

# 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.

## A LIRE ATTENTIVEMENT : Évaluation du projet

L'ensemble des codes que vous réaliserez seront écrits dans le fichier `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 et le format des réponses ouvertes !
- Le 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 pdf \*\*.
- le fichier `projet.py` doit commencer par une entête qui aura cette forme (dans les premières lignes) :

---

```
001 # prenom1 nom1
002 # prenom2 nom2
```

---

- Si des questions méritent des réponses ouvertes, ces réponses seront insérées dans `projet.py` en commentaire avec le titre de la question et nulle par ailleurs. Par exemple :



TITRE DE LA QUESTION-EXEMPLE

Indique qu'il faudra introduire dans `projet.py` un commentaire qui aura cette forme :

---

```
143 #####
144 # TITRE DE LA QUESTION-EXEMPLE
145 #####
146 # ceci est ma réponse à la question-exemple de l'énoncé.
146 # ...
147 #####
```

---

Bien noter la position des 5 `#` : avant le titre, après le titre et à la fin de la question.

Vous pouvez mettre ces réponses où vous voulez. Par exemple à la fin ou au début du fichier `projet.py`.

- Une attention soutenue sera demandée à la documentation de votre code et à sa qualité ainsi qu'à la qualité des réponses ouvertes. Le fichier `utils.py` contient des exemples

corrects de documentation de classes et de fonctions:

```
class AbstractClassifier:
    """
    Un classifieur implémente un algorithme pour estimer la classe d'un vecteur d'attributs. Il propose aussi comme service
    de calculer les statistiques de reconnaissance à partir d'un pandas.dataframe.
    """
    # CLASSE BIEN DOCUMENTEE

    def __init__(self):
        pass

    def estimClass(self, attrs):
        """
        à partir d'un dictionnaire d'attributs, estime la classe 0 ou 1

        Parameters
        -----
            attrs: Dict[str,value]
                le dictionnaire nom-valeur des attributs

        Returns
        -----
            la classe 0 ou 1 estimée
        """
    # FONCTION BIEN DOCUMENTEE
```

- Enfin, tous nos classifieurs seront codés dans des classes, sous-classes de `AbstractClassifier` qui se trouve dans le fichier `utils.py`. Pour rappel de la programmation orientée objet en python 3, par exemple : <https://realpython.com/python3-object-oriented-programming/#dog-park-example>.
- Vous soumettrez un unique `projet.py`. **Toute autre type de soumission sera considérée comme invalide !**

```
In [1]: # Afin de vous simplifier la vie : à chaque modification de projet.py, il sera rechargé par la session jupyter.
%load_ext autoreload
%autoreload 2

# utils.py contient des fonctions et des classes d'aide au projet. IL NE FAUT RIEN CHANGER DANS CE FICHIER
import utils

# Cette ligne importe vos codes. VOTRE CODE EST A ÉCRIRE DANS projet.py.
import projet # votre code
```

## Base utilisée : 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 researchers to this date.

The `target` field refers to the presence of heart disease in the patient. It is integer valued from 0 (no presence) to 4. Experiments with the Cleveland database have concentrated on simply attempting 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.

## 0- Simplification de la base

**prélude au projet : pas de travail à faire dans cette partie**

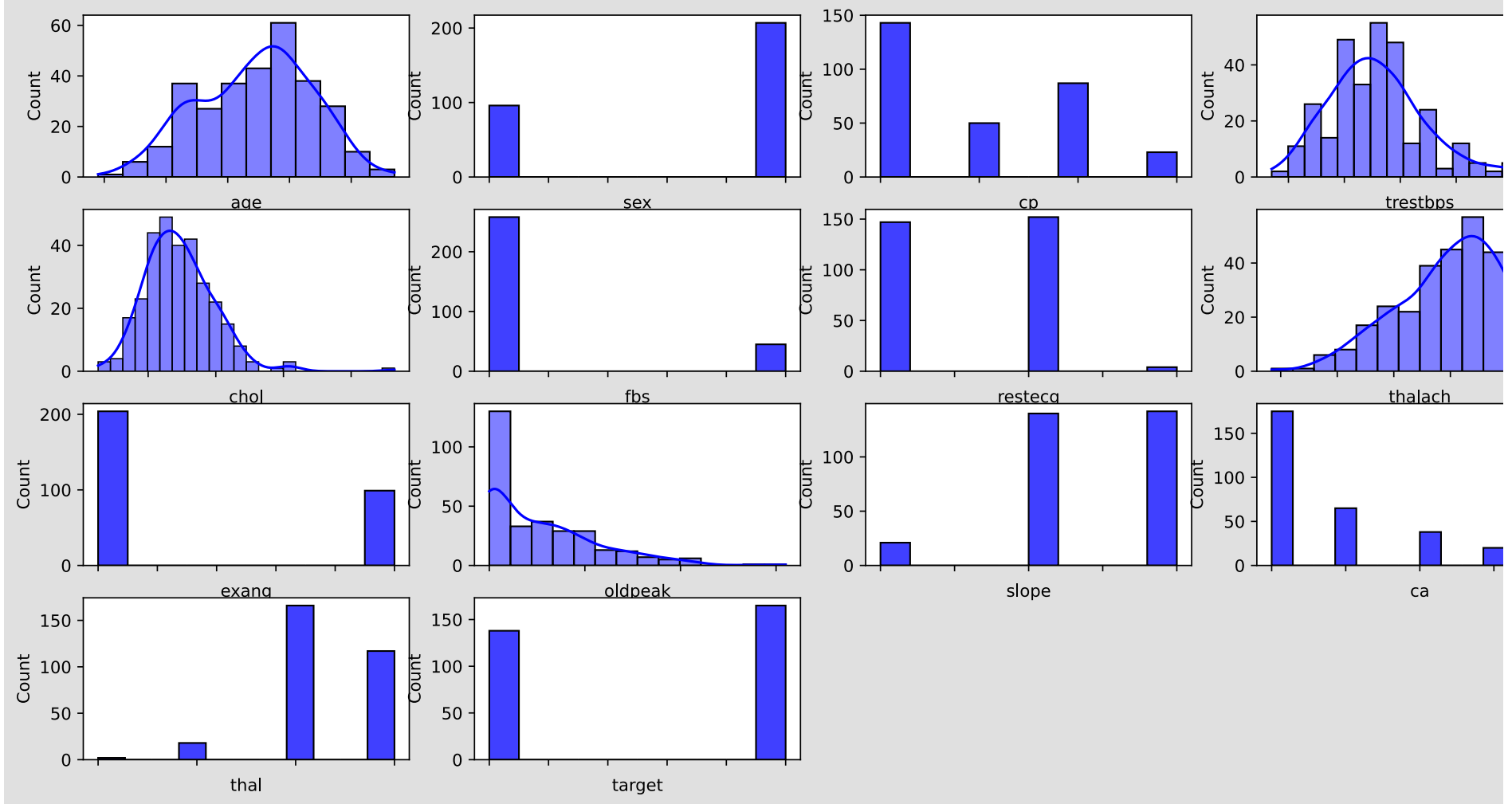
```
In [2]: import pandas as pd # package for high-performance, easy-to-use data structures and data analysis
import numpy as np # fundamental package for scientific computing with Python
```

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

```
Out[3]:
```

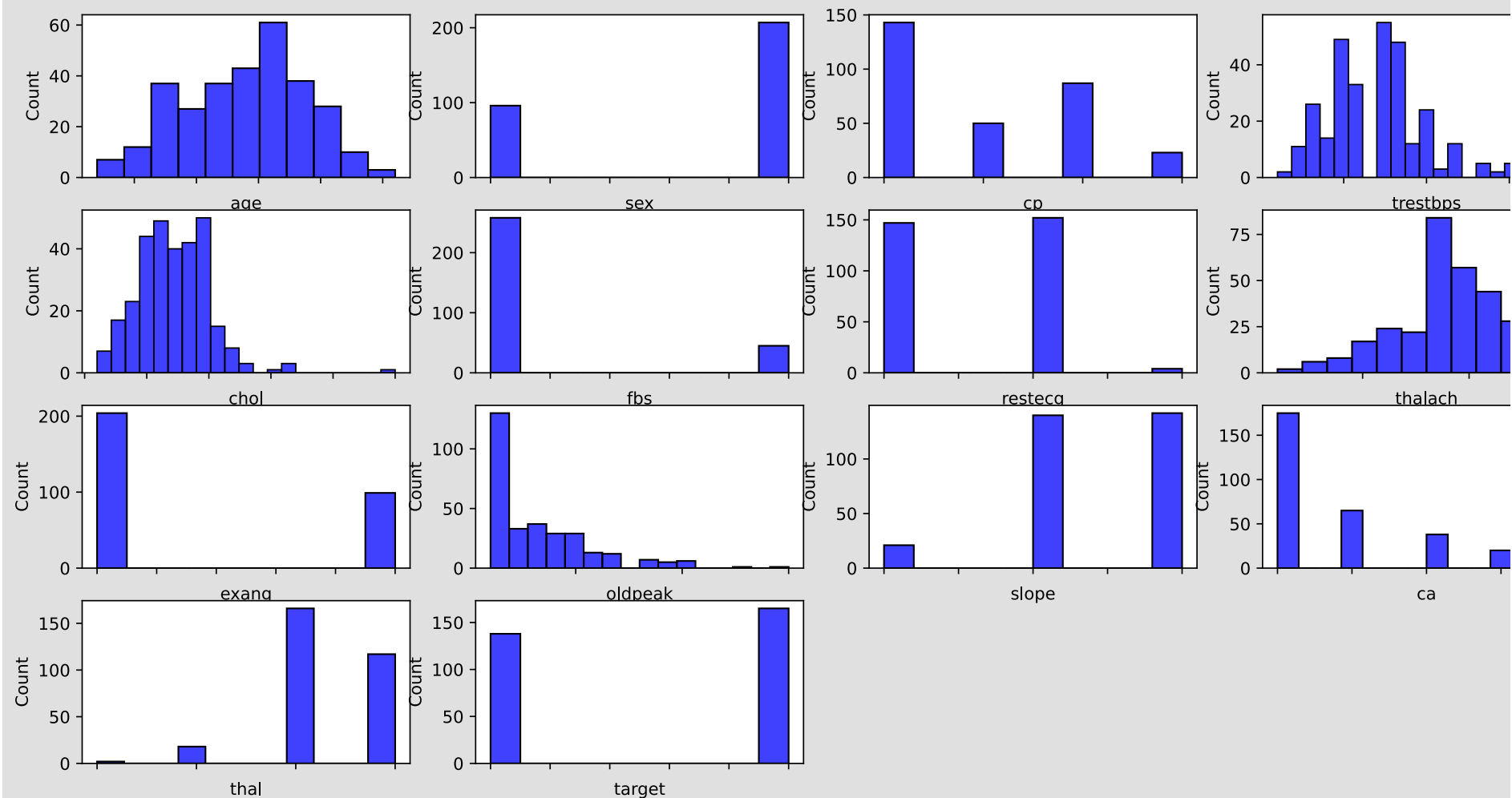
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

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



Certaines variables 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.

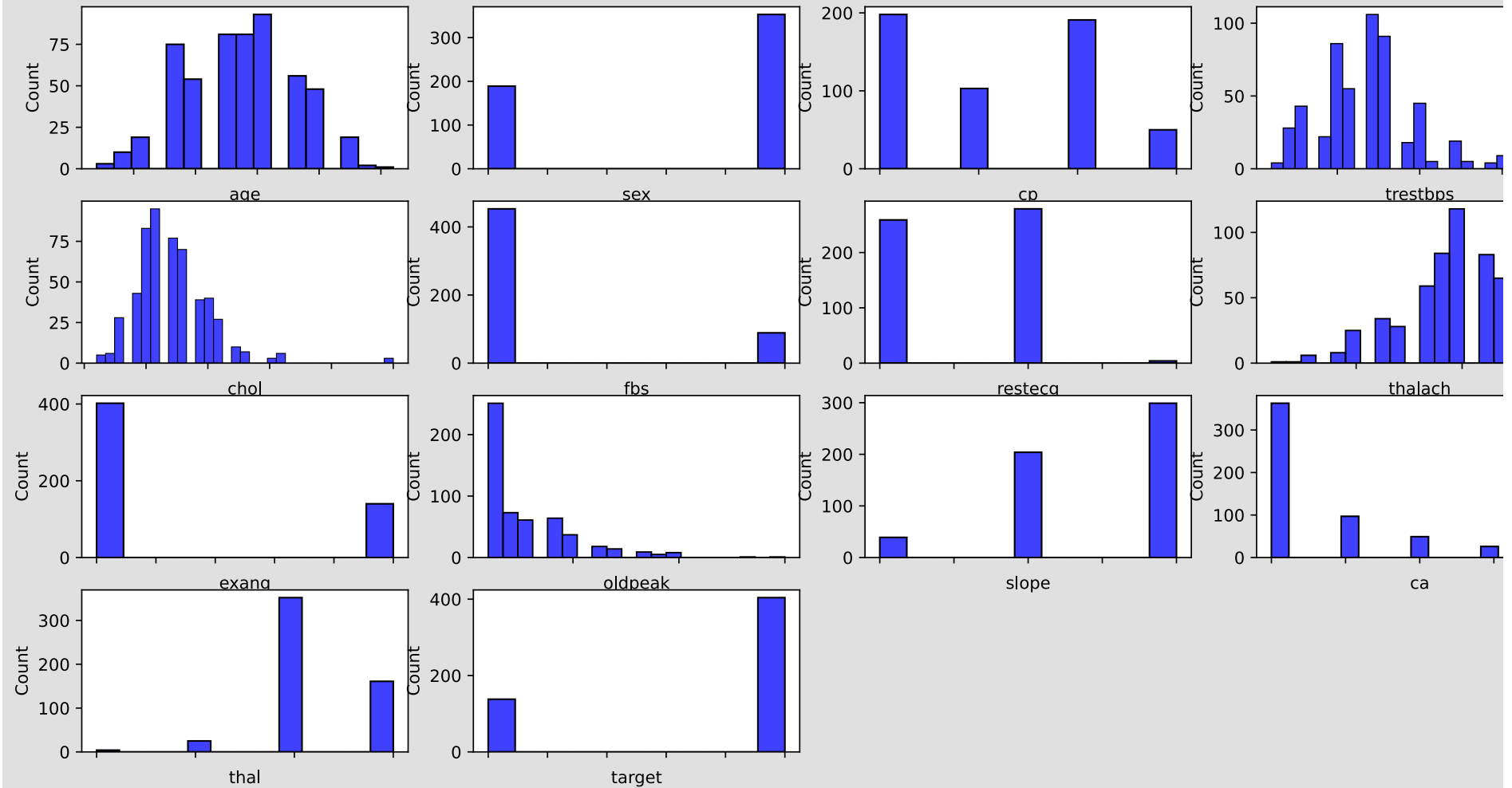
```
In [5]: discretise=utils.discretizeData(data)
utils.viewData(discretise,kde=False)
```



Nous utilisons maintenant 2 fichiers csv **préparés à l'avance pour ce projet** 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("data/train.csv")
test=pd.read_csv("data/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.

# 1- Classification a priori

## Question 1.1 : calcul de la probabilité a priori

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]: # cette fonction doit donc rendre un dictionnaire contenant 3 clés 'estimation', 'min5pourcent', 'max5pourcent' (L'ORDRE DES CLES N'EST PAS IMPORTANTE)
projet.getPrior(train)
```

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

```
In [8]: projet.getPrior(test)
```

```
Out[8]: {'estimation': 0.69,
        'min5pourcent': 0.62590170673099,
        'max5pourcent': 0.7540982932690099}
```

## Question 1.2 : programmation orientée objet dans la hiérarchie des `Classifier`

On propose de représenter les classifieurs en python par des classes d'une hiérarchie. Un classifieur répond à une question principale : étant donné un vecteur d'attributs, quelle est la classe proposée ? Nous proposons donc une classe de base qu'il s'agira d'**améliorer et de spécialiser en la sous-classant** : `AbstractClassifier` dans le fichier `utils.py`

Telle qu'elle est définie dans `utils.py`, la classe `AbstractClassifier` ressemble à une *interface* (en Java) : elle décrit sans implémenter les deux méthodes que doivent implémenter les `Classifier` que vous allez écrire.

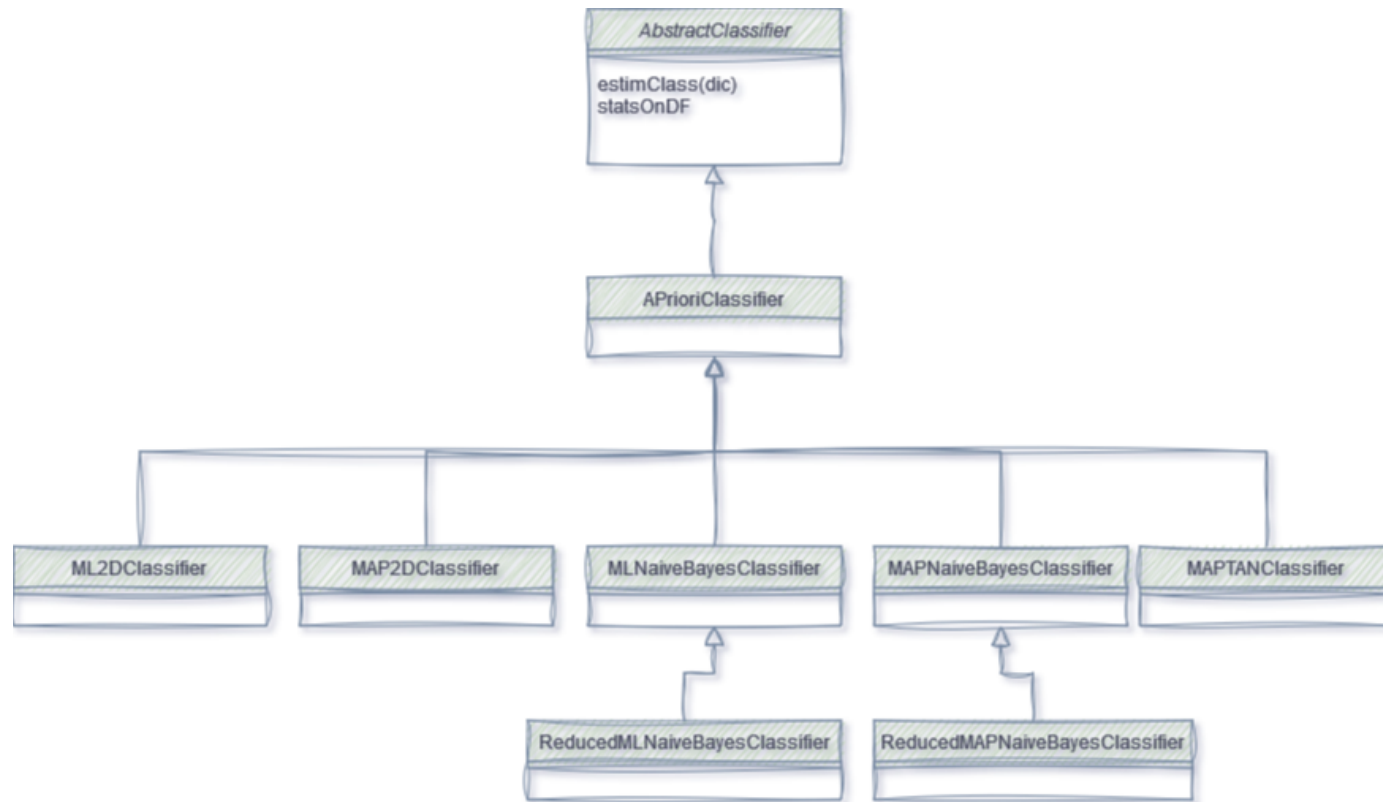
- `estimClass` qui à partir d'un dictionnaire tel que :

```
{ '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 }
```

sera capable de prédire si le patient semble malade ou non en retournant soit `1` soit `0`. (que l'on connaît grâce à `target` qu'il ne faut pas utiliser dans le classifieur bien sûr ! 😊)

- `statsOnDF` qui fournit des statistiques de qualité du `Classifier` en le confrontant à une base de donnée (un `pandas.DataFrame`).

Voici un schéma des classes que vous allez pouvoir créer dans ce projet (en affichant que le nom des classes et non le nom des méthodes à implémenter dans chacune):



### Question 1.2.a

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 [9]: cl=projet.APrioriClassifier()
clpredite=cl.estimClass(None) # n'importe quoi donne la même classe pour un classifieur a priori
# la valeur prédite n'est pas affichée sciemment mais doit valoir 0 ou 1
```

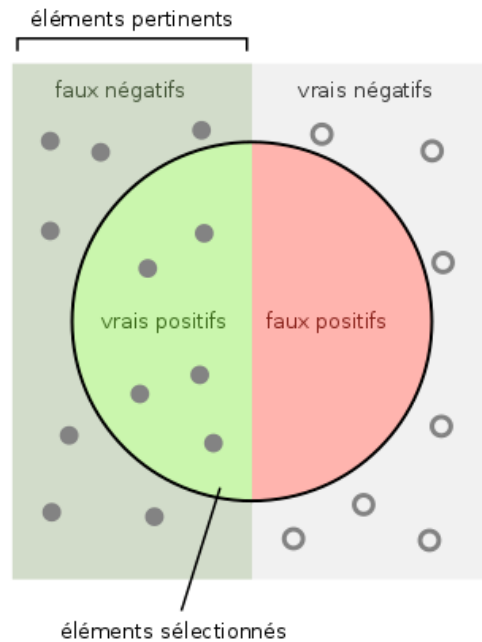
### Question 1.2.b : évaluation de classifieurs

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

- VP: `vrai positif` . Le nombre d'individus avec `target=1` et classe prévue=1
- VN: `vrai négatif` . Le nombre d'individus avec `target=0` et classe prévue=0
- FP: `faux positif` . Le nombre d'individus avec `target=0` et classe prévue=1
- FN: `faux négatif` . Le nombre d'individus avec `target=1` et classe prévue=0



- précision
- rappel



Combien de candidats sélectionnés sont pertinents ?	Combien d'éléments pertinents sont sélectionnés ?
$\text{Précision} = \frac{\text{vrais positifs}}{\text{vrais positifs} + \text{faux positifs}}$	$\text{Rappel} = \frac{\text{vrais positifs}}{\text{vrais positifs} + \text{faux négatifs}}$

- *Petite aide : comment itérer sur un dataframe*

---

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

---

- Par ailleurs, dans `utils`, il y a une fonction `getNthDict(df, n)` qui rend le dictionnaire des attributs de la  $n$ sième 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 [10]: cl=projet.APrioriClassfier()
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': 0.7453874538745388, 'Rappel': 1.0}
test en validation: {'VP': 138, 'VN': 0, 'FP': 62, 'FN': 0, 'Précision': 0.69, 'Rappel': 1.0}
```



Si la méthode `statsOnDF` est correctement écrite, elle sera la même pour tous les prochains classifieurs. **Afin de ne pas avoir à réécrire cette méthode `statsOnDF` qui ne devrait pas changer, on fera hériter tous les classifieurs de `APrioriClassfier` plutôt que de `AbstractClassfier`.**

## 2- classification probabiliste à 2 dimensions

La classification a priori ne donne pas d'excellents résultats puisqu'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 2.1 : probabilités conditionnelles

#### Question 2.1.a

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

```
In [11]: p_thal_given_target=projet.P2D_l(train,'thal')

print(p_thal_given_target)
print()
print(f"Dans la base train, la probabilité que thal=3 sachant que target=1 est {p_thal_given_target[1][3]}")

{1: {1: 0.03217821782178218, 2: 0.7821782178217822, 3: 0.1782178217821782, 0: 0.007425742574257425}, 0: {1: 0.08695652173913043, 2: 0.260869565217391
3, 3: 0.644927536231884, 0: 0.007246376811594203}}
```

Dans la base train, la probabilité que thal=3 sachant que target=1 est 0.1782178217821782

#### Question 2.1.b

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

```
In [12]: p_target_given_thal=projet.P2D_p(train,'thal')
```

```
print(p_target_given_thal)
print()
print(f"Dans la base train, la probabilité que target=1 sachant que thal=3 est {p_target_given_thal[3][1]}")
```

```
{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}}
```

Dans la base train, la probabilité que target=1 sachant que thal=3 est 0.4472049689440994

## Question 2.2 : classifieurs 2D par maximum de vraisemblance

Supposons qu'un individu ait la valeur  $a$  pour  $\text{attr}$ , un classifieur du type  $\text{P2D\_l}$  pourrait donc utiliser  $P(\text{attr}=a|\text{target}=t)$  et sélectionner comme estimation de la classe de l'individu la valeur  $t=0$  ou  $t=1$  maximisant cette probabilité.  $P(\text{attr}=a|\text{target})$  est la vraisemblance d'observer  $\text{attr}=a$  quand  $\text{target}=0$  ou  $\text{target}=1$ . Un tel classifieur utilise donc le principe du **maximum de vraisemblance** (ML=Max Likelihood).

- Pour construire un tel classifieur (dans la méthode `__init__`), 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  $\text{thal}=3$ , alors dans la table `P2Dl`, on trouve  $0.178$  pour  $\text{target}=1$  et  $0.644$  pour  $\text{target}=0$ , 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 de la stricte égalité des 2 probabilités, on choisira la classe  $0$ .

```
In [13]: 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.estimClass(
        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 [14]: 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': 0.8960674157303371, 'Rappel': 0.7896039603960396}

test en validation: {'VP': 113, 'VN': 48, 'FP': 14, 'FN': 25, 'Précision': 0.889763779527559, 'Rappel': 0.8188405797101449}

## Question 2.3 : classifieurs 2D par maximum a posteriori

Supposons qu'un individu ait la valeur  $a$  pour  $\text{attr}$ , un classifieur du type  $\text{P2D\_p}$  pourrait donc utiliser  $P(\text{target}=t|\text{attr}=a)$  et sélectionner comme estimation de la classe de l'individu la valeur  $t=0$  ou  $t=1$  maximisant cette probabilité.  $P(\text{target}|\text{attr}=a)$  est la distribution a posteriori de  $\text{target}$  après avoir observé  $\text{attr}=a$ . 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  $\text{thal}=3$ , alors dans la table `P2Dp`, on trouve  $0.447$  pour  $\text{target}=1$  et  $0.552$  pour  $\text{target}=0$ , la bonne classe d'après le critère du MAP est donc  $0$

Écrire une classe `MAP2DClassifier` qui utilise une telle procédure de maximum a posteriori 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 [15]: 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.estimClass(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 [16]: 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': 0.8713910761154856, 'Rappel': 0.8217821782178217}
test en validation: {'VP': 114, 'VN': 43, 'FP': 19, 'FN': 24, 'Précision': 0.8571428571428571, 'Rappel': 0.8260869565217391}
```

## Question 2.4 : comparaison



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

Cette question ouverte doit donc être répondue dans `projet.py` sous la forme :

```
#####
# Question 2.4 : comparaison
#####
# Nous préférons ... parce que ...
# et aussi parce que ...
#####
```

## 3- Complexités

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 ...xClassifier 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 tailles !!

### Question 3.1 : complexité en mémoire

Écrire une fonction nbParams qui affiche la taille mémoire de ces tables  $P(\text{target}|\text{attr}_1, \dots, \text{attr}_k)$  étant donné un dataframe et la liste  $[\text{target}, \text{attr}_1, \dots, \text{attr}_l]$  en supposant qu'un float est représenté sur 8 octets.



La fonction affiche le résultat et retourne le nombre d'octets !

```
In [17]: 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) # seul résultat visible en sortie de cellule
```

```
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
```

```
Out[17]: 58657996800
```

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 3.2 : complexité en mémoire sous hypothèse d'indépendance complète

Écrire 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 8 octets et **en supposant l'indépendance des variables**.



La fonction affiche le résultat et **rend la valeur calculée en octets** !

```
In [18]: projet.nbrParamsIndep(train[['target']])
projet.nbrParamsIndep(train[['target', 'thal']])
projet.nbrParamsIndep(train[['target', 'age']])
projet.nbrParamsIndep(train[['target', 'age', 'thal', 'sex', 'exang']])
```

```
projet.nbParamsIndep(train[['target','age','thal','sex','exang','slope','ca','chol']])
projet.nbParamsIndep(train) # seul résultat visible en sortie de cellule
```

```
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
```

Out[18]: 800

### Question 3.3 : indépendance partielle

L'indépendance complète comme ci-dessus amène forcément à un classifieur 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 :

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

#### Question 3.3.a : preuve



Pouvez vous le prouver ?

#### Question 3.3.b : complexité en indépendance partielle



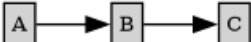
Si les 3 variables  $A$ ,  $B$  et  $C$  ont 5 valeurs, quelle est la taille mémoire en octet nécessaire pour représenter cette distribution avec et sans l'utilisation de l'indépendance conditionnelle ?

## 4- 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  $X$  apparaîtra dans un facteur de la forme  $P(X|\text{Parents}_X)$ . 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 [19]: utils.drawGraphHorizontal("A->B;B->C")
```

Out[19]: 

## Question 4.1 : Exemples

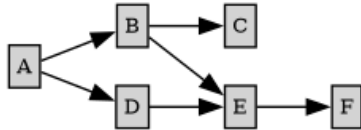


Proposer le code pour dessiner les graphes pour 5 variables \$A,B,C,D,E\$ complètement indépendantes puis pour ces 5 même variables sans aucune indépendance.

(vous pouvez tester dans les 2 cellules suivantes, mais n'oubliez pas de reporter votre proposition dans `projet.py` )

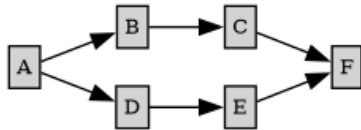
```
In [20]: utils.drawGraphHorizontal("A->B->C;A->D->E->F;B->E") # ce graphe n'est qu'un exemple
```

Out[20]:



```
In [21]: utils.drawGraphHorizontal("A->B->C->F;A->D->E->F") # ce graphe n'est qu'un exemple
```

Out[21]:



## Question 4.2 : 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  $P(\text{attr1}, \text{attr2}, \text{attr3}, \dots | \text{target})$ .
- Écrire comment se décompose la distribution a posteriori  $P(\text{target} | \text{attr1}, \text{attr2}, \text{attr3}, \dots)$  (ou du moins une fonction proportionnelle à cette distribution a posteriori).

## Question 4.3 : 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.

### Question 4.3.a

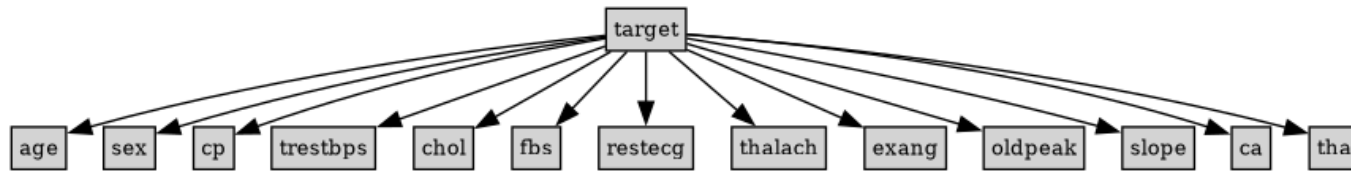
Construire une fonction `drawNaiveBayes` qui a partir d'un dataframe et du nom de la colonne qui est la classe, dessine le graphe.

**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 [22]: projet.drawNaiveBayes(train,"target")
```

```
Out[22]:
```



### Question 4.3.b

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



Comme plus haut, la fonction affiche le résultat et retourne le nombre d'octets !

```
In [23]: 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') # seul résultat visible en sortie de cellule
```

```
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
```

```
Out[23]: 1584
```

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

### Question 4.4 : classifier naïve bayes

Écrire les classes `MLNaiveBayesClassifier` et `MAPNaiveBayesClassifier` 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 [24]: cl=projet.MLNaiveBayesClassifier(train)
```

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

```
Estimation de la proba de l'individu 0 par MLNaiveBayesClassifier : {0: 5.265474022893809e-11, 1: 8.779438846356186e-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: 1.9903404816168002e-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: 1.5920340255297037e-06}
Estimation de la classe de l'individu 2 par MLNaiveBayesClassifier : 1
-----
test en apprentissage : {'VP': 350, 'VN': 116, 'FP': 22, 'FN': 54, 'Précision': 0.9408602150537635, 'Rappel': 0.8663366336633663}
test en validation      : {'VP': 49, 'VN': 60, 'FP': 2, 'FN': 89, 'Précision': 0.9607843137254902, 'Rappel': 0.35507246376811596}
```

```
In [25]: cl=projet.MAPNaiveBayesClassifier(train)
```

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

```
Estimation de la proba de l'individu 0 par MLNaiveBayesClassifier : {0: 0.6719863008964105, 1: 0.32801369910358946}
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: 1.0}
Estimation de la classe de l'individu 1 par MLNaiveBayesClassifier : 1
-----
Estimation de la proba de l'individu 2 par MLNaiveBayesClassifier : {0: 7.90267948988375e-05, 1: 0.9999209732051012}
Estimation de la classe de l'individu 2 par MLNaiveBayesClassifier : 1
-----
test en apprentissage : {'VP': 382, 'VN': 111, 'FP': 27, 'FN': 22, 'Précision': 0.9339853300733496, 'Rappel': 0.9455445544554455}
test en validation      : {'VP': 53, 'VN': 57, 'FP': 5, 'FN': 85, 'Précision': 0.9137931034482759, 'Rappel': 0.38405797101449274}
```

## 5- Feature selection dans le cadre du classifieur naïf 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.

## Question 5.1

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 [26]: for attr in train.keys():
         if attr!='target':
             print(f"target independant de {attr} ? {'YES' if projet.isIndepFromTarget(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
```

## Question 5.2

Proposer des classes `ReducedMLNaiveBayesClassifier` et `ReducedMAPNaiveBayesClassifier` qui utilise cette indépendance pour minimiser le classifieur Naïve Bayes.

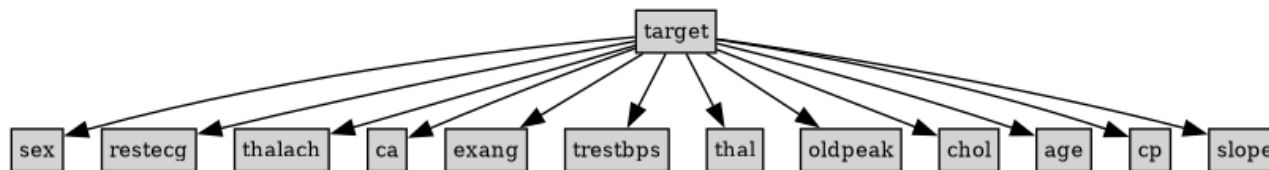


Ces classes devraient hériter des NaiveBayes précédent et ne redéfinir que la construction du classifieur.

Ajouter à ces deux classifieurs une méthode `draw()` qui permet de dessiner le Naive Bayes contenant uniquement les variables finalement sélectionnées.

```
In [27]: cl=projet.ReducedMLNaiveBayesClassifier(train,0.05)
         cl.draw()
```

Out[27]:



```
In [28]: for i in [0,1,2]:
         print(f"Estimation de la proba de l'individu {i} par ReducedMLNaiveBayesClassifier : {cl.estimProbas( utils.getNthDict(train,i))}")
         print(f"Estimation de la classe de l'individu {i} par ReducedMLNaiveBayesClassifier : {cl.estimClass( utils.getNthDict(train,i))}")
         print("-----")
```

```
print(f"test en apprentissage : {cl.statsOnDF(train)}")
print(f"test en validation : {cl.statsOnDF(test)}")
```

Estimation de la proba de l'individu 0 par ReducedMLNaiveBayesClassifier : {0: 3.302888250724298e-10, 1: 5.2938705879520895e-11}  
 Estimation de la classe de l'individu 0 par ReducedMLNaiveBayesClassifier : 0

-----

Estimation de la proba de l'individu 1 par ReducedMLNaiveBayesClassifier : {0: 0.0, 1: 2.386046156003524e-09}  
 Estimation de la classe de l'individu 1 par ReducedMLNaiveBayesClassifier : 1

-----

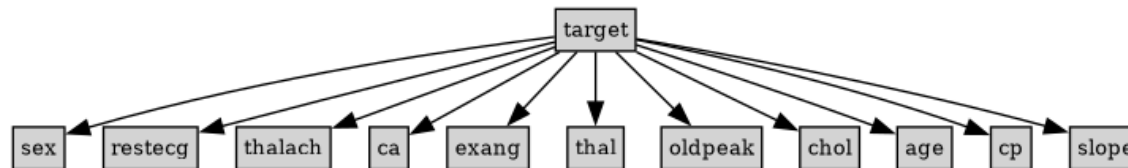
Estimation de la proba de l'individu 2 par ReducedMLNaiveBayesClassifier : {0: 4.3821214730004385e-10, 1: 1.9085511760059345e-06}  
 Estimation de la classe de l'individu 2 par ReducedMLNaiveBayesClassifier : 1

-----

test en apprentissage : {'VP': 350, 'VN': 116, 'FP': 22, 'FN': 54, 'Précision': 0.9408602150537635, 'Rappel': 0.8663366336633663}  
 test en validation : {'VP': 49, 'VN': 60, 'FP': 2, 'FN': 89, 'Précision': 0.9607843137254902, 'Rappel': 0.35507246376811596}

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

Out[29]:



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

Estimation de la proba de l'individu 0 par ReducedMLNaiveBayesClassifier : {0: 5.697482232499415e-09, 1: 2.1387237175326438e-09}  
 Estimation de la classe de l'individu 0 par ReducedMLNaiveBayesClassifier : 0

-----

Estimation de la proba de l'individu 1 par ReducedMLNaiveBayesClassifier : {0: 0.0, 1: 1.147574579792171e-08}  
 Estimation de la classe de l'individu 1 par ReducedMLNaiveBayesClassifier : 1

-----

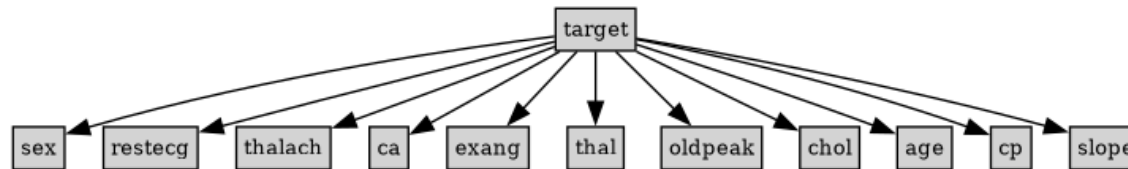
Estimation de la proba de l'individu 2 par ReducedMLNaiveBayesClassifier : {0: 2.748785287609366e-09, 1: 9.17922232269521e-06}  
 Estimation de la classe de l'individu 2 par ReducedMLNaiveBayesClassifier : 1

-----

test en apprentissage : {'VP': 348, 'VN': 117, 'FP': 21, 'FN': 56, 'Précision': 0.943089430894309, 'Rappel': 0.8613861386138614}  
 test en validation : {'VP': 49, 'VN': 61, 'FP': 1, 'FN': 89, 'Précision': 0.98, 'Rappel': 0.35507246376811596}

```
In [31]: cl=projet.ReducedMAPNaiveBayesClassifier(train,0.01)
cl.draw()
```

Out[31]:



```
In [32]: for i in [0,1,2]:
          print(f"Estimation de la proba de l'individu {i} par ReducedMAPNaiveBayesClassifieur : {cl.estimProbas(utils.getNthDict(train,i))}")
          print(f"Estimation de la classe de l'individu {i} par ReducedMAPNaiveBayesClassifieur : {cl.estimClass(utils.getNthDict(train,i))}")
          print("-----")
          print(f"test en apprentissage : {cl.statsOnDF(train)}")
          print(f"test en validation : {cl.statsOnDF(test)}")
```

```
Estimation de la proba de l'individu 0 par ReducedMAPNaiveBayesClassifieur : {0: 0.47643095845795075, 1: 0.5235690415420493}
Estimation de la classe de l'individu 0 par ReducedMAPNaiveBayesClassifieur : 1
-----
Estimation de la proba de l'individu 1 par ReducedMAPNaiveBayesClassifieur : {0: 0.0, 1: 1.0}
Estimation de la classe de l'individu 1 par ReducedMAPNaiveBayesClassifieur : 1
-----
Estimation de la proba de l'individu 2 par ReducedMAPNaiveBayesClassifieur : {0: 0.00010227941341238203, 1: 0.9998977205865877}
Estimation de la classe de l'individu 2 par ReducedMAPNaiveBayesClassifieur : 1
-----
test en apprentissage : {'VP': 375, 'VN': 110, 'FP': 28, 'FN': 29, 'Précision': 0.9305210918114144, 'Rappel': 0.9282178217821783}
test en validation : {'VP': 53, 'VN': 56, 'FP': 6, 'FN': 85, 'Précision': 0.8983050847457628, 'Rappel': 0.38405797101449274}
```

## 6- Evaluation 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 (la précision et le rappel sont des valeurs entre \$0\$ et \$1\$).

### Question 6.1



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

### Question 6.2

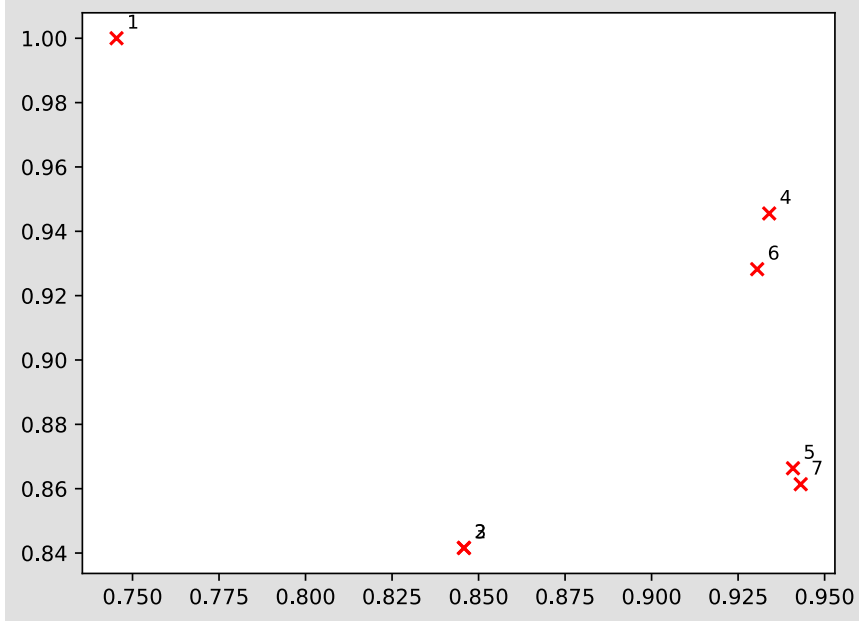
Écrire une fonction `mapClassifieurs(dic,df)` qui, à partir d'un dictionnaire `dic` de `{nom:instance de classifieur}` et d'un dataframe `df`, représente graphiquement ces classifieurs dans l'espace \$(précision,rappel)\$.

```
In [33]: projet.mapClassifieurs({"1":projet.APrioriClassifieur(),
                                "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)

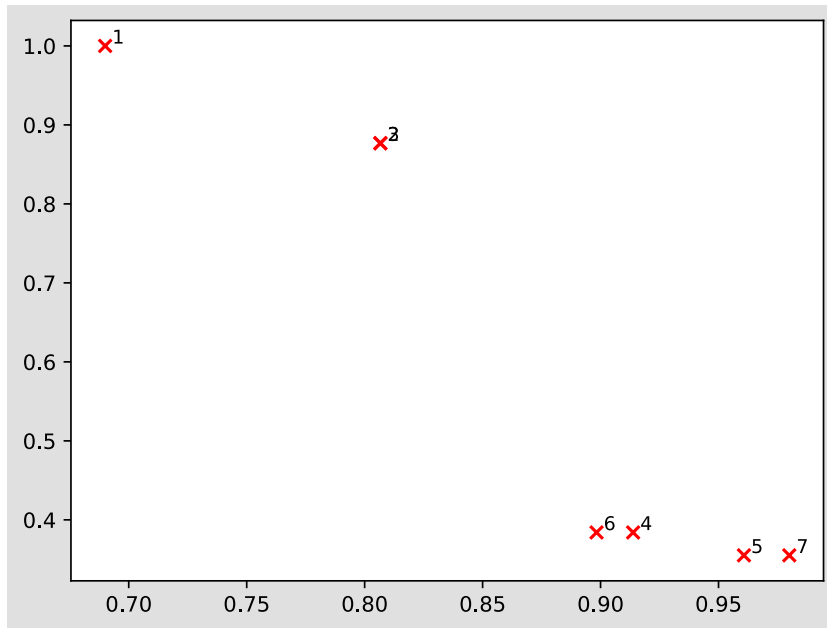
```



```

In [34]: 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 6.3 : Conclusion



Qu'en concluez vous ?

## 7- Sophistication du modèle (question BONUS)

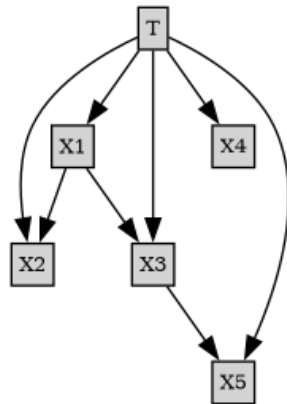
Utiliser un arbre pour représenter la factorisation de la loi jointe est bien sûr une simplification : beaucoup de distribution ne peuvent être représentées avec un seul parent par variable.

Un modèle plus sophistiqué 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 [35]: utils.drawGraph("T->X1;T->X2;T->X3;T->X4;T->X5;X1->X2;X1->X3;X3->X5")
```

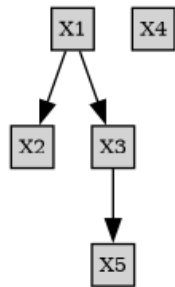
Out[35]:



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

```
In [36]: utils.drawGraph("X1->X2;X1->X3;X3->X5;X4")
```

Out[36]:



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

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

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 7.1 : 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 [37]: for attr in train.keys():
         if attr!='target':
             print(f"target->{attr:10} : {projet.MutualInformation(train,'target',attr):5.7f}")
```

```
target->age      : 0.0590907
target->sex      : 0.0359445
target->cp       : 0.1599540
target->trestbps  : 0.0411980
target->chol     : 0.0405824
target->fbs      : 0.0000413
target->restecg  : 0.0161392
target->thalach  : 0.1401572
target->exang    : 0.1014837
target->oldpeak  : 0.1393573
target->slope    : 0.0938838
target->ca       : 0.1405104
target->thal     : 0.1625536
```

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

On peut également calculer la matrice des informations mutuelles entre attributs, conditionnellement à `target`.

```
In [38]: 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] # on affiche qu'une partie 5x5 de la matrice
```

```
Out[38]: 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 7.2 : 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 [39]: projet.MeanForSymetricWeights(cmis)
```

```
Out[39]: 0.14490408192274776
```

```
In [40]: projet.SimplifyConditionalMutualInformationMatrix(cmis)
cmis[0:5,0:5]
```

```
Out[40]: 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 7.3 : Arbre (forêt) optimal entre les attributs

Un algorithme pour trouver un arbre de poids maximal est l'algorithme 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 [41]: liste_arcs=projet.Kruskal(train,cmis)
liste_arcs
```

```
Out[41]: [('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 7.4: 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 [42]: # 3 arcs de poids quelconques dans le graphe a--b--c    d--e
projet.ConnexSets([( 'a','b',0.878),
                   ( 'a','c',0.4568),
                   ( 'd','e',0.123156)])
```

```
Out[42]: [{'a', 'b', 'c'}, {'d', 'e'}]
```

```
In [43]: projet.ConnexSets(liste_arcs)
```

```
Out[43]: [{'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 [44]: projet.OrientConnexSets(train,liste_arcs,'target')
```

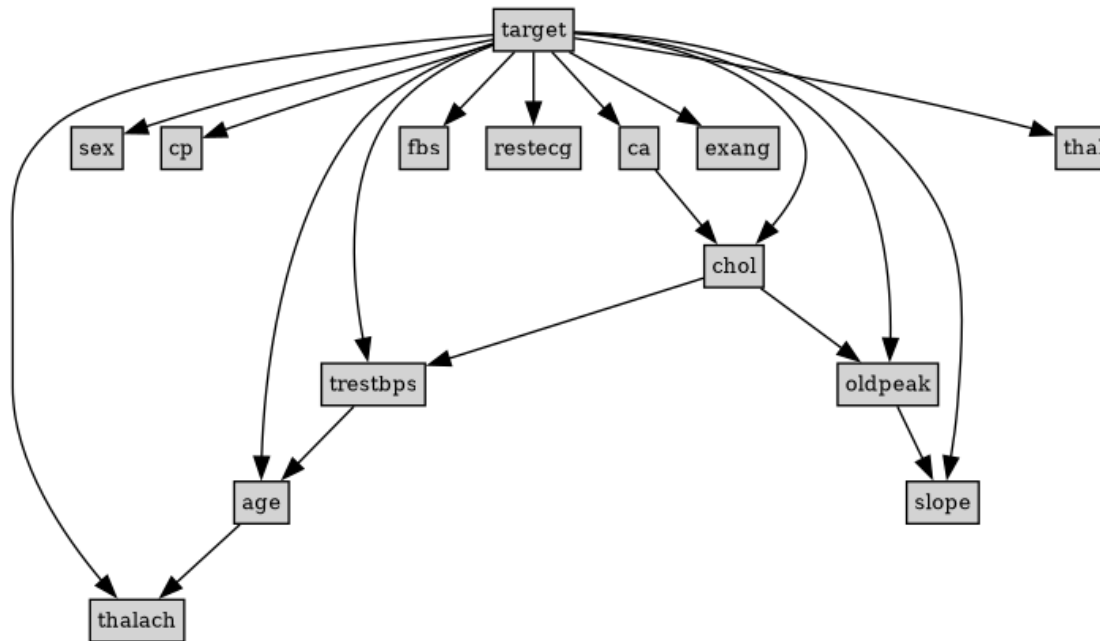
```
Out[44]: [('ca', 'chol'),  
          ('chol', 'trestbps'),  
          ('trestbps', 'age'),  
          ('age', 'thalach'),  
          ('chol', 'oldpeak'),  
          ('oldpeak', 'slope')]
```

## Question 7.5: Classifieur TAN

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

```
In [45]: tan=projet.MAPTANClassifier(train)  
tan.draw()
```

```
Out[45]:
```



```
In [46]: for i in [0,1,2]:
```

```

print(f"Estimation de la proba de l'individu {i} par ReducedMAPNaiveBayesClassifier : {tan.estimProbas(utils.getNthDict(train,i))}")
print(f"Estimation de la classe de l'individu {i} par ReducedMAPNaiveBayesClassifier : {tan.estimClass(utils.getNthDict(train,i))}")
print("-----")
print(f"test en apprentissage : {tan.statsOnDF(train)}")
print(f"test en validation : {tan.statsOnDF(test)}")

```

```

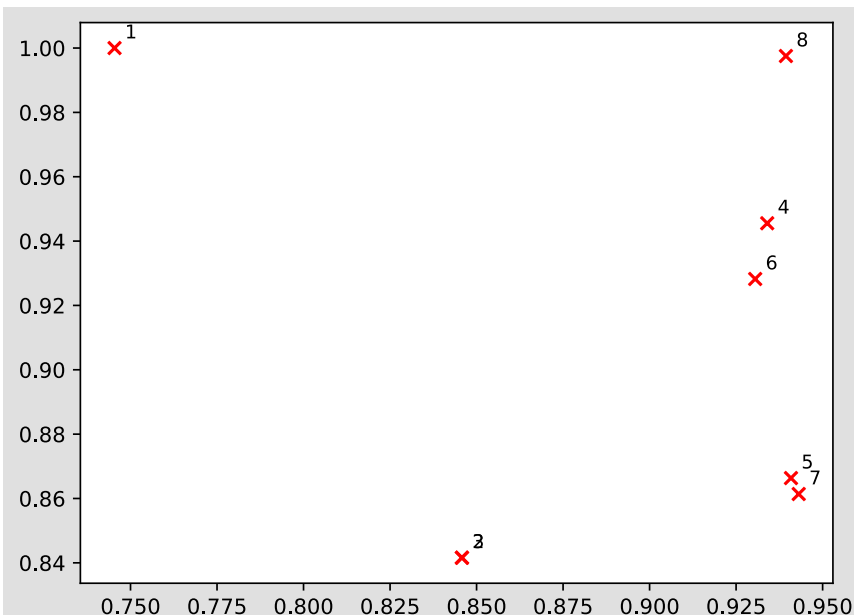
Estimation de la proba de l'individu 0 par ReducedMAPNaiveBayesClassifier : {0: 0.02209731891716822, 1: 0.9779026810828317}
Estimation de la classe de l'individu 0 par ReducedMAPNaiveBayesClassifier : 1
-----
Estimation de la proba de l'individu 1 par ReducedMAPNaiveBayesClassifier : {0: 0.00025585738109411353, 1: 0.999744142618906}
Estimation de la classe de l'individu 1 par ReducedMAPNaiveBayesClassifier : 1
-----
Estimation de la proba de l'individu 2 par ReducedMAPNaiveBayesClassifier : {0: 2.8906366364158206e-06, 1: 0.9999971093633636}
Estimation de la classe de l'individu 2 par ReducedMAPNaiveBayesClassifier : 1
-----
test en apprentissage : {'VP': 403, 'VN': 112, 'FP': 26, 'FN': 1, 'Précision': 0.9393939393939394, 'Rappel': 0.9975247524752475}
test en validation : {'VP': 127, 'VN': 52, 'FP': 10, 'FN': 11, 'Précision': 0.927007299270073, 'Rappel': 0.9202898550724637}

```

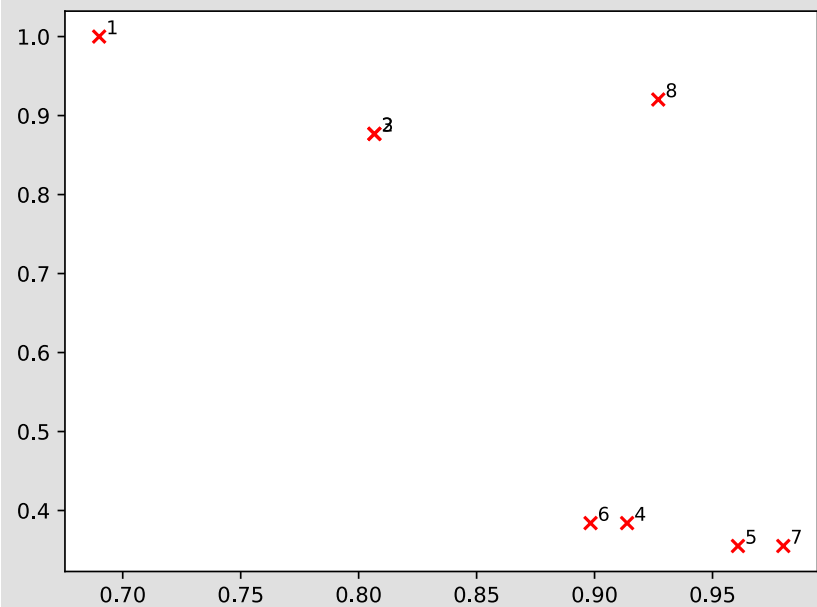
```

In [47]: 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 [48]: 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)
```



## 8- Conclusion finale



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

In [ ]: