet.isIndepFromTar
get(train,attr,0.01) else "no"))
```

```
target independant de age ? no
target independant de sex ? no
target independant de cp ? no
target independant de trestbps ? **YES**
target independant de chol ? no
target independant de fbs ? **YES**
target independant de restecg ? no
target independant de thalach ? no
target independant de exang ? no
target independant de oldpeak ? no
target independant de slope ? no
target independant de ca ? no
target independant de thal ? no
```

Proposer des classifieurs `ReducedMLNaiveBayesClassifier` et `ReducedMAPNaiveBayesClassifier` qui utilisent le maximum de vraisemblance (ML) et le maximum a posteriori (MAP) pour estimer la classe d'un individu sur un modèle Naïve Bayes qu'ils auront préalablement optimisé grâce à des tests d'indépendance au seuil de $x\%$ (donné en paramètre du constructeur). Rajouter une méthode `ReducedMAPNaiveBayesClassifier.draw` afin de pouvoir dessiner le Naïve Bayes réduit effectivement utilisé.

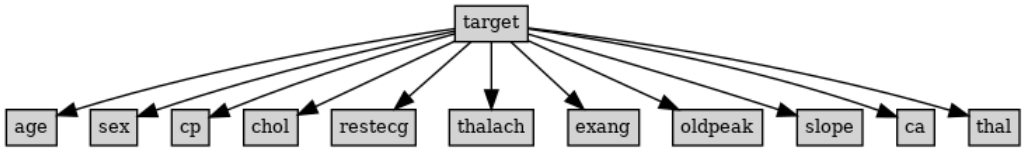
In [26]:
```
cl=projet.ReducedMLNaiveBayesClassifier(train,0.05)
cl.draw()
```

Out[26]:



In [27]:
```
cl=projet.ReducedMLNaiveBayesClassifier(train,0.01)
cl.draw()
```

Out[27]:

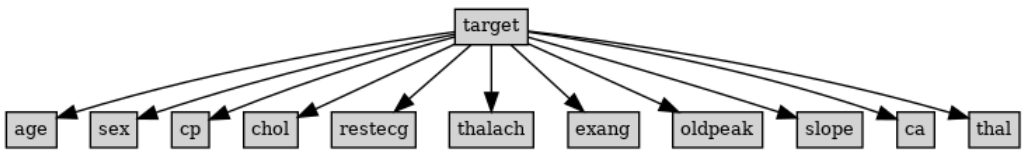


In [28]:
```
for i in [0,1,2]:
    print("Estimation de la proba de l'individu {} par MAPNaiveBayesClassifier : {}".format(
i,cl.estimProbas(utils.getNthDict(train,i))))
    print("Estimation de la classe de l'individu {} par MAPNaiveBayesClassifier : {}".format
(i,cl.estimClass(utils.getNthDict(train,i))))
print("test en apprentissage : {}".format(cl.statsOnDF(train)))
print("test en validation: {}".format(cl.statsOnDF(test)))
```

```
Estimation de la proba de l'individu 0 par MAPNaiveBayesClassifier : {0: 5.697482232499412e-
09, 1: 2.138723717532643e-09}
Estimation de la classe de l'individu 0 par MAPNaiveBayesClassifier : 0
Estimation de la proba de l'individu 1 par MAPNaiveBayesClassifier : {0: 0.0, 1: 1.147574579
7921708e-08}
Estimation de la classe de l'individu 1 par MAPNaiveBayesClassifier : 1
Estimation de la proba de l'individu 2 par MAPNaiveBayesClassifier : {0: 2.7487852876093664e
-09, 1: 9.179222322695213e-06}
Estimation de la classe de l'individu 2 par MAPNaiveBayesClassifier : 1
test en apprentissage : {'VP': 348, 'VN': 117, 'FP': 21, 'FN': 56, 'Précision': 0.9430894308
94309, 'Rappel': 0.8613861386138614}
test en validation: {'VP': 49, 'VN': 61, 'FP': 1, 'FN': 89, 'Précision': 0.98, 'Rappel': 0.3
5507246376811596}
```

In [29]:
```
cl=projet.ReducedMAPNaiveBayesClassifier(train,0.01)
cl.draw()
```

Out[29]:



In [30]:
```
for i in [0,1,2]:
    print("Estimation de la proba de l'individu {} par MAPNaiveBayesClassifier : {}".format(
i,cl.estimProbas(utils.getNthDict(train,i))))
    print("Estimation de la classe de l'individu {} par MAPNaiveBayesClassifier : {}".format
(i,cl.estimClass(utils.getNthDict(train,i))))
print("test en apprentissage : {}".format(cl.statsOnDF(train)))
print("test en validation: {}".format(cl.statsOnDF(test)))
```

```
Estimation de la proba de l'individu 0 par MAPNaiveBayesClassifier : {0: 0.47643095845795086
, 1: 0.5235690415420491}
Estimation de la classe de l'individu 0 par MAPNaiveBayesClassifier : 1
Estimation de la proba de l'individu 1 par MAPNaiveBayesClassifier : {0: 0.0, 1: 1.0}
Estimation de la classe de l'individu 1 par MAPNaiveBayesClassifier : 1
Estimation de la proba de l'individu 2 par MAPNaiveBayesClassifier : {0: 0.00010227941341238
206, 1: 0.9998977205865877}
Estimation de la classe de l'individu 2 par MAPNaiveBayesClassifier : 1
test en apprentissage : {'VP': 375, 'VN': 110, 'FP': 28, 'FN': 29, 'Précision': 0.9305210918
114144, 'Rappel': 0.9282178217821783}
test en validation: {'VP': 53, 'VN': 56, 'FP': 6, 'FN': 85, 'Précision': 0.8983050847457628,
'Rappel': 0.38405797101449274}
```

Question 7 : évaluation des classifieurs

Nous commençons à avoir pas mal de classifieurs. Pour les comparer, une possibilité est d'utiliser la représentation graphique des points $(précision, rappel)$ de chacun.

Question 7.1

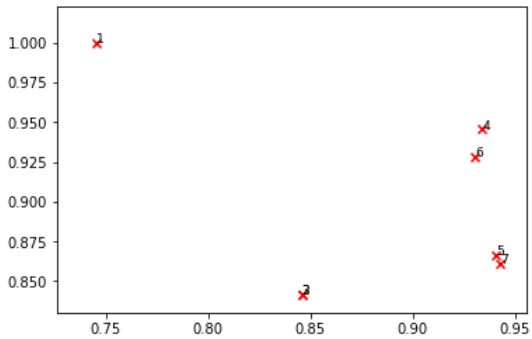
Où se trouve à votre avis le point idéal ? Comment pourriez-vous proposer de comparer les différents classifieurs dans cette représentation graphique ?

(ici votre réponse)

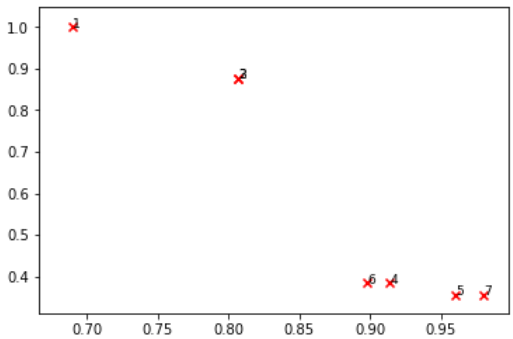
Question 7.2

Ecrire une fonction `mapClassifiers(dic,df)` qui, à partir d'un dictionnaire `dic` de `{nom:instance de classifier}` et d'un dataframe `df` , représente graphiquement ces classifiers dans l'espace $(précision, rappel)$.

In [31]:
```
projet.mapClassifiers({"1":projet.APrioriClassifier(),
                       "2":projet.ML2DClassifier(train,"exang"),
                       "3":projet.MAP2DClassifier(train,"exang"),
                       "4":projet.MAPNaiveBayesClassifier(train),
                       "5":projet.MLNaiveBayesClassifier(train),
                       "6":projet.ReducedMAPNaiveBayesClassifier(train,0.01),
                       "7":projet.ReducedMLNaiveBayesClassifier(train,0.01),
                      },train)
```



In [32]:
```
projet.mapClassifiers({"1":projet.APrioriClassifier(),
                       "2":projet.ML2DClassifier(train,"exang"),
                       "3":projet.MAP2DClassifier(train,"exang"),
                       "4":projet.MAPNaiveBayesClassifier(train),
                       "5":projet.MLNaiveBayesClassifier(train),
                       "6":projet.ReducedMAPNaiveBayesClassifier(train,0.01),
                       "7":projet.ReducedMLNaiveBayesClassifier(train,0.01),
                      },test)
```



Question 8 : Sophistication du modèle (question BONUS)

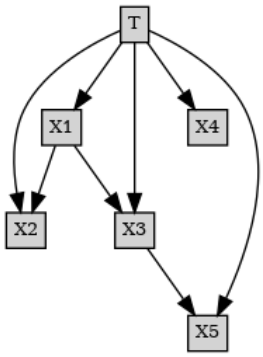
Utiliser un arbre pour représenter la factorisation de la loi jointe est bien sûr une simplification : beaucoup de distribution ne peuvent être représentées avec un seul parent par variable.

Un modèle plus sophisitiqué existe donc : le TAN (Tree-augmented Naïve Bayes). Il consiste à rajouter au plus un parent à chaque attribut parmi les autres attributs (sans créer de cycle). En plus des arcs les reliant à la classe, un TABN induit donc un arbre (plus exactement une forêt) parmi les attributs.

Ci-dessous un TAN dont la classe est T.

```
In [33]: utils.drawGraph("T->X1;T->X2;T->X3;T->X4;T->X5;X1->X2;X1->X3;X3->X5")
```

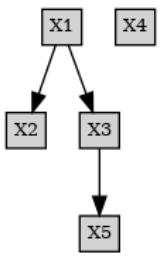
Out[33]:



et dont l'arbre (la forêt) sur les attributs est bien :

```
In [34]: utils.drawGraph("X1->X2;X1->X3;X3->X5;X4")
```

Out[34]:



L'algorithme pour générer cette structure se base sur une autre façon de tester l'indépendance entre deux variables aléatoires : l'information mutuelle qui calcule une distance entre la distribution des 2 variables et la distribution si ces 2 variables étaient indépendantes (voir https://fr.wikipedia.org/wiki/Information_mutuelle (https://fr.wikipedia.org/wiki/Information_mutuelle)). Pour construire l'arbre (la forêt) entre les attributs, sachant qu'on garde les arcs issus de la classe, il faut tester des indépendances conditionnelles et donc calculer des informations mutuelles conditionnelles (https://en.wikipedia.org/wiki/Conditional_mutual_information (https://en.wikipedia.org/wiki/Conditional_mutual_information)).

On gardera de ces pages les deux formules :

$$I(X;Y) = \sum_{x,y} P(x,y) \log_2 \frac{P(x,y)}{P(x)P(y)}$$

$$I(X;Y|Z) = \sum_{z,x,y} P(x,y,z) \log_2 \frac{P(z)P(x,y,z)}{P(x,z)P(y,z)}$$

Et on remarquera que :

- $P(x,z) = \sum_y P(x,y,z)$,
- $P(y,z) = \sum_x P(x,y,z)$,
- $P(z) = \sum_{x,y} P(x,y,z)$,
- etc.

Question 8.1 : calcul des informations mutuelles

Écrire des fonctions `projet.MutualInformation(df,x,y)` et `projet.ConditionalMutualInformation(df,x,y,z)` qui calcule ces informations mutuelles

In [35]:
```
for attr in train.keys():
    if attr!='target':
        print("{}->{} : {}".format("target",attr,projet.MutualInformation(train,"target",attr)))
```

```
target->age : 0.059090666566536484
target->sex : 0.0359445359672019
target->cp : 0.15995396264990075
target->trestbps : 0.04119801216101553
target->chol : 0.040582433305734356
target->fbs : 4.13455968783088555e-05
target->restecg : 0.01613920867246313
target->thalach : 0.14015721668140788
target->exang : 0.10148366175826332
target->oldpeak : 0.13935734517832749
target->slope : 0.0938837825561079
target->ca : 0.14051038130632754
target->thal : 0.16255361669359983
```

(On retrouve au passage que trestbs et surtout fbs sont très peu dépendantes de la classe ...)

In [36]:
```
cmis=np.array([[0 if x==y else projet.ConditionalMutualInformation(train,x,y,"target")
                for x in train.keys() if x!="target"]
               for y in train.keys() if y!="target"])
cmis[0:5,0:5]
```

Out[36]:
```
array([[0.        , 0.07172827, 0.20250622, 0.6417183 , 0.55128095],
       [0.07172827, 0.        , 0.01672381, 0.06361231, 0.12140024],
       [0.20250622, 0.01672381, 0.        , 0.22413205, 0.18890573],
       [0.6417183 , 0.06361231, 0.22413205, 0.        , 0.68149423],
       [0.55128095, 0.12140024, 0.18890573, 0.68149423, 0.        ]])
```

(on remarque que, évidemment, la matrice `cmis` est symétrique)

Question 8.2 : calcul de la matrice des poids

La matrice `cmis` calculé ci-dessus représente l'ensemble des arcs possibles entre les attributs et leur poids. Pour trouver un arbre dans ces arcs, on commence par simplifier cette matrice en supprimant les poids faibles. Par exemple, en retirant la moyenne.

Faites une fonction `projet.MeanForSymetricWeights(a)` qui calcule la moyenne des poids pour une matrice `a` symétrique de diagonale nulle.

Puis écrire une fonction `projet.simplifyContitionalMutualInformationMatrix(a)` qui annule toutes les valeurs plus petites que cette moyenne dans une matrice `a` symétrique de diagonale nulle.

In [37]:
```
projet.MeanForSymetricWeights(cmis)
```

Out[37]:
```
0.14490408192274776
```

In [38]:
```
projet.SimplifyConditionalMutualInformationMatrix(cmis)
cmis[0:5,0:5]
```

Out[38]:
```
array([[0.        , 0.        , 0.20250622, 0.6417183 , 0.55128095],
       [0.        , 0.        , 0.        , 0.        , 0.        ],
       [0.20250622, 0.        , 0.        , 0.22413205, 0.18890573],
       [0.6417183 , 0.        , 0.22413205, 0.        , 0.68149423],
       [0.55128095, 0.        , 0.18890573, 0.68149423, 0.        ]])
```

Question 8.3 : Arbre (forêt) optimal entre les attributs

Un algorithme pour trouver un arbre de poids maximal est l'algorithm de Kruskal (https://fr.wikipedia.org/wiki/Algorithme_de_Kruskal (https://fr.wikipedia.org/wiki/Algorithme_de_Kruskal)). En se souvenant qu'on veut relier les attributs si ils sont très dépendants, écrire la fonction `projet.Kruskal(df,a)` qui propose la liste des arcs (non orientés pour l'instant) à ajouter dans notre classifieur sous la forme d'une liste de triplet $(attr1, attr2, poids)$.

Remarque : `df` ne sert ici qu'à retrouver le nom des attributs à partir de leur indice grâce à `train.keys()[i]` .

In [39]:
```
liste_arcs=projet.Kruskal(train,cmis)
liste_arcs
```

Out[39]:
```
[('trestbps', 'chol', 0.6814942282235203),
 ('age', 'trestbps', 0.641718295908513),
 ('age', 'thalach', 0.6365766485465845),
 ('chol', 'oldpeak', 0.5246930555244587),
 ('oldpeak', 'slope', 0.25839871090530614),
 ('chol', 'ca', 0.2528327956181666)]
```

Question 8.4: Orientation des arcs entre attributs.

Il s'agit maintenant d'orienter l'arbre (la forêt) entre les attributs. On choisit la (ou les) racine(s) en maximisant l'information mutuelle entre ces attributs et la classe (donc en utilisant `projet.MutualInformation` .

Créer une fonction `projet.ConnexSet(list_arcs)` qui rend une liste d'ensemble d'attributs connectés,

In [40]:
```
# 3 arcs de poids 1 dans le graphe a--b--c   d--e
projet.ConnexSets([('a','b',1),
                   ('a','c',1),
                   ('d','e',1)])
```

Out[40]:
```
[{'a', 'b', 'c'}, {'d', 'e'}]
```

In [41]:
```
projet.ConnexSets(liste_arcs)
```

Out[41]:
```
[{'age', 'ca', 'chol', 'oldpeak', 'slope', 'thalach', 'trestbps'}]
```

Puis écrire une fonction `projet.OrientConnexSets(df,arcs,classe)` qui utilise l'information mutuelle (entre chaque attribut et la `classe`) pour proposer pour chaque ensemble d'attributs connexes une racine et qui rend la liste des arcs orientés.

In [42]:
```
projet.OrientConnexSets(train,liste_arcs,'target')
```

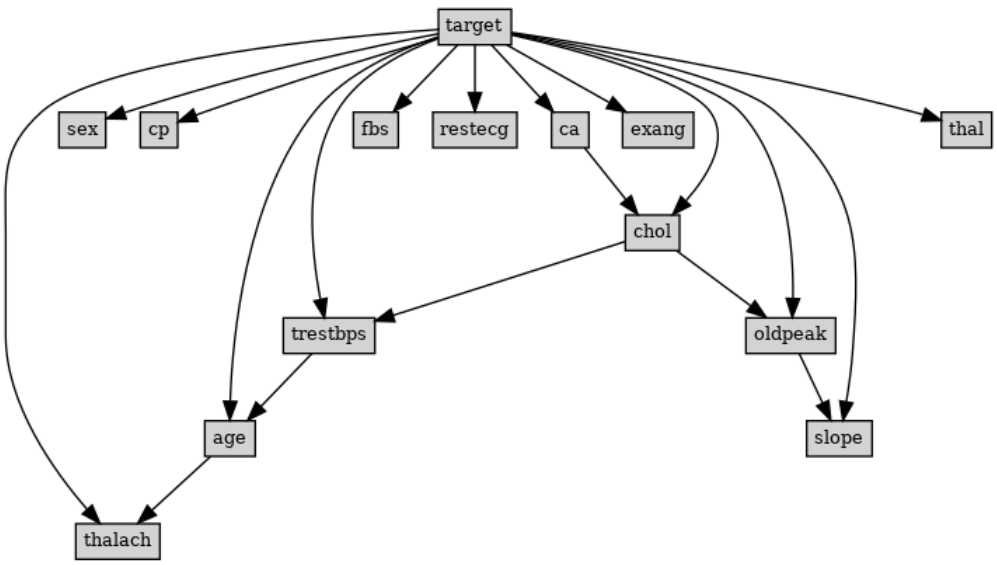
Out[42]:
```
[('ca', 'chol'),
 ('chol', 'trestbps'),
 ('trestbps', 'age'),
 ('age', 'thalach'),
 ('chol', 'oldpeak'),
 ('oldpeak', 'slope')]
```

Question 8.5: Classifieur TAN

Écrire un `MAPTANClassifier(df)` qui construit un modèle TAN en suivant la procédure ci-dessus. Lui ajouter une procédure `Draw()`

In [43]:
```
tan=projet.MAPTANClassifier(train)
tan.draw()
```

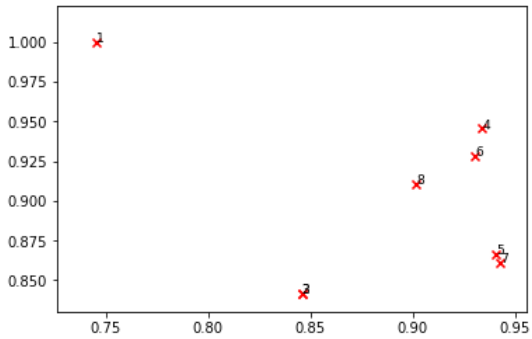
Out[43]:



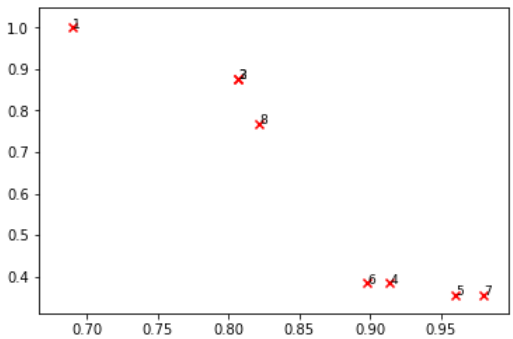
In [44]:
```
for i in [0,1,2]:
    print("Estimation de la proba de l'individu {} par MAPTANClassifier : {}".format(i,tan.estimProbas(utils.getNthDict(train,i))))
    print("Estimation de la classe de l'individu {} par MAPTANClassifier : {}".format(i,tan.estimClass(utils.getNthDict(train,i))))
print("test en apprentissage : {}".format(tan.statsOnDF(train)))
print("test en validation: {}".format(tan.statsOnDF(test)))
```

```
Estimation de la proba de l'individu 0 par MAPTANClassifier : {0: 0.37723376588463897, 1: 0.622766234115361}
Estimation de la classe de l'individu 0 par MAPTANClassifier : 1
Estimation de la proba de l'individu 1 par MAPTANClassifier : {0: 9.584367095154832e-06, 1: 0.9999904156329049}
Estimation de la classe de l'individu 1 par MAPTANClassifier : 1
Estimation de la proba de l'individu 2 par MAPTANClassifier : {0: 0.028638687699112764, 1: 0.9713613123008872}
Estimation de la classe de l'individu 2 par MAPTANClassifier : 1
test en apprentissage : {'VP': 330, 'VN': 97, 'FP': 41, 'FN': 74, 'Précision': 0.889487870619946, 'Rappel': 0.8168316831683168}
test en validation: {'VP': 104, 'VN': 41, 'FP': 21, 'FN': 34, 'Précision': 0.832, 'Rappel': 0.7536231884057971}
```

In [45]:
```
projet.mapClassifiers({"1":projet.APrioriClassifier(),
                       "2":projet.ML2DClassifier(train,"exang"),
                       "3":projet.MAP2DClassifier(train,"exang"),
                       "4":projet.MAPNaiveBayesClassifier(train),
                       "5":projet.MLNaiveBayesClassifier(train),
                       "6":projet.ReducedMAPNaiveBayesClassifier(train,0.01),
                       "7":projet.ReducedMLNaiveBayesClassifier(train,0.01),
                       "8":projet.MAPTANClassifier(train),
                      },train)
```



In [46]:
```
projet.mapClassifiers({"1":projet.APrioriClassifier(),
                       "2":projet.ML2DClassifier(train,"exang"),
                       "3":projet.MAP2DClassifier(train,"exang"),
                       "4":projet.MAPNaiveBayesClassifier(train),
                       "5":projet.MLNaiveBayesClassifier(train),
                       "6":projet.ReducedMAPNaiveBayesClassifier(train,0.01),
                       "7":projet.ReducedMLNaiveBayesClassifier(train,0.01),
                       "8":projet.MAPTANClassifier(train),
                      },test)
```



Question 9

Quelle leçons & conclusion tirez-vous de ces expériences sur les classifieurs bayésiens ?

In [ ]: