

3i005 - projet 2 - 2022oct

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.

Évaluation du projet

L'ensemble des codes que vous réaliserez seront accessibles à partir du 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 ! 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 (visible par



-) et nulle part ailleurs (la cellule existe et contient le texte (votre réponse ici)).
- 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.
- Enfin, tous nos classifieurs seront codés dans des classes, sous-classes de `AbstractClassifier` 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> (<https://realpython.com/python3-object-oriented-programming/#dog-park-example>).
- Vous soumettrez un fichier `projet2-GrX-Y.{zip|tgz}` où `X` est votre groupe, `Y` est votre (ou vos) noms. Le format `zip` ou `tgz` est au choix. (Par exemple : `projet2-Gr1-Nom1Prenom1-Nom2Prenom2.tgz`.)

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.

Simplification de la base (prélude au projet : pas de travail à faire dans cette partie)

```
In [1]: 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 [2]: # 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 FICHER
import utils

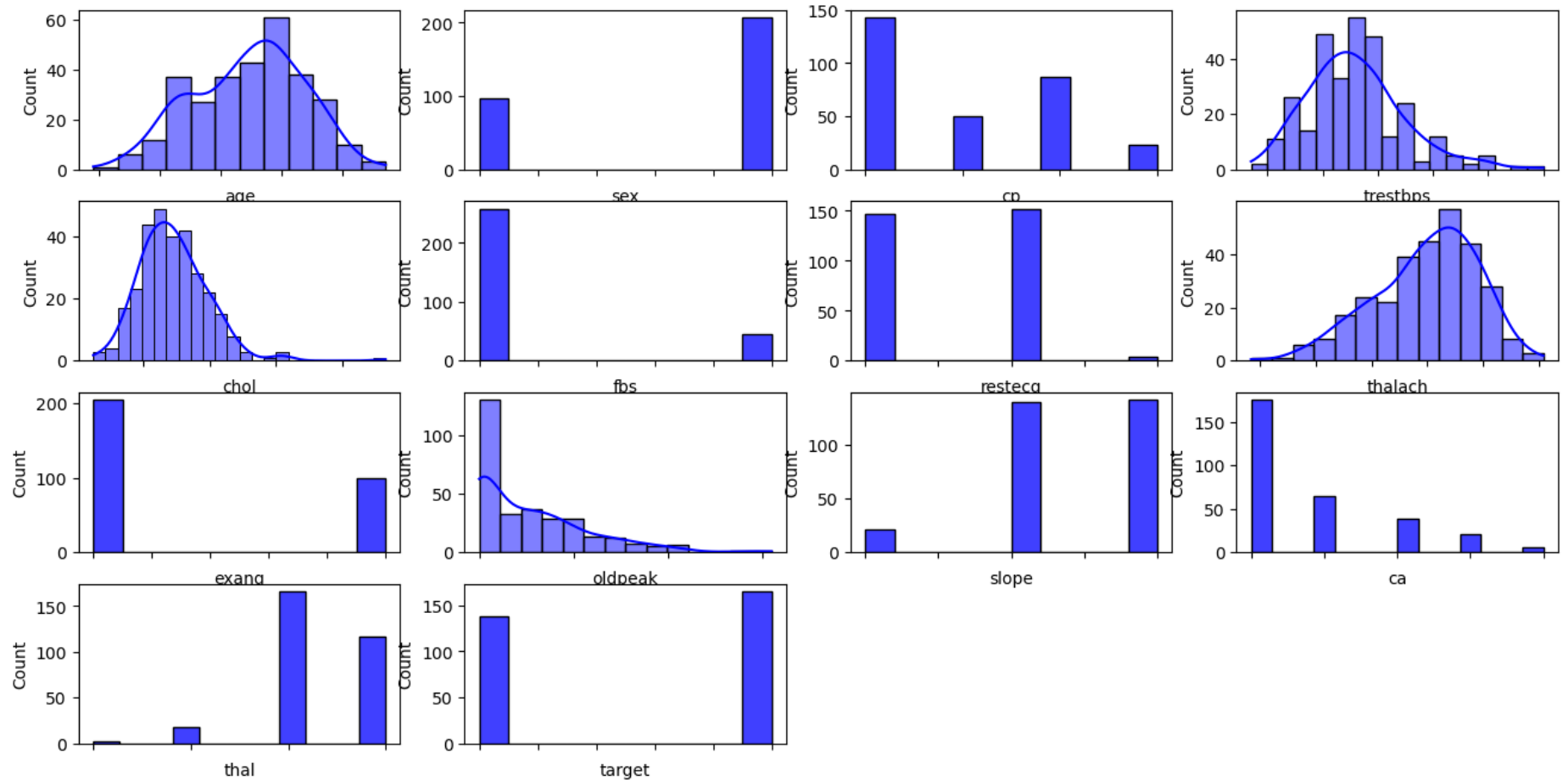
# Cette ligne importe vos codes. VOTRE CODE EST A ÉCRIRE DANS projet.py.
import projet # votre code
```

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

Out[3]:

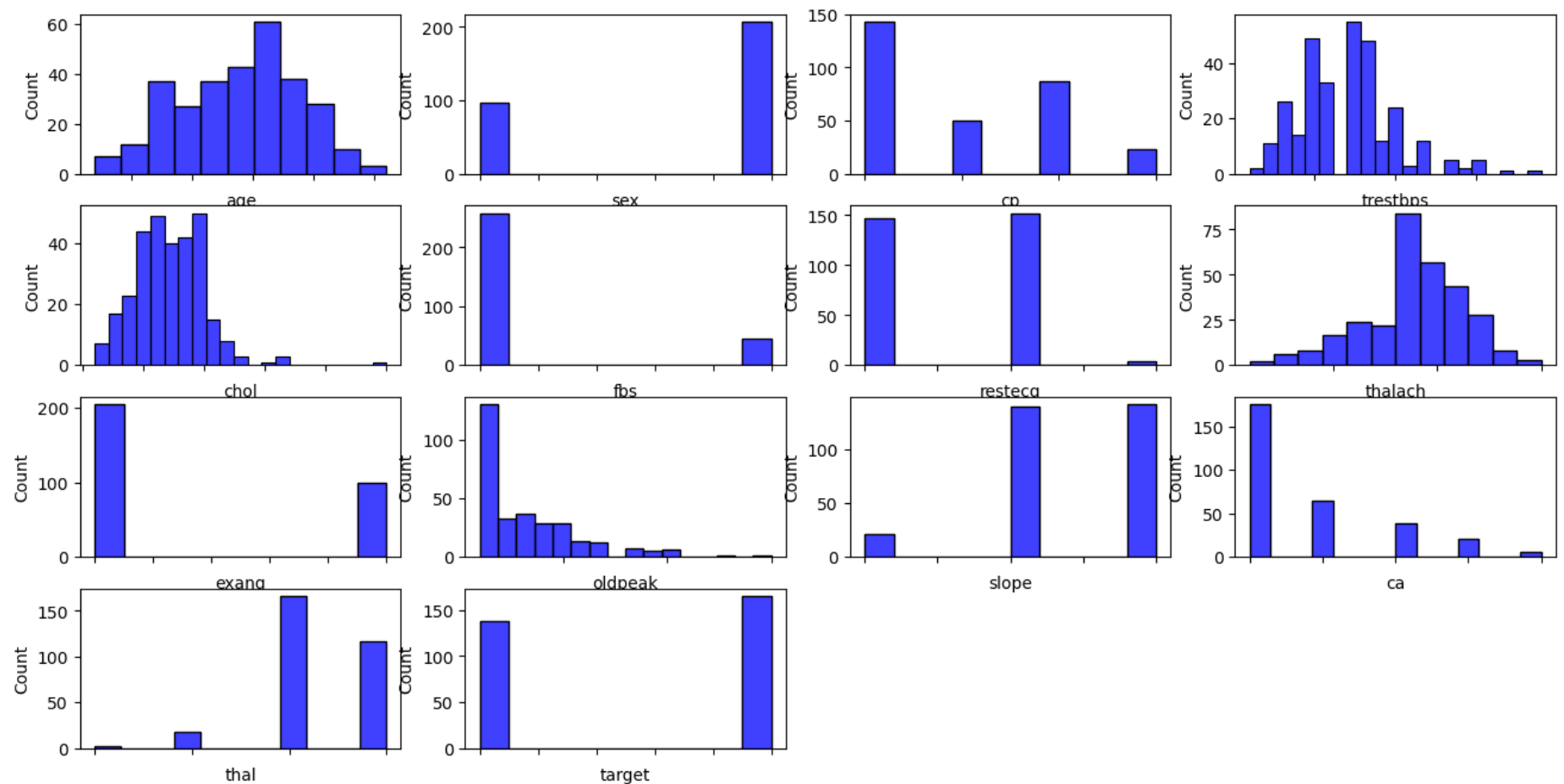
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

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



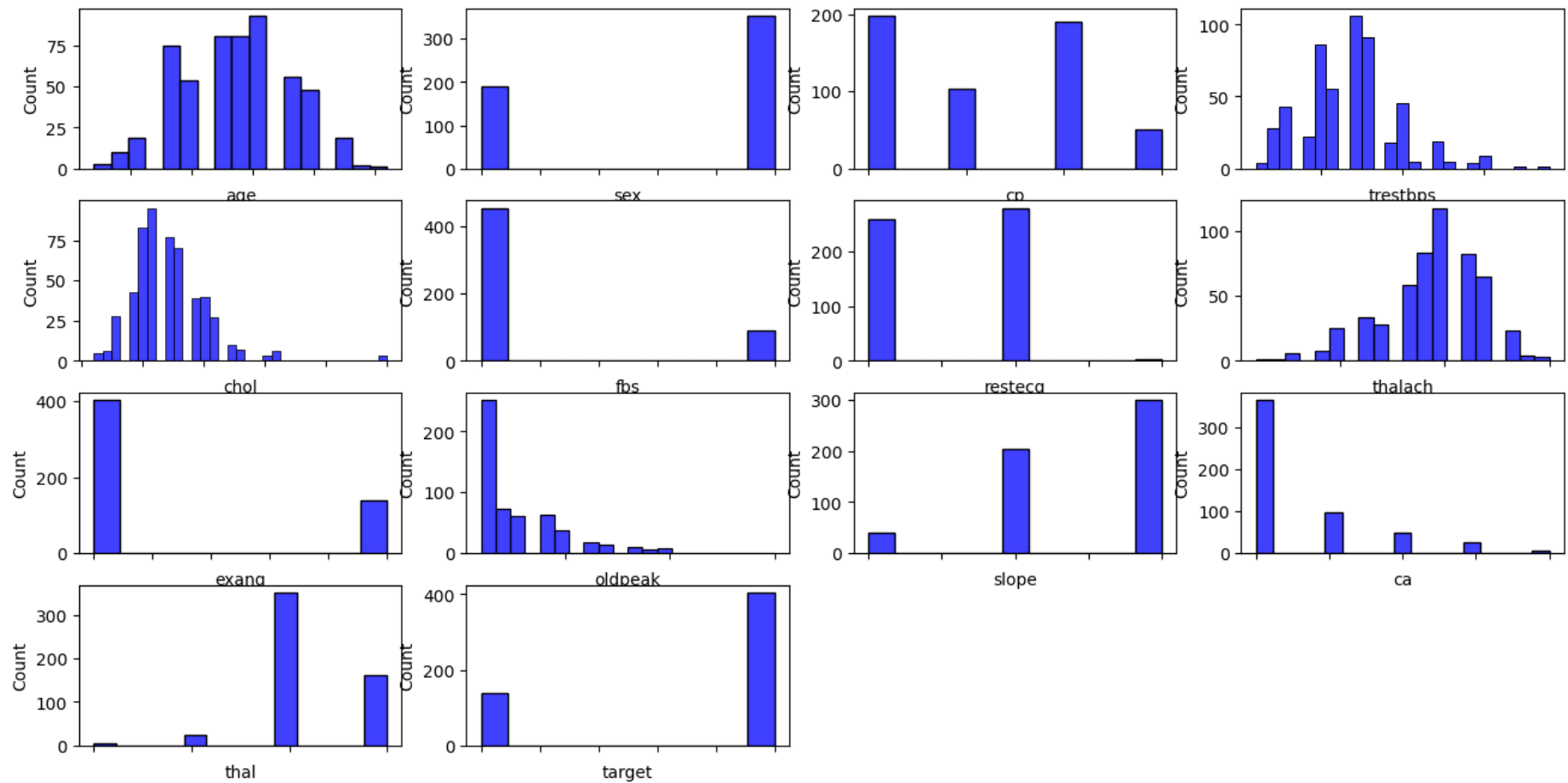
Certaines variables comme `age`, `thalach`, etc. possèdent un grand nombre de modalités, rendant difficile le traitement. Nous simplifions donc la base en discrétisant au mieux toutes les variables qui ont plus de 5 valeurs.

```
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("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]: # cette fonction doit donc rendre un dictionnaire contenant 3 clés 'estimation', 'min5pourcent', 'max5pourcent'  
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 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`

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 [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
```

Question 2b : évaluation de classifieurs

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

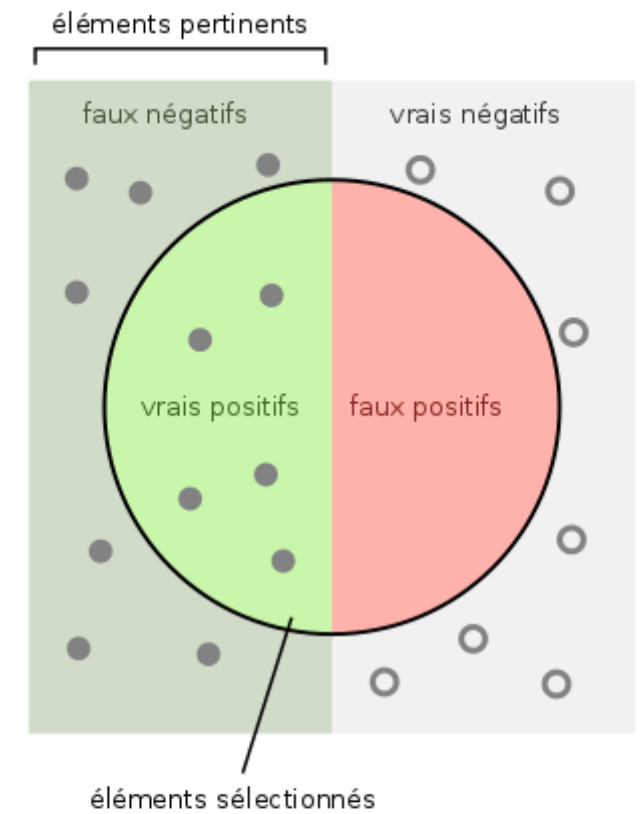
- VP : nombre d'individus avec target=1 et classe prévue=1
- VN : nombre d'individus avec target=0 et classe prévue=0
- FP : nombre d'individus avec target=0 et classe prévue=1
- FN : nombre d'individus avec target=1 et classe prévue=0
- précision
- rappel

- *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 nième 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}
```

Combien
de candidats sélectionnés
sont pertinents ?

$$\text{Précision} = \frac{\text{vrais positifs}}{\text{vrais positifs} + \text{faux positifs}}$$

Combien
d'éléments pertinents
sont sélectionnés ?

$$\text{Rappel} = \frac{\text{vrais positifs}}{\text{vrais positifs} + \text{faux négatifs}}$$

```
In [10]: 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': 0.7453874538745388, 'Rappel': 1.0}
#test en validation: {'VP': 138, 'VN': 0, 'FP': 62, 'FN': 0, 'Précision': 0.69, 'Rappel': 1.0}

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}
```

Question 3 : 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 3.a : probabilités conditionnelles

Écrire une fonction `P2D_1(df, attr)` qui calcule dans le dataframe la probabilité $P(attr|target)$ sous la forme d'un dictionnaire associant à la valeur t un dictionnaire associant à la valeur a la probabilité $P(attr = a|target = t)$.

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

#{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}}
```

```
Out[11]: {1: {1: 0.03217821782178218,
2: 0.7821782178217822,
3: 0.1782178217821782,
0: 0.007425742574257425},
0: {2: 0.2608695652173913,
3: 0.644927536231884,
1: 0.08695652173913043,
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 associant à la valeur t la probabilité $P(target = t|attr = a)$.

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

#{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}}
```

```
Out[12]: {1: {1: 0.52, 0: 0.48},
2: {1: 0.8977272727272727, 0: 0.10227272727272729},
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 $P2D_l$ pourrait 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 $P2D_l$. La fonction `estimClass` rendra la position du maximum trouvé dans cette table.

Supposons un individu dont $thal = 3$, alors dans la table $P2D_l$, 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 $P2D_l$ 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 [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.getNtl
```

```
#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
```

```
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}

#Cette fonction sur ce classifieur ne marche malheureusement pas, mais nous avons quand même essayé
#de l'implémenter correctement plusieurs fois pour qu'elle fonctionne (en vain). La version la plus "poussée"
#est présente dans le code
```

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

Supposons qu'un individu ait la valeur a pour l'attribut, un classifieur du type $P2D_p$ pourrait 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 $P2D_p$. La fonction `estimClass` rendra la position du maximum trouvé dans cette table.

Supposons un individu dont $thal = 3$, alors dans la table $P2D_p$, 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 $P2D_p$ 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 MAP2DClassifier) : {}".format(i,cl.estimClass(utils.getI
```

```
Estimation de la classe de l'individu 0 par MAP2DClassifier) : 1
Estimation de la classe de l'individu 1 par MAP2DClassifier) : 1
Estimation de la classe de l'individu 2 par MAP2DClassifier) : 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}
```

```
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 3.d - comparaison



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

(votre réponse ici)

Nous écartons d'abord 'APrioriClassifier', parce que sa prédiction est toujours retourne avec la même valeur pour toutes les données. Cela ne veut rien dire.

Comparation entre MLE et MAP

ML2DClassifier : MLE consiste à trouver le paramètre θ , qui maximise la fonction de vraisemblance $P(x_0|\theta)$

MAP2DClassifier : MAP essaie de trouver θ , de sorte que $P(x_0|\theta) * P(\theta)$ soit maximisé (en tenant compte de la probabilité antérieure, la probabilité postérieure).

La principale différence entre MAP et MLE est que MAP inclut la distribution de probabilité des paramètres du modèle eux-mêmes, ou plutôt, MLE considère que la probabilité des paramètres du modèle eux-mêmes est homogène, c'est-à-dire que la probabilité est une valeur fixe.

Opinion personnelle

Dans le contexte du titre, je préfère utiliser l'algorithme **MLE**.

Par exemple : Un jour, un patient se rend à l'hôpital pour voir un médecin. Il a dit au médecin qu'il avait mal à la tête. Puis le médecin a décidé, d'après son expérience, qu'il avait un rhume et lui a donné des médicaments à prendre. Bien sûr, il existe de nombreuses causes de maux de tête, comme les rhumes, les accidents vasculaires cérébraux, les hémorragies cérébrales...

Alors qu'est-ce qui fait dire au médecin que le patient a un rhume? Du point de vue de la probabilité, le médecin est calculateur.

- $P(\text{rhumes}|\text{mauxdetête})$ (probabilité qu'un mal de tête soit causé par un rhume, similaire ci-dessous)
- $P(\text{accidentvasculairecérébral}|\text{mauxdetête})$
- $P(\text{hémorragiecérébrale}|\text{mauxdetête})$

Ce médecin constate alors que $P(\text{rhumes}|\text{mauxdetête})$ est le plus élevé et suppose donc que le patient a un rhume.

Dans un hôpital, la plupart des informations dont dispose le médecin sont l'état physique du patient (symptômes). Par conséquent, la façon la plus fiable pour eux dans cette situation est de se baser sur l'expérience passée, même si la valeur de θ est la plus importante.

Bien sûr, l'algorithme MAP peut être meilleur si nous l'utilisons uniquement pour déterminer si un patient est atteint d'une maladie spécifique. Mais c'est un cas très rare.

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

```
In [17]: projet.nbParams(train, ['target'])  
#1 variable(s) : 16 octets
```

Out[17]: 16

```
In [18]: projet.nbParams(train, ['target', 'thal'])  
#2 variable(s) : 64 octets
```

Out[18]: 64

```
In [19]: projet.nbParams(train, ['target', 'age'])  
#2 variable(s) : 208 octets
```

Out[19]: 208

```
In [20]: projet.nbParams(train, ['target', 'age', 'thal', 'sex', 'exang'])  
#5 variable(s) : 3328 octets = 3ko 256o
```

Out[20]: 3328

```
In [21]: projet.nbParams(train, ['target', 'age', 'thal', 'sex', 'exang', 'slope', 'ca', 'chol'])  
#8 variable(s) : 798720 octets = 780ko 0o
```

Out[21]: 798720

```
In [22]: projet.nbParams(train)  
#14 variable(s) : 58657996800 octets = 54go 644mo 640ko 0o
```

Out[22]: 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 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 8 octets et **en supposant l'indépendance des variables**.

```
In [23]: projet.nbrParamsIndep(train[['target']])  
#1 variable(s) : 16 octets
```

Out[23]: 16

```
In [24]: projet.nbrParamsIndep(train[['target', 'thal']])  
#2 variable(s) : 48 octets
```

Out[24]: 48

```
In [25]: projet.nbrParamsIndep(train[['target', 'age']])  
#2 variable(s) : 120 octets
```

Out[25]: 120

```
In [26]: projet.nbrParamsIndep(train[['target', 'age', 'thal', 'sex', 'exang']])  
#5 variable(s) : 184 octets
```

Out[26]: 184

```
In [27]: projet.nbParamsIndep(train[['target', 'age', 'thal', 'sex', 'exang', 'slope', 'ca', 'chol']])
#8 variable(s) : 376 octets
```

Out[27]: 376

```
In [28]: projet.nbParamsIndep(train)
#14 variable(s) : 800 octets
```

Out[28]: 800

Question 4.3

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 :

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



Pouvez vous le prouver ?

(votre réponse ici)

On sait que A est indépendant de B sachant C , donc A est indépendante avec $B|C$. Et on a :

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

En fin, nous pouvons déduire :

$$\begin{aligned} p(A, B, C) &= p(B) * p(A, C|B) \\ &= p(B) * p(A|B) * p(C|B, A) \\ &= p(AB) * p(C|B) \\ &= p(A) * p(B|A) * p(C|B) \end{aligned}$$



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)

Avec l'utilisation de l'indépendance conditionnelle :

$$\text{somme} = 8 * 5 * 5 * 5 = 1000$$

Sans l'utilisation de l'indépendance conditionnelle :

$$\text{somme} = (5+5+5) * 8 = 120$$

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 apparaîtra 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 [29]: utils.drawGraphHorizontal("A->B;B->C")
```




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 [30]: utils.drawGraphHorizontal("A;B;C;D;E")
```

Out[30]:



```
In [31]: utils.drawGraphHorizontal("A->B;B->C;C->D;D->E")
```

Out[31]:



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 $P(attr1, attr2, attr3, \dots | target)$.

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).

(votre réponse ici)

$$p(C|F_1, \dots, F_n) = \frac{p(C) p(F_1, \dots, F_n|C)}{p(F_1, \dots, F_n)}.$$

Dans le **Naïve Bayes**, F_i est indépendant des autres caractéristiques $F_j \neq i$, conditionnellement à C alors :

$$p(F_i|C, F_j) = p(F_i|C) p(F_i|C, F_j) = p(F_i|C)$$

Pour tout $j \neq i$, par conséquent la probabilité conditionnelle peut s'écrire:

$$p(F_1, \dots, F_n|C) = p(F_1|C) p(F_2|C) p(F_3|C) \cdots p(F_n|C) = \prod_{i=1}^n p(F_i|C).$$

Par conséquent, en tenant compte de l'hypothèse d'indépendance ci-dessus, la probabilité conditionnelle de la variable de classe C peut être exprimée par :

$$p(C|F_1, \dots, F_n) = \frac{p(C) \prod_{i=1}^n p(F_i|C)}{\prod_{i=1}^n p(F_i)}$$

où Z (appelé « évidence ») est un facteur d'échelle qui dépend uniquement de F_1, \dots, F_n , à savoir une constante dans la mesure où les valeurs des variables caractéristiques sont connues.

Question 5.3 : modèle graphique et naïve bayes

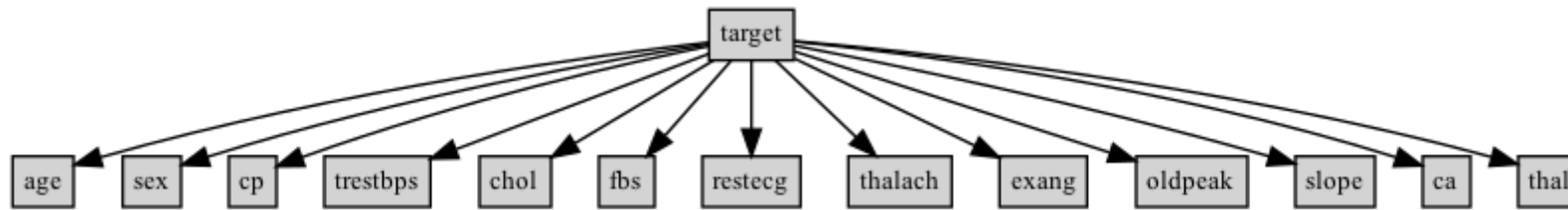
Un modèle naïve bayes se représente sous la forme d'un graphe où le noeud `target` est l'unique parent de tous les attributs. Construire une fonction `drawNaiveBayes` qui à partir d'un dataframe et du nom de la colonne qui est la classe, dessine le graphe.

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

Out[32]:



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 8 octets et **en utilisant l'hypothèse du Naive Bayes**.

```
In [33]: projet.nbParamsNaiveBayes(train, 'target', [])
```

```
#0 variable(s) : 16 octets
```

Out[33]: 16

```
In [34]: projet.nbParamsNaiveBayes(train, 'target', ['target', 'thal'])
```

```
#2 variable(s) : 80 octets
```

Out[34]: 80

```
In [35]: projet.nbParamsNaiveBayes(train, 'target', ['target', 'age'])
```

```
#2 variable(s) : 224 octets
```

Out[35]: 224

```
In [36]: projet.nbParamsNaiveBayes(train, 'target', ['target', 'age', 'thal', 'sex', 'exang'])  
  
#5 variable(s) : 352 octets
```

Out[36]: 352

```
In [37]: projet.nbParamsNaiveBayes(train, 'target', ['target', 'age', 'thal', 'sex', 'exang', 'slope', 'ca', 'chol'])  
  
#8 variable(s) : 736 octets
```

Out[37]: 736

```
In [38]: projet.nbParamsNaiveBayes(train, 'target')  
  
#14 variable(s) : 1584 octets = 1ko 560o
```

Out[38]: 1584

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 [39]: projet.MLNaiveBayesClassifier(train)

for i in [0,1,2]:
    print("Estimation de la proba de l'individu {} par MLNaiveBayesClassifier : {}".format(i,cl.estimProbas(utils.c
    print("Estimation de la classe de l'individu {} par MLNaiveBayesClassifier : {}".format(i,cl.estimClass(utils.c
    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-11, 1: 8.77943884635618
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: 1.5920340255297
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': 0.8
test en validation: {'VP': 49, 'VN': 60, 'FP': 2, 'FN': 89, 'Précision': 0.9607843137254902, 'Rappel': 0.355072463
```

```
Estimation de la proba de l'individu 0 par MLNaiveBayesClassifier : {0: 5.2654740228938083e-11, 1: 8.77943884
6356184e-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: 1.9903404816168007e-09}
Estimation de la classe de l'individu 1 par MLNaiveBayesClassifier : 1
Estimation de la proba de l'individu 2 par MLNaiveBayesClassifier : {0: 3.6835223975945714e-10, 1: 1.59203402
55297035e-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': 0.3550
7246376811596}
```



```
In [47]: cl=projet.MAPNaiveBayesClassifier(train)
for i in [0,1,2]:
    print("Estimation de la proba de l'individu {} par MAPNaiveBayesClassifier : {}".format(i,cl.estimProbas(ut:
    print("Estimation de la classe de l'individu {} par MAPNaiveBayesClassifier : {}".format(i,cl.estimClass(ut:
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: 0.328013699103
#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-05, 1: 0.9999209732
#Estimation de la classe de l'individu 2 par MAPNaiveBayesClassifier : 1
#test en apprentissage : {'VP': 382, 'VN': 111, 'FP': 27, 'FN': 22, 'Précision': 0.9339853300733496, 'Rappel': 
#test en validation: {'VP': 53, 'VN': 57, 'FP': 5, 'FN': 85, 'Précision': 0.9137931034482759, 'Rappel': 0.38405
```

```
Estimation de la proba de l'individu 0 par MAPNaiveBayesClassifier : {0: 0.6719863008964105, 1: 0.32801369910
358946}
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: 7.902679489868447e-05, 1: 0.99992097
32051013}
Estimation de la classe de l'individu 2 par MAPNaiveBayesClassifier : 1
test en apprentissage : {'VP': 382, 'VN': 111, 'FP': 27, 'FN': 22, 'Précision': 0.9339853300733496, 'rappel':
0.9455445544554455}
test en validation: {'VP': 134, 'VN': 32, 'FP': 30, 'FN': 4, 'Précision': 0.8170731707317073, 'rappel': 0.971
0144927536232}
```

Question 6 : *feature selection* dans le cadre du classifieur naïve 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.

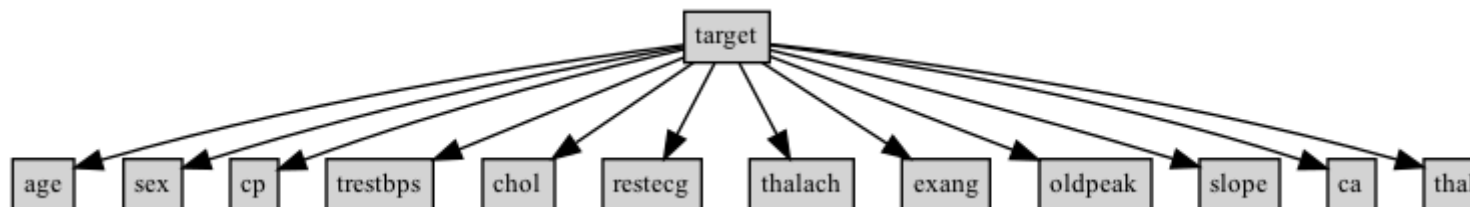
```
In [50]: for attr in train.keys():
         if attr!='target':
             print("target independant de {} ? {}".format(attr,"**YES**" if projet.isIndepFromTarget(train,attr,0.01)

#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
```

```
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
```

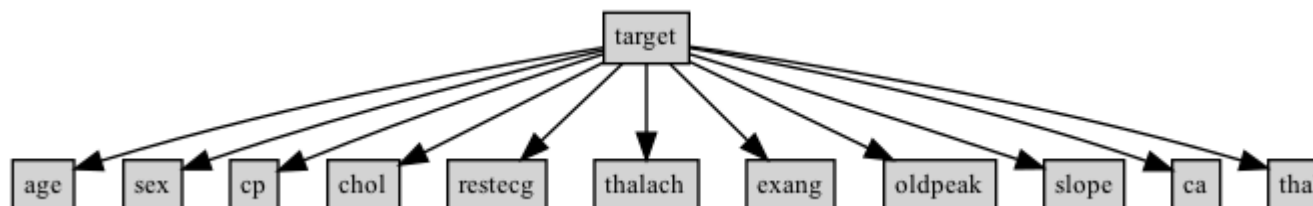
```
In [51]: cl=projet.ReducedMLNaiveBayesClassifier(train,0.05)  
cl.draw()
```

Out[51]:



```
In [52]: cl=projet.ReducedMLNaiveBayesClassifier(train,0.01)  
cl.draw()
```

Out[52]:



In [53]:

```

on de la proba de l'individu {} par MAPNaiveBayesClassifier : {}".format(i,cl.estimProbas(utils.getNthDict(train,
on de la classe de l'individu {} par MAPNaiveBayesClassifier : {}".format(i,cl.estimClass(utils.getNthDict(train,
entissage : {}".format(cl.statsOnDF(train)))
lation: {}".format(cl.statsOnDF(test)))

```

```

Estimation de la proba de l'individu 0 par MAPNaiveBayesClassifier : {0: 3.305142252251692e-09, 1: 1.76920695
8057589e-09}
Estimation de la classe de l'individu 0 par MAPNaiveBayesClassifier : 0
Estimation de la proba de l'individu 1 par MAPNaiveBayesClassifier : {0: 8.93268254119366e-13, 1: 9.511677254
4212e-09}
Estimation de la classe de l'individu 1 par MAPNaiveBayesClassifier : 1
Estimation de la proba de l'individu 2 par MAPNaiveBayesClassifier : {0: 1.594561019107612e-09, 1: 7.57616557
8008588e-06}
Estimation de la classe de l'individu 2 par MAPNaiveBayesClassifier : 1
test en apprentissage : {'VP': 352, 'VN': 114, 'FP': 24, 'FN': 52, 'Précision': 0.9361702127659575, 'rappel':
0.8712871287128713}
test en validation: {'VP': 98, 'VN': 29, 'FP': 33, 'FN': 40, 'Précision': 0.7480916030534351, 'rappel': 0.710
1449275362319}

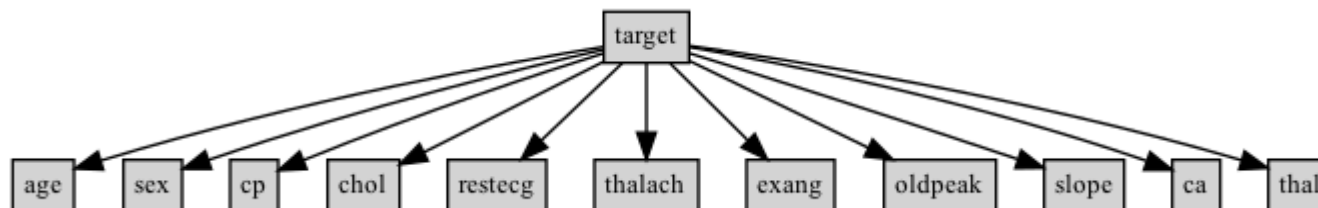
```

```

In [54]: cl=projet.ReducedMAPNaiveBayesClassifier(train,0.01)
cl.draw()

```

Out[54]:



$$précision = \frac{VP}{VP + FP}$$
$$rappel = \frac{VP}{VP + FN}$$

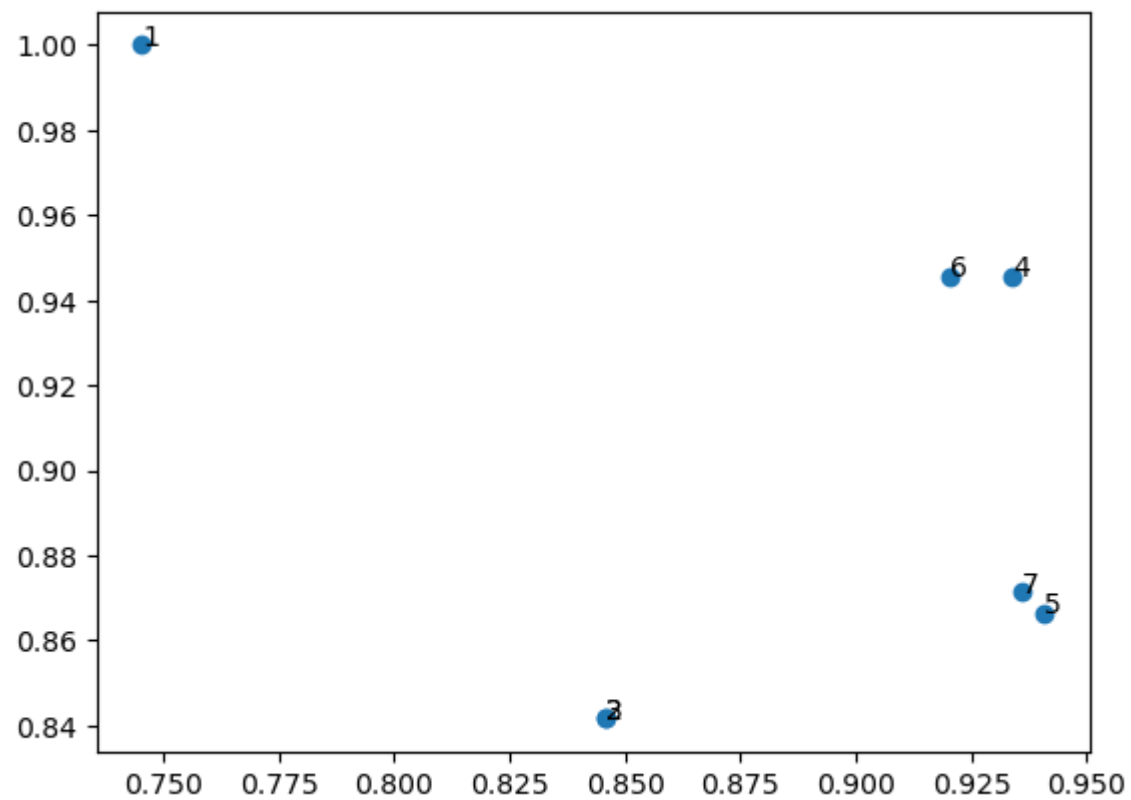
Nous voulons certainement que notre classificateur soit parfaitement exact, c'est-à-dire sans faux positifs (**FP**) ni faux négatifs (**FN**). Dans ce cas, la valeur de précision et rappel vaut 1, donc le point idéal est **(1.0, 1.0)**.

Ainsi, pour comparer chaque classificateur, nous pouvons calculer la distance entre le point correspondant à chaque classificateur sur le graphique et le point idéal. Si la distance est plus petite, le classificateur est meilleur.

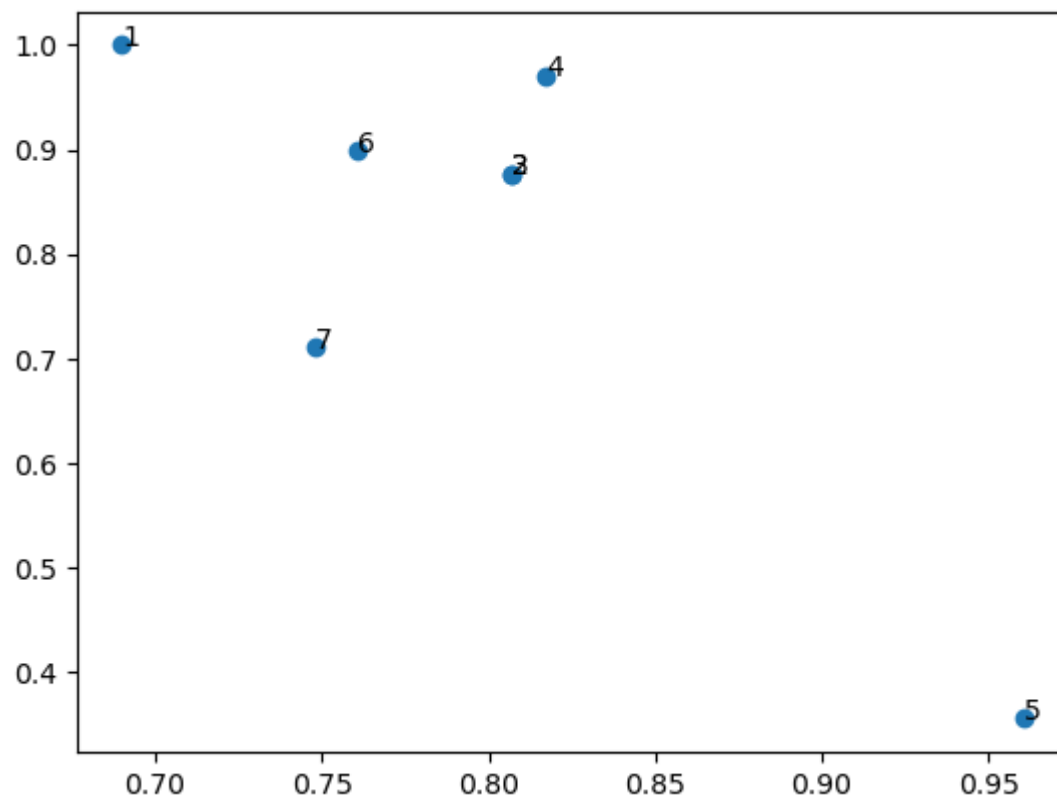
Question 7.2

Écrire 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 classifieurs dans l'espace (*précision*, *rappel*).

```
In [56]: 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 [57]: 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)
```



Conclusion



(votre réponse ici)

1. Les points 2 et 3 sont très proches.

Nous pouvons supposer que la différence entre les algorithmes MLE et MAP n'est pas très grande.

2. Le modèle bayésien simple a des limitations.

Dans la classification bayésienne classique, nous considérons que toutes les caractéristiques sont indépendantes les unes des autres. En réalité, il est impossible que les différents paramètres soient indépendants les uns des autres, et nous faisons cette hypothèse pour faciliter le traitement des données. Cela démontre également que les expériences menées en utilisant séparément les données du '**test**' et du '**train**' ont donné une grande différence entre les deux résultats.

3. Nous avons vérifié la validité de l'algorithme **Reduced_NaiveBayesClassifier()**

La distance entre point 4,6 et point 5,7 est petite. On peut dire que la précision de classer est presque la même. Par conséquent, notre amélioration de l'algorithme est efficace et nous économisons du temps de calcul (la complexité du temps) sans changer le résultat.

Question 8 : Sophistication du modèle (question BONUS)

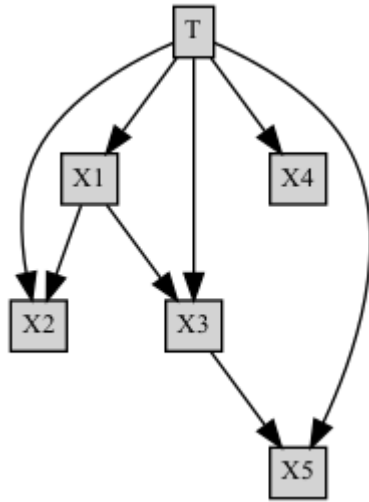
Utiliser un arbre pour représenter la factorisation de la loi jointe est bien sûr une simplification : beaucoup de distributions ne peuvent être représentées avec un seul parent par variable.

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

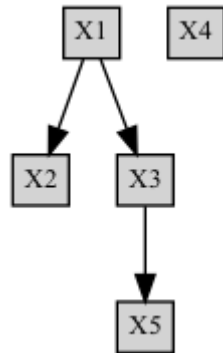
Out[169]:



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

```
In [170]: utils.drawGraph("X1->X2;X1->X3;X3->X5;X4")
```

```
Out[170]:
```



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

```
-----
AttributeError                                Traceback (most recent call last)
/var/folders/5g/nzw_lgz52q76wp0nfc3j1q9c0000gn/T/ipykernel_28474/1429283051.py in <module>
      1 for attr in train.keys():
      2     if attr!='target':
----> 3         print("{}->{} : {}".format("target",attr,projet.MutualInformation(train,"target",attr)))

AttributeError: module 'projet' has no attribute 'MutualInformation'
```

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

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

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

Question 8.3 : 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 [ ]: liste_arcs=projet.Kruskal(train,cmis)
liste_arcs
```

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

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

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

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

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

```
In [ ]: 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 [ ]: 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 ?

(votre réponse ici)

La classification bayésienne est un terme général pour une classe d'algorithmes de classification qui sont tous basés sur le théorème de Bayes

Nous utiliserons principalement les formules suivantes:

$$p(\text{Classe}|\text{Paramètres}) = \frac{p(\text{Paramètres}|\text{Classe}) \times p(\text{Classe})}{p(\text{Paramètres})}$$

Dans la classification bayésienne classique, nous considérons que toutes les caractéristiques sont **indépendantes** les unes des autres. On note également que le dénominateur est **constante** : $\prod p(a_i)$, donc lorsque le nombre de l'échantillon est suffisamment grande (suivi la loi normale), on peut parler de fréquences comme la probabilités directement.

Avantages:

- (1) Logique algorithmique simple, facile à mettre en œuvre.
- (2) Faible surcharge de l'espace et temporelle dans le processus de classification

Désavantages:

Théoriquement, le modèle bayésien simple présente le taux d'erreur le plus faible par rapport aux autres méthodes de classification. Toutefois, ce n'est pas toujours le cas dans la pratique, car l'hypothèse selon laquelle les attributs sont indépendants les uns des autres n'est souvent pas valable dans la pratique, et la classification n'est pas efficace lorsque le nombre d'attributs est important ou que la corrélation entre les attributs est élevée.

En revanche, la méthode bayésienne ordinaire donne de meilleurs résultats lorsque les attributs sont moins corrélés. Pour cela, il existe des algorithmes tels que Bayes semi-parsimonieux qui s'améliorent modérément en prenant en compte la corrélation partielle.