Rapport final

IA pour la santé

Prédiction de l'état de santé d'un patient à l'aide des méthodes d'intelligence artificielle et de machine learning

Client :
Option Projet

Encadrants :
M. Magnin
Mme Poirson

Equipe :
De Catheu Cyril
Fournier Camille
Lam Eric
Sun Honglu
Xie Zheyu
Xinran Tong

CENTRE HOSPITALIER UNIVERSITAIRE DE NANTES

CENTRALE NANTES

Table des matières

Introduction

Le projet s'inscrit dans le cadre d'une collaboration de l'option projet "Maison Connectée pour la santé" de l'Ecole Centrale de Nantes avec le CHU de Nantes. Cette collaboration s'articule sur la possibilité d'effectuer un suivi, à distance, de l'état de santé des patients atteints de maladie chronique. Dans ce cadre, l'utilisation de méthodes issues de l'intelligence artificielle, en particulier le machine learning, a été évoquée comme outil potentiel de prédiction des états de santé de ces patients.

L'objectif de ce projet est par conséquent d'explorer la piste du machine learning et de l'appliquer sur cette problématique. Le but recherché est la prédiction de l'état de santé d'un patient à partir de données diverses obtenues via les objets connectés et dispositifs développés par les équipes de l'option "Maison connectée pour la santé", pour ainsi pouvoir détecter les configurations à risque pour le patient, et de ce fait, dans les cas requérant une assistance médicale d'urgence, alerter le personnel médical compétent de la situation de ce dernier.

Nous disposons pour cela initialement de deux jeux de données non étiquetées recueillis par le CHU et fournis par les étudiants de l'option projet, concernant des patients atteints d'insuffisance cardiaque et de diabète. Ces données n'étant que peu exploitables, des données complémentaires obtenus sur le web seront par la suite utilisées.

Les résultats des prédictions, issus des modèles prédictifs générés par le machine learning, seront présentés à travers plusieurs interfaces graphiques distinctes selon les jeux de données considérés. Nous apporterons ensuite des critiques quant à la portée de nos résultats.

I/ Apprentissage non-supervisé: clustering

1.1. Jeu de données relatif à l'insuffisance cardiaque

Le jeu de données d'insuffisance cardiaque est un jeu de données fournis par le CHU contenant plus de 106000 mesures biologiques de patients suivi pour des raisons d'insuffisance cardiaque.

1.1.1. Pré-traitement des données

Le fichier de mesures original était composé de lignes de la forme :

ID patient | Type de mesure biologique | Date-Heure | Valeur de la mesure | Tranche d'âge .

PATIENT_ID_ANONYMOUS	INDICATEUR	DATE_HEURE	RESULTAT	TRANCHE_AGE
2808863666	PRESSIONSANGUINESYSTOLIQUE	02:48.2	166	80-89
2808863666	PRESSIONSANGUINEDIASTOLIQUE	02:48.2	80	80-89
2808863666	FRÉQUENCECARDIAQUEBPM	02:48.3	87	80-89
2808863666	TEMPÉRATURETYMPANIQUE	02:48.3	37.1	80-89
2808863666	SATURATIONDOXYGENEINSITU	02:48.3	97	80-89
2808863666	PRESSIONSANGUINESYSTOLIQUE	14:51.1	166	80-89
2808863666	PRESSIONSANGUINEDIASTOLIQUE	14:51.1	67	80-89
2808863666	FRÉQUENCECARDIAQUEBPM	14:51.1	95	80-89
2808863666	SATURATIONDOXYGENEINSITU	14:51.1	94	80-89
2808863666	TEMPÉRATURETYMPANIQUE	14:51.1	37.1	80-89
2808863666	FRÉQUENCECARDIAQUEBPM	23:02.2	95	80-89
2808863666	SATURATIONDOXYGENEINSITU	23:02.2	97	80-89

Dans un premier temps, on utilise un tableau croisé dynamique (TCD) pour placer les différentes mesures biologiques en colonne. On rassemble dans la même ligne les différentes mesures biologiques prise à la même date-heure pour un même patient :

Étiquettes de lignes	FRÉQUENCECAR	FRÉQUENCERESPI	POIDS	PRESSIONSANGUINEDIASTO	PRESSIONSANGUINESYSTO	SATURATIONDOXYGENE	TAILLE	TEMPÉRATURETYMPAN
2097387	799	235.2	832	1261	996	341	327.8
40:57.0	70	78	70	113	99	171	37.2
26:25.0	63	78.4	82	118	100	170	36.6
45:54.0	64		71	105	100		36.2
38:19.0	62		77	118	100		36.7
02:18.0	66		68	102	100		36.2
37:11.0	68		74	109	100		35.7
23:36.0			65	111			
23:47.0	71						
38:05.0	67		68	100	100		
03:53.0	71		64	95			
47:59.0	65		67	100	99		36.4
28:29.0	69		55	87	98		36.2
12:08.0	63	78.8	71	103	100		36.6

On peut noter que le codage de date-heure n'est pas passé dans le TCD, mais il peut être retrouvé avec les options de mise en forme. On a enlevé la tranche d'âge : la répartition des valeurs de tranche d'âge semblait très peu intéressante, avec plus de 98% à 50 ans et plus.
On note 4 points importants :

1) les lignes ne sont pas complètes : tous les types de mesures biologiques ne sont pas pris aux mêmes intervalles.
2) Les intervalles et les types de mesure dépendent des patients.
3) Certaines mesures sont prises à des intervalles de temps faible.

4) Certaines mesures étaient en doublons, ont été sommées par le TCD et génèrent des valeurs erronées.

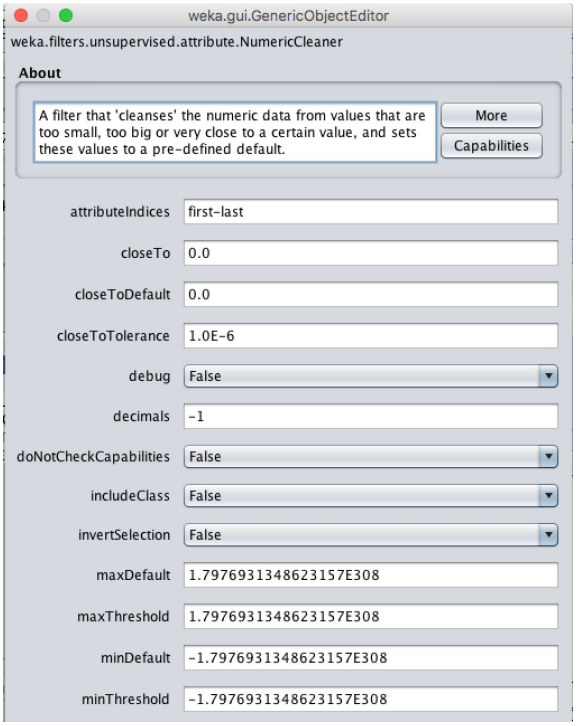
Pour obtenir plus de lignes “pleines”, on réalise encore les traitements suivants :

- on considère la taille du patient constante et on l’ajoute à toutes les lignes des patients pour laquelle la taille est mesurée une fois
- En considérant les intervalles de temps, on se permet de rajouter des poids. Ceci est possible en considérant que le poids d’un jour au suivant varient peu, que l’approximation faite est cohérente avec l’incertitude sur la mesure du poids, et que cette variable n’aura a priori que peu d’importance dans les modèles de clustering (comme on a pu le voir dans la littérature sur les sujets d’insuffisance cardiaque)
- On fusionne des lignes de mesure prises à moins de 10 minutes d’écart environ.

Ces traitements ont été réalisés à la main sur un tiers des données fournies. Nous avons enlevé les valeurs erronées que nous avons repérées pendant ces opérations.
On ne considère pas l’évolution des mesures pour un patient, donc on supprime la colonne ID patient, on obtient alors un fichier de la forme suivante, possédant 3142 lignes :

FReQUENCE(	FReQUENCEI	POIDS	PRESSIONSA	PRESSIONSA	SATURATION	TAILLE	TEMPeRATUI
66	18	100.7	94	150	98	190	35.7
105	17	78	101	120	97	176	36.3
101	21	78	82	123	93	176	36.3
100	11	79	81	119	95	176	35.9
113	16	79	86	119	96	176	37.1
115	28	79	95	122	91	176	36.7
108	26	79	74	115	92	176	36.3
109	29	79	79	109	91	176	36

Il y a à ce point encore des valeurs erronées dans le jeu de données. Comme ces valeurs sont dues à des doublons, elles sont repérables pour beaucoup de constantes qui ne peuvent passer du simple au double comme la température tympanique, le poids et la taille. On supprime ces valeurs à l’aide des fonctions de filtre de Weka *NumericCleaner :*



On complète les champs vides par le caractère “?”. On obtient finalement un jeu de 3076 lignes enregistrées sous forme d’arff, format optimisé pour Weka :

```
@relation 'data-all lines-09022018-weka.filters.unsup
filters.unsupervised.instance.RemoveWithValues-S60.0-
C5-Lfirst-last-V-weka.filters.unsupervised.instance.R
veWithValues-S240.0-C1-Lfirst-last-V'

@attribute FReQUENCECARDIAQUEBPM numeric
@attribute FReQUENCERESPIRATOIRE numeric
@attribute PRESSIONSANGUINEDIASTOLIQUE numeric
@attribute PRESSIONSANGUINESYSTOLIQUE numeric
@attribute SATURATIONDOXYGENEINSITU numeric
@attribute TEMPeRATURETYMPANIQUE numeric

@data
70,?,70,113,99,37.2
63,?,82,118,100,36.6
64,?,71,105,100,36.2
62,?,77,118,100,36.7
66,?,68,102,100,36.2
68,?,74,109,100,35.7
71,?,65,111,100,36
67,?,68,100,100,36
71,?,64,95,100,36
65,?,67,100,99,36.4
69,?,55,87,98,36.2
63,?,71,103,100,36.6
97,?,77,115,86,36.1
90,?,75,118,90,36.3
```

C’est ce jeu final, *insuff_cardiaque-preprocessed.arff*, disponible dans les livrables, et le jeu comprenant encore des valeurs fausses, *insuff_cardiaque-preprocessing_incorrect_values.csv*, qui ont été utilisé par la suite pour tester des algorithmes de clustering.

S'il est souhaité de reprendre le prétraitement plus en amont, le fichier comprenant le TCD, *insuff_cardiaque-preprocessing_TCD.xlsx*, est disponible dans le dossier de livrables.

1.1.2 . Résultat des algorithmes de clustering

Après le traitement des données du CHU, nous avons appliqué des algorithmes de clustering sur ces données. N'ayant aucune indication sur le niveau de pathologie des patients dans ce jeu de données, cela permettait d'obtenir des premiers groupes de patients facilement. La signification médicale de ces groupes devait être ensuite validée par des médecins.

Sur le jeu de données du CHU, après traitement, nous avions 216 lignes complètes et 3141 lignes au total. Nous avons d'abord appliqué les algorithmes sur les lignes complètes avant de les tester à plus grandes échelles.

Parmi les algorithmes de clustering sur Weka, trois se sont révélés prometteurs sur le jeu de données complètes :

- Le Density Based Clusterer : cet algorithme représente chaque instance du jeu de données par un point et associe un point à un cluster selon le nombre de points voisins de ce cluster. S'il y a un nombre minimum *MinPts* de points appartenant à un cluster autour d'une distance *d*, alors ce point appartient à ce cluster. Cet algorithme est initialisé avec une estimation de densité des clusters à trouver et s'exécute ensuite de proche en proche sur tous les points. Nous pouvons optimiser cet algorithme avec les 3 paramètres d'entrées.

Attribut	Jeu complet : 216	Cluster 0 : 153 (83%) ± Deviance	Cluster 1 : 63 (17%) ± Deviance
FReQUENCECARDIAQUEBPM	93	82 ± 18	119 ± 47
FReQUENCERESPIRATOIRE	25	22 ± 6	33 ± 17
POIDS	98	79 ± 17	144 ± 11
PRESSIONSANGUINEDIASTOLIQUE	71	71 ± 11	71 ± 25
PRESSIONSANGUINESYSTOLIQUE	136	123 ± 19	167 ± 66
SATURATIONDOXYGENEINSITU	96	94 ± 3	100 ± 42
TAILLE	168	165 ± 9	175 ± 5
TEMPeRATURETYMPANIQUE	39	36 ± 0.6	44 ± 15
		Prior Probability : 0.70	**Prior Probability : 0.30**

Cet algorithme trouve 2 groupes distincts avec des données complètes. Cependant, il s'est avéré décevant en l'appliquant sur l'ensemble des données du CHU, avec 2 groupes de taille complètement différentes et les valeurs correspondants au cluster ayant un écart-type important :

Attribut	Jeu complet : 3141	Cluster 0 : 3044 (97%) ± Deviance	Cluster 1 : 97 (3%) ± Deviance
FREQUENCECARDIAQUEBPM	83	81 ± 19	168 ± 37
FREQUENCERESPIRATOIRE	26	25 ± 3	44 ± 15
POIDS	83	82 ± 16	148 ± 25
PRESSIONSANGUINEDIASTOLIQUE	69	69 ± 13	82 ± 32
PRESSIONSANGUINESYSTOLIQUE	122	120 ± 24	230 ± 73
SATURATIONDOXYGENEINSITU	95	96 ± 9	102 ± 33
TAILLE	167	167 ± 20	178 ± 5
TEMPERATURETYMPANIQUE	37.1	37 ± 0.6	50 ± 18
		Prior Probability : 0.70	**Prior Probability : 0.30**

- EM (Espérance - Maximisation) : c'est un algorithme itératif basé sur deux étapes. Il calcule d'abord l'espérance de la vraisemblance en tenant compte des dernières variables observées, puis maximise la vraisemblance trouvée à l'étape d'espérance

Attribut	Jeu complet : 216	Cluster 0 : 192 (89%) ± Deviance	Cluster 1 : 24 (11%) ± Deviance
FREQUENCECARDIAQUEBPM	93	83 ± 17	170 ± 35
FREQUENCERESPIRATOIRE	25	23 ± 6	49 ± 18
POIDS	98	91 ± 29	151 ± 12
PRESSIONSANGUINEDIASTOLIQUE	71	70 ± 11	84 ± 36
PRESSIONSANGUINESYSTOLIQUE	136	126 ± 19	219 ± 83
SATURATIONDOXYGENEINSITU	96	94 ± 3	111 ± 68
TAILLE	168	166 ± 9	180 ± 2
TEMPERATURETYMPANIQUE	39	36.8 ± 0.7	57 ± 19

Cet algorithme trouve deux groupes bien distincts qu'on retrouve après sur le jeu complet.
Le deuxième cluster semble toutefois regrouper les valeurs extrêmes.

Attribut	Jeu complet : 3141	Cluster 0 : 2519 (80%) ± Deviance	Cluster 1 : 622 (20%) ± Deviance
FREQUENCECARDIAQUEBPM	83	81 ± 19	94 ± 34
FREQUENCERESPIRATOIRE	26	25 ± 0	26 ± 10
POIDS	83	81 ± 14	94 ± 28
PRESSIONSANGUINEDIASTOLIQUE	69	68 ± 12	74 ± 19
PRESSIONSANGUINESYSTOLIQUE	122	118 ± 22	142 ± 47
SATURATIONDOXYGENEINSITU	95	95 ± 3	97 ± 22
TAILLE	167	167 ± 21	168 ± 8
TEMPERATURETYMPANIQUE	37.1	36.7 ± 0.5	38.5 ± 7.2

- Canopy Clusterer : cet algorithme utilise 2 distances : la distance longue *T1* et la distance courte *T2*. Pour chaque point du jeu de données, l'algorithme calcule la distance *d* au point central du cluster. Si $d<T1$, ce point fait partie du cluster. Si $d>T2$, ce point est trop proche et n'apporte aucune information, il est donc retiré du jeu de données.

Attribut	Cluster 0 : 146	Cluster 1 : 47	Cluster 2 : 13	Cluster 3 : 8
FREQUENCECARDIAQUEBPM	86	76	177	185
FREQUENCERESPIRATOIRE	23	21	49	55
POIDS	97	75	154	176
PRESSIONSANGUINEDIASTOLIQUE	71	66	79	68
PRESSIONSANGUINESYSTOLIQUE	128	120	237	176
SATURATIONDOXYGENEINSITU	94	94	90	92
TAILLE	171	154	180	180
TEMPERATURETYMPANIQUE	36.9	36.6	74	37

Cet algorithme trouve cette fois 4 clusters. Cependant, ceux-ci se retrouvent totalement déséquilibrés en appliquant le Canopy sur le jeu complet de données.

Attribut	Cluster 0 : 3040	Cluster 1 : 35	Cluster 2 : 37	Cluster 3 : 5	Cluster 4 : 8	Cluster 5 : 16
FREQUENCECARDIAQUEBPM	81	129	176	102	113	191
FREQUENCERESPIRATOIRE	25	28	46	69	26	50
POIDS	82	95	155	72	152	144
PRESSIONSANGUINEDIASTOLIQUE	69	129	72	58	79	80
PRESSIONSANGUINESYSTOLIQUE	120	223	211	121	223	257
SATURATIONDOXYGENEINSITU	95	111	98	148	92	91
TAILLE	166	170	180	163	182	178
TEMPERATURETYMPANIQUE	36.8	37.2	37	37.1	73	75.1

L'algorithme EM de clustering est prometteur avec le faible jeu de données que l'on a. Cependant, les clusters trouvés doivent pouvoir être validés et explicables par un médecin pour pouvoir être réellement exploitables dans la pratique médicale. Or, ceci n'est pas toujours possible même si ces clusters ont une réelle signification.
De plus, le jeu de données sur lequel nous avons appliqué ces algorithmes reste de taille relativement faible et non représentatif de la population (les données viennent d'un hôpital, tous les patients sont plus ou moins malades).
Les algorithmes de clustering ne sont donc pas adaptés à notre problématique d'analyse médicale de patients. Ils pourraient toutefois servir pour orienter des recherches (médicales ou autre).

1.1.3 . Remarques sur le jeu de données

Le jeu de données fourni ne comportait a priori pas de patient sain. Avoir des patients sains aurait sûrement amélioré les performances du clustering en créant un cluster de patients sains et des clusters qui se distinguent, correspondant à des patients malades ou à surveiller.
Les mesures ont été fournies pour des patients sous surveillance d'insuffisance cardiaque, mais certains patients cumulaient peut-être d'autres maladies ayant des impacts sur certaines mesures.
Il n'y avait pas d'information sur les médicaments consommés par les patients. Ceux-ci peuvent avoir un impact très important sur certaines mesures et sont nécessaire pour mieux sélectionner les lignes à considérer dans le modèle.

1.2. Jeu de données relatif au diabète

1.2.1. Pré-traitement des données

Le CHU a fourni un jeu de données pour des patients suivi pour du diabète. La structure des données originales est illustrée dans la figure ci-dessous.

PATIENT_I	INDICATEUR	DATE_HEURE	RESULTAT	TRANCHE_AGE
18440410	POIDS	16/05/27 13:35	94.7	50-59
18440410	POIDS	16/05/31 6:21	94.6	50-59
18440410	PRESSIONSANGUINESYSTOLIQUE	16/05/31 20:22	105	50-59
18440410	PRESSIONSANGUINESYSTOLIQUE	16/06/01 6:51	107	50-59
18440410	TAILLE	16/05/27 13:35	175	50-59
18440410	TAILLE	16/05/31 6:21	175	50-59
18440410	TEMPEATURETYMPANIQUE	16/05/27 13:35	36.2	50-59
18440410	TEMPEATURETYMPANIQUE	16/05/27 20:03	36.6	50-59

Dans le tableau il y a six indicateurs différents. Nous voudrions utiliser tous ces six indicateurs pour la prédiction de diabète. On essaie donc de former des lignes pour un patient à une heure données qui comprennent des informations pour ces 6 indicateurs. Pour cela, on utilise un script de Python pour faire des jointures. On joint les indicateurs pour un patient et une heure donnée. On laisse une marge de 5 minutes sur l'heure pour joindre des heures proches. Le résultat est illustré dans la figure ci-dessous.

PATIENT_ID_ANONYMOUS	TEMPEATURETYMPANIQUE	PRESSIONSANGUINEDIASTOLIQUE	PRESSIONSANGUINESYSTOLIQUE	GLYCEMIECAPILLAIREMMOLL	POIDS	TAILLE	TRANCHE_AGE
118087838	37.5	94	159	10.3	123	198	25
118087838	36.9	86	128	9.8	123	198	25
139530623	36.7	59	117	9.8	68.7	188	15
243444455	36.2	62	104	14.3	96	172	65
424768199	37.3	77	145	31.7	94	165	55
459086597	36.4	81	131	3.8	59.5	175	25
459086597	35.5	69	107	11	60.4	175	25
532825729	37.2	73	124	14.6	65.5	170	25
532825729	37.2	67	115	8.2	65.5	170	25

Après avoir fait la jointure, on trouve encore plusieurs lignes incomplètes. On a essayé de compléter certaines lignes à la main. Par exemple, si pour un patient il manque l'information sur le poids alors on regarde la poids de ce patient dans une période d'un ou deux jours. Si ce dernier ne varie pas trop au cours des jours considérés, on l'utilise pour remplir cette ligne. On obtient finalement 346 lignes.

1.2.2. Résultat des algorithmes de clustering

Nous avons appliqué plusieurs méthodes de clustering sur ces 346 lignes. Nous n'avons pas obtenu de résultats très intéressants. Nous pensons qu'il y a deux possibilités : soit le nombre des lignes complètes est trop faible pour les méthodes clustering, soit il n'existe pas une relation simple entre ces indicateurs et le diabète.

Les résultats de clustering en WEKA:
k-means:

Attribute	Full Data (345.0)	Cluster# 0 (236.0)	1 (109.0)
TEMPEATURETYMPANIQUE	36.6417	36.7278	36.455
PRESSIONSANGUINEDIASTOLIQUE	73.7391	70.0297	81.7706
PRESSIONSANGUINESYSTOLIQUE	122.3304	116	136.0367
GLYCEMIECAPILLAIREMMOLL	13.849	12.2797	17.2466
POIDS	67.762	59.1953	86.3101
TAILLE	164.1947	160.2889	172.6514

```
Time taken to build model (full training data) : 0.16 seconds

=== Model and evaluation on training set ===

Clustered Instances

0      236 ( 68%)
1      109 ( 32%)
```

EM:

```
Clustered Instances

0       29 (  8%)
1      204 ( 59%)
2       14 (  4%)
3       98 ( 28%)


Log likelihood: -20.76516
```

Canopy:

```
=== Clustering model (full training set) ===


Canopy clustering
=================

Number of canopies (cluster centers) found: 6
T2 radius: 0.635
T1 radius: 0.793

Cluster 0: 36.620031,74.166667,122.462264,13.779214,65.773585,170.889937,{318} <0,1,3>
Cluster 1: 37.141667,77.083333,136,8.775,55.083333,26.7925,{12} <0,1,4>
Cluster 2: 37.4,10,16,15,75,177,{2} <2>
Cluster 3: 36.728571,79.285714,139.857143,13.271429,144.785714,179.571429,{7} <0,3,5>
Cluster 4: 35.75,46,77.5,6.8,51.7,26.83,{2} <1,4>
Cluster 5: 37.133333,66.666667,116.333333,19.1,154,48,{3} <3,5>



Time taken to build model (full training data) : 0.04 seconds

=== Model and evaluation on training set ===

Clustered Instances

0      309 ( 90%)
1       12 (  3%)
2        2 (  1%)
3       17 (  5%)
4        2 (  1%)
```

Les résultats des différents méthodes ne sont pas pareils et nous manquons de connaissances dans le domaine médical pour juger de la qualité et de la pertinence des classifications.

1.3. Automated Machine Learning

L'idée d'explorer la piste de l'automated machine learning a été évoquée dans le cahier des charges. L'automated machine learning regroupe un ensemble de méthodes exploitant différentes approches permettant d'automatiser une ou plusieurs phases du machine learning, à savoir entre autres parmi les tâches les plus courantes: préparation des données, feature engineering, sélection d'algorithmes, optimisation des hyperparamètres, sélection des métriques ou procédures de validation.
Une première approche naïve a été abordée pour tenter de savoir s'il était possible d'appliquer ces méthodes à notre problème non-supervisé.

L'outil utilisé ici est une extension (plugin) de Weka appelée auto-WEKA. Il propose une recherche par optimisation bayésienne des hyperparamètres en effectuant une exploration de combinaisons d'hyperparamètres pour les algorithmes disponibles, puis suggère à l'utilisateur les méthodes qui sont susceptibles d'avoir de bons résultats. Dans la pratique, il renvoie le modèle, c'est-à-dire l'algorithme avec une combinaison d'hyperparamètres tel que la métrique demandée (taux d'erreur, pourcentage de faux négatifs, etc.) soit optimisée. Auto-WEKA propose nativement plusieurs de ces métriques, ainsi que d'autres paramètres utilisées dans la recherche d'un modèle optimal.

Pour notre problème, il restait à qualifier si cette approche automatique est applicable et/ou efficace pour trouver des modèles satisfaisants.

Une première remarque que l'on peut faire après s'être familiarisé avec l'approche automatique est, utilisé dans le cadre d'une recherche de modèles en se basant sur l'optimisation de métriques, qu'elle n'est pas applicable pour une recherche d'algorithmes dans le domaine non-supervisé, comme il est question dans cadre initial dans lequel nous nous situons. En effet, sans étiquettes sur nos données, il est impossible de quantifier l'exactitude ou la précision de la prédiction trouvée. Notre jeu de données initial étant un problème de clustering et non de classification ou de régression, auto-WEKA ne peut pas proposer de modèles car aucune métrique traduisant de ce fait la justesse de la prédiction ne peut être optimisée.

Il existe néanmoins de nombreuses approches novatrices quant à la recherche automatique d'algorithmes pour des problèmes de clustering qui font l'objet de recherches actives (voir liens en annexe). Cette piste ne sera pas explorée car ces méthodes ne font pas encore objet d'une validation par la communauté scientifique.

En revanche, dans la suite du projet qui s'effectue dans un cadre supervisé, il est possible d'appliquer cette recherche automatique, et que nous allons voir, ne donne pas forcément des résultats probants (diffèrent selon le temps accordé au calcul, parfois de façon illogique…) et qui ne donne pas de résultats plus efficace qu'une recherche naïve d'algorithmes, c'est-à-dire à des essais successifs avec méthodes et hyperparamètres ajustés "à la main". Il reste toutefois un outil efficace pour cibler rapidement le(s) algorithme(s) pouvant donner des résultats satisfaisants, et pour tester rapidement l'exploitabilité d'un jeu de données par les méthodes de machine learning.

II/ Apprentissage supervisé

2.1. Jeu de données Heart Disease

Le jeu de données Heart Disease est un jeu de données qui a fait l'objet de beaucoup d'expérimentations. Nous l'avons décomposé en 2 grandes parties : le jeu de données "Cleveland" composé de 303 lignes quasiment toute pleines, et l'ensemble du jeu de données comportant 920 lignes mais dont beaucoup d'entre elles sont incomplètes.

Le jeu de données possède 14 attributs de mesures biologiques et possède en label le résultat d'un examen médical appelé angiographie. Une angiographie détermine la taille des vaisseaux sanguins et en particulier la réduction de diamètre de ceux-ci. La réduction du diamètres des vaisseaux sanguins augmente la probabilité d'accident cardio-vasculaire. Un examen angiographique est coûteux: le but est donc d'entraîner un algorithme de machine learning sur les données fournies et que celui-ci soit ensuite capable à partir des 14 mesures biologiques moins coûteuses de prédire le résultat d'une angiographie.
On ne s'attardera pas sur le sens de chaque mesure biologique, puisqu'il est difficile sans connaissance médicale de mesurer leur impact sur les vaisseaux sanguins et leur importance dans un modèle de prédiction de résultat d'angiographie.
Le label avait 5 résultats différents : non malade et 4 statuts différents d'angiographie impliquant une maladie. Les recherches sur ce jeu de données faites précédemment ont toutes fait le choix de réduire le problème à la distinction malade/non malade. Ceci peut s'expliquer par le jeu de données trop peu important et le peu d'intérêt dans un premier temps de faire une distinction précise entre des maladies semblables.
Nous avons préalablement chercher à déterminer rapidement si cette distinction en 5 statuts était susceptible de donner des résultats exploitables. En l'absence de résultats probants, nous avons par conséquent choisi l'approche de considérer uniquement deux valeurs, malade/non malade. Un pré-traitement en début d'algorithme transforme toutes les labels de maladie en un label unique de maladie.

2.1.1. Tests préliminaires sur Weka

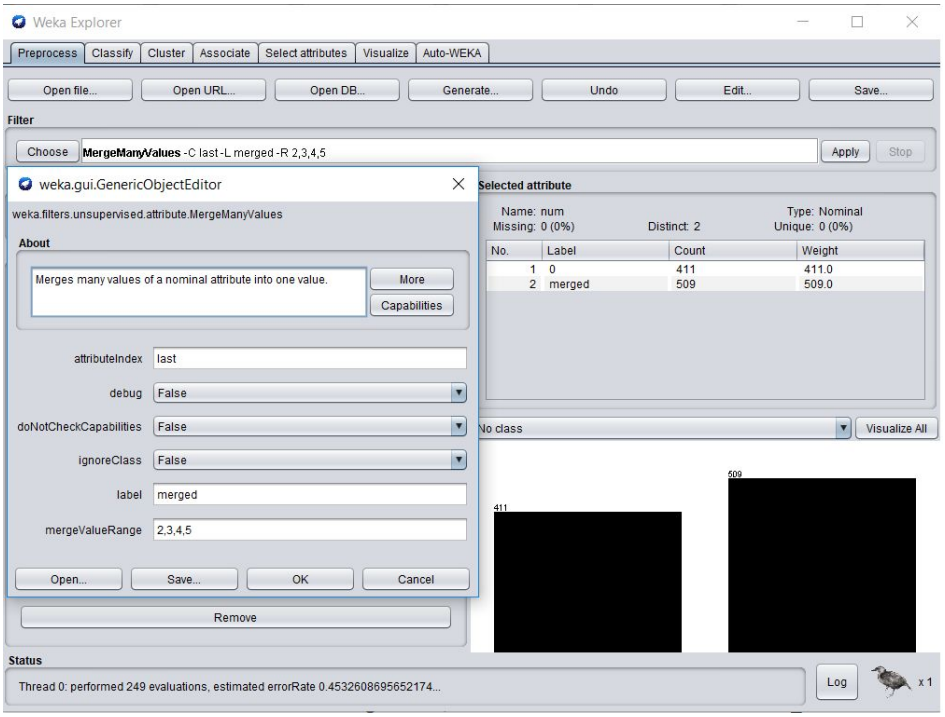
Nous procédons préalablement à une analyse de l'exploitabilité du jeu de données en gardant les cinq labels. Pour ce faire, on utilise une approche automatique, maintenant utilisable dans notre contexte supervisé:

Auto-WEKA output

```
Auto-WEKA result:
best classifier: weka.classifiers.functions.SimpleLogistic
arguments: [-W, 0]
attribute search: null
attribute search arguments: []
attribute evaluation: null
attribute evaluation arguments: []
metric: errorRate
estimated errorRate: 0.4097826086956522
training time on evaluation dataset: 0.316 seconds

You can use the chosen classifier in your own code as follows:

Classifier classifier = AbstractClassifier.forName("weka.classifiers.functions.SimpleLogistic", new Str
classifier.buildClassifier(instances);


Correctly Classified Instances         531               57.7174 %
Incorrectly Classified Instances       389               42.2826 %
Kappa statistic                          0.3453
Mean absolute error                      0.2134
Root mean squared error                  0.3234
Relative absolute error                 77.3793 %
Root relative squared error             87.1093 %
Total Number of Instances              920

=== Confusion Matrix ===

   a   b   c   d   e   <-- classified as
 353  50   5   3   0 |   a = 0
  97 148  13   7   0 |   b = 1
  24  64   7  14   0 |   c = 2
  13  58  13  21   2 |   d = 3
   4  11   1  10   2 |   e = 4

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0,859    0,271    0,719      0,859    0,783      0,586    0,887     0,859     0
               0,558    0,279    0,447      0,558    0,497      0,263    0,720     0,521     1
               0,064    0,039    0,179      0,064    0,095      0,040    0,781     0,247     2
               0,196    0,042    0,382      0,196    0,259      0,209    0,814     0,324     3
               0,071    0,002    0,500      0,071    0,125      0,181    0,853     0,188     4
Weighted Avg.  0,577    0,211    0,531      0,577    0,538      0,372    0,817     0,606
```

La distinction en 5 statuts différents ne donne pas de résultats vraiment exploitables, étant donné que le meilleur algorithme, parmi 375 configurations testées par auto-Weka, donne un taux d'erreur de 42.3%. Ceci est probablement dû, comme énoncé précédemment, à la taille faible du jeu de données.

Par la suite, nous ne garderons uniquement que les deux labels correspondant à un patient non malade (num=0 dans le jeu de données) et un patient malade (num>0 dans le jeu de données. Pour ce faire, sur Weka, nous appliquons un filtre *MergeManyValues* pour merger les labels >0:



Auto-WEKA output

```
Auto-WEKA result:
best classifier: weka.classifiers.bayes.NaiveBayes
arguments: []
attribute search: weka.attributeSelection.GreedyStepwise
attribute search arguments: [-C, -B, -R]
attribute evaluation: weka.attributeSelection.CfsSubsetEval
attribute evaluation arguments: [-M]
metric: errorRate
estimated errorRate: 0.15760869565217392
training time on evaluation dataset: 0.042 seconds

You can use the chosen classifier in your own code as follows:

AttributeSelection as = new AttributeSelection();
ASSearch asSearch = ASSearch.forName("weka.attributeSelection.GreedyStepwise", new String[]{"-C", "-B", "-R"});
as.setSearch(asSearch);
ASEvaluation asEval = ASEvaluation.forName("weka.attributeSelection.CfsSubsetEval", new String[]{"-M"});
as.setEvaluator(asEval);
as.SelectAttributes(instances);
instances = as.reduceDimensionality(instances);
Classifier classifier = AbstractClassifier.forName("weka.classifiers.bayes.NaiveBayes", new String[]{});
classifier.buildClassifier(instances);


Correctly Classified Instances         775               84.2391 %
Incorrectly Classified Instances       145               15.7609 %
Kappa statistic                          0.6805
Mean absolute error                      0.1886
Root mean squared error                  0.3616
Relative absolute error                 38.144  %
Root relative squared error             72.7278 %
Total Number of Instances              920

=== Confusion Matrix ===

   a   b   <-- classified as
 334  77 |   a = 0
  68 441 |   b = merged

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,813    0,134    0,831      0,813    0,822      0,681    0,896     0,872     0
                 0,866    0,187    0,851      0,866    0,859      0,681    0,896     0,902     merged
Weighted Avg.    0,842    0,163    0,842      0,842    0,842      0,681    0,896     0,889
```

Après t=30 minutes, l'algorithme d'auto-WEKA a essayé 311 configurations: la meilleure d'entre elles est l'algorithme sans hyperparamètres Naive Bayes (classification naïve bayésienne) en minimisant le taux d'erreur qui est la métrique que nous avons considéré dans une approche initiale. Ce modèle a un taux de prédiction correct de 84.2%, ce qui est très bon. En revanche, auto-WEKA ne nous fournit qu'un seul algorithme, et il est fort probable que l'essai sur uniquement 311 configurations ne soit pas suffisant pour réellement trouver l'algorithme le plus performant.

Un point sur lequel nous pouvons revenir est qu'en analysant la matrice de confusion et ses termes non diagonaux, nous pouvons remarquer que le nombre de patients malades classifiés en tant que sain, c'est-à-dire de faux négatifs, est relativement élevé (testés négatifs à tort, 13.4%) et contribue de façon conséquente dans le calcul du taux d'erreur, calculé avec le taux de faux positifs (testés positifs à tort, 18.7%).

Dans un contexte médical, cette distinction a son importance, du fait qu'il est plus critique de diagnostiquer un patient malade comme étant sain que le contraire. De plus, cette prédiction ne sera initialement qu'une aide au personnel médical, un patient diagnostiqué malade pourra subir une batterie de tests ultérieurs pour confirmer ou infirmer son état, ce qui est moins vrai pour la situation inverse. De ce fait, malgré le fait que l'algorithme ait un taux d'erreur faible (le plus faible parmi les configurations essayées par l'algorithme automatique), il peut être intéressant de considérer la minimisation du taux de faux négatifs.

Nous cherchons donc ici à minimiser cette métrique. L'algorithme donne le résultat suivant après 442 itérations:

Auto-WEKA output

```
Auto-WEKA result:
best classifier: weka.classifiers.rules.ZeroR
arguments: []
attribute search: null
attribute search arguments: []
attribute evaluation: null
attribute evaluation arguments: []
metric: falseNegativeRate
estimated falseNegativeRate: 0.0
training time on evaluation dataset: 0.0 seconds

You can use the chosen classifier in your own code as follows:

Classifier classifier = AbstractClassifier.forName("weka.classifiers.rules.ZeroR", new String[]{});
classifier.buildClassifier(instances);


Correctly Classified Instances         509               55.3261 %
Incorrectly Classified Instances       411               44.6739 %
Kappa statistic                          0
Mean absolute error                      0.4943
Root mean squared error                  0.4972
Relative absolute error                100      %
Root relative squared error            100      %
Total Number of Instances              920

=== Confusion Matrix ===

   a   b   <-- classified as
   0 411 |   a = 0
   0 509 |   b = merged

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0,000    0,000    ?          0,000    ?          ?        0,500     0,447     0
               1,000    1,000    0,553      1,000    0,712      ?        0,500     0,553     merged
Weighted Avg.  0,553    0,553    ?          0,553    ?          ?        0,500     0,506
```

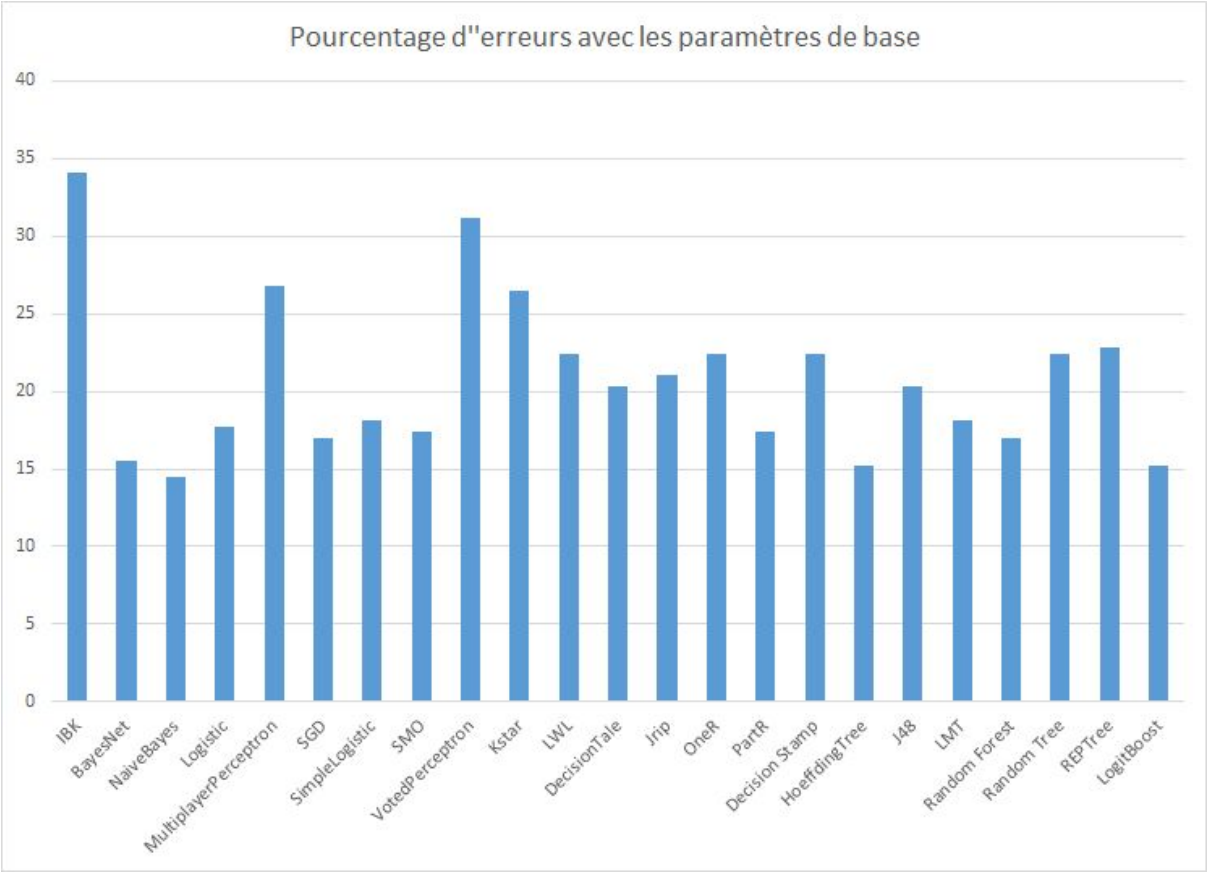
L'algorithme trouvé est ZeroR, qui dans notre cas minimise le taux de faux négatifs (0%)... mais maximise également le taux de faux positifs (100%). Un résultat logique du fait que ZeroR, par

définition, ne prédit correctement que la classe dominante. En effet, il s'agit d'une méthode de classification simpliste qui prédit uniquement en se basant sur la valeur de classe possédant la plus grande cardinalité, sans tenir compte des autres attributs du jeu de données. Etant donné que la catégorie des malades possède un nombre de patients plus élevé que la catégorie des patients sains, l'algorithme ZeroR les classe tous comme tels. Il n'est donc pas du tout intéressant dans notre cas, et il est malheureusement impossible de l'exclure de la liste d'algorithmes considérés par auto-WEKA. Il est également probable que d'autres algorithmes renvoient des résultats similaires (FN faible mais TE élevé). Il nous est par conséquent impossible d'essayer de minimiser la métrique taux de faux négatifs à l'aide d'une analyse automatique.

Par la suite, au cours des essais à la main, nous ne considérerons plus que le taux d'erreur comme métrique d'appréciation de la qualité d'un algorithme. Cette métrique s'avère plus que suffisante dans notre cas (alentours de ~15% d'erreur pour les meilleurs) et permet d'optimiser bien plus rapidement les algorithmes essayés du fait que l'on ne prend pas en compte les combinaisons d'algorithmes/hyperparamètres aberrants tels que nous avons pu en rencontrer. Le choix ultérieur de considérer le taux de faux négatifs ou non devra ensuite être établie avec l'accord du personnel médical, selon l'utilisation faite de nos algorithmes.

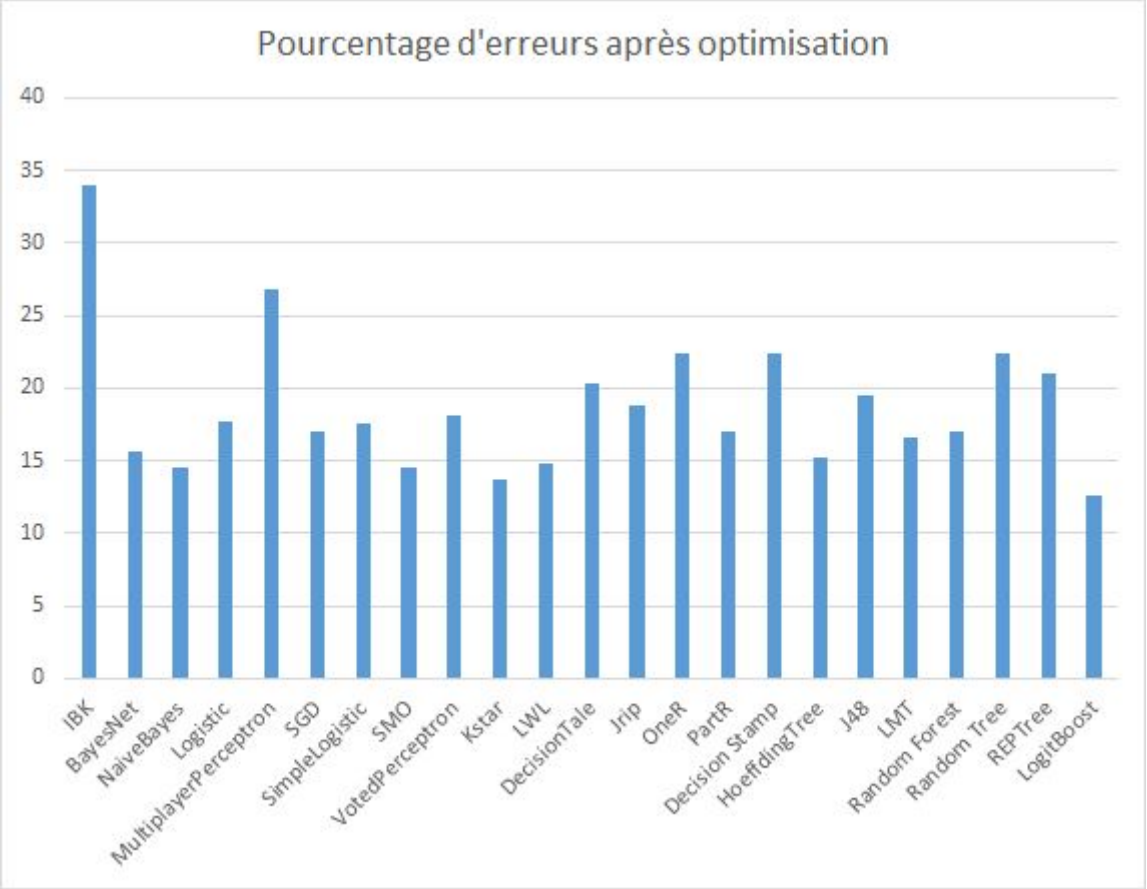
Nous pouvons également noter que l'utilisation d'auto-WEKA donne des résultats très aléatoires. Nous avons tenté de lui laisser plus de temps de recherche mais il renvoie des résultats incohérents, c'est-à-dire plus de temps alloué pour des résultats moins bons. De plus, il donne régulièrement des algorithmes différents qui présentent de bons résultats, ce qui nous pousse à penser que de nombreux algorithmes sont utilisables, et devront par la suite être optimisés "à la main", ce qui fera l'objet de la partie suivante. Une fonctionnalité utile aurait été de pouvoir imposer un algorithme et laisser une méthode automatique chercher les meilleurs hyperparamètres, mais cette option n'est pas disponible.

Une fois ces tests préliminaires effectués, nous procédons donc au test de tous les algorithmes de classification disponibles avec les paramètres de base, avant d'essayer de les optimiser.



On observe déjà que le taux d'erreur pour chacun d'entre eux est déjà beaucoup plus faible que ce qu'on obtient avec les algorithmes de classification.
On remarque aussi que Naive Bayes est le plus performant comme trouvé précédemment.

Afin d'optimiser ces résultats, nous avons étudié l'impact de chaque paramètre sur les résultats avant de les combiner pour obtenir la meilleure optimisation possible.
Après optimisation, nous avons obtenus les résultats suivants :



Nous avons pu faire baisser la moyenne de 20% d'erreurs à 15%.

Nous pouvons observer 2 algorithmes particulièrement efficaces après optimisation : le K* (13,8% d'erreur) et le LogitBoost (12,6%). L'algorithme de classification K* classe chaque instance en la comparant à d'autres déjà classés par un calcul d'entropie. L'algorithme LogitBoost réutilise le training set en augmentant le poids des instances qu'il a classifié incorrectement précédemment. Une variante de l'algorithme LogitBoost a été utilisé lors de l'implémentation. Malheureusement, nous n'avons pas pu implémenter l'algorithme K*.

2.1.2. Principe de l'implémentation

Trois algorithmes ont été implémentés et correspondent à 3 fichiers Python : *HeartAI.py*, *HeartAI-Bayes.py* et *HeartAI-Boost.py*.. On peut noter une forte redondance du code. Notre manque d'expérience dans le développement d'algorithmes d'apprentissage nous a poussé à ne pas prendre de risques pour ne pas confondre et mélanger tous les nombreux paramètres et subtilités des différents algorithmes.

L'utilisation d'algorithmes de machine learning a été faite à l'aide de la librairie *sklearn* disponible sur Python.

a) Fonction principale de prédiction *main*

La fonction principale *main* de chaque fichier tente une prédiction en utilisant un algorithme de machine learning. Le principe général de la fonction *main* des algorithmes est le suivant :

1) on charge un fichier de données, on le pré-traite en enlevant la distinction des différentes maladies

2) si on ne trouve pas de modèle entraîné correspondant au fichier de données, on en crée un sous la forme *[nomdufichier]+"Model" +[Algorithme]+ ".p"* .

3) on charge le modèle et on réalise une prédiction à partir de 14 mesures biologiques données en entrée.

b) Fonction *faireModele*

La fonction *faireModele* crée un modèle et l'enregistre dans un format *pickle*. Ce format est nécessaire pour les modèles établis par *sklearn*.

c) Fonction *analyse*

La fonction *analyse* réalise un test en validation croisée à 10 échantillons (10-fold cross-validation). Le principe est de découper le jeu de données en 10 parties de même taille, d'utiliser une partie comme jeu de test et les 9 autres comme jeu d'entraînement. On mesure alors les 4 indicateurs de performance : accuracy, recall, precision et ROC_AUC. On répète cette opération pour les 10 configurations possibles et on renvoie la moyenne des 4 indicateurs de performances.

accuracy : nombre d'échantillons correctement attribués / nombre d'échantillons

recall : rappel : nombre d'échantillons correctement attribués à la classe i / nombre de document de la classe i

precision : précision : nombre d'échantillons correctement attribués à la classe i / nombre attribués à la classe i

ROC_AUC : aire sous la courbe ROC. La courbe ROC correspond au nombre d'échantillons correctement attribués en fonction du nombre d'échantillons mal attribués. On trace cette courbe au fur et à mesure des classifications sur le jeu de tests. Une valeur de ROC_AUC à 0.5 correspond à un modèle qui donnerait autant d'erreur que de bonnes réponses. Un ROC_AUC à 1 correspond à un modèle qui ne donne que des bonnes réponses.

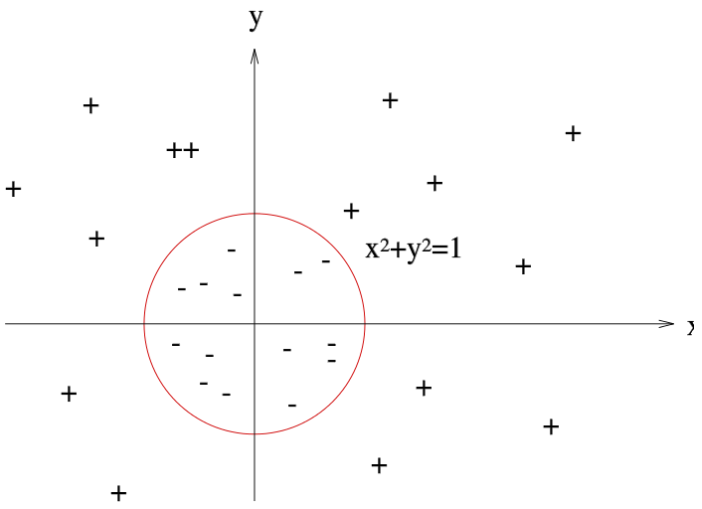
2.1.3. Algorithme SVC

Le premier algorithme implémenté est la machine à vecteur de support SVC.

Le premier principe de cet algorithme est de déterminer une maximiser la marge. La marge est la distance entre un hyperplan de séparation de 2 classes et les échantillons les plus proches. Sans rentrer dans les détails il a été montré que maximiser la marge fait tendre vers de bons modèles par la théorie de Vapnik-Chervonenkis.

Le deuxième principe de l'algorithme SVC est d'utiliser l'astuce du noyau (kernel trick). Cette méthode permet de rendre des problèmes non linéairement séparables en des problèmes linéairement séparables en augmentant la dimension de l'espace des données d'entrées.

Ainsi, un problème de séparation comme celui-ci, qui n'est clairement pas linéaire :



est résolu par l'algorithme SVC.

Nous avons choisi cet algorithme car il été utilisé par un certain Brandon Veber sur le jeu de données Heart Disease et promettait de très bon résultats.

On réalise des premiers test sur le jeu de données Cleveland :

Moy Accuracy	Moy Recall	Moy Precision	Moy ROC_AUC
0.842	0.792	0.863	0.932

On obtient exactement les mêmes résultats que Brandon Veber.

On réalise ensuite des tests sur le jeu de données complet, en mélangeant le jeu aléatoirement avant de réaliser la validation croisée :

Moy Accuracy	Moy Recall	Moy Precision	Moy ROC_AUC
0.822	0.869	0.820	0.919

Moy Accuracy	Moy Recall	Moy Precision	Moy ROC_AUC
0.829	0.878	0.826	0.867

Moy Accuracy	Moy Recall	Moy Precision	Moy ROC_AUC
0.837	0.880	0.833	0.928

Les résultats obtenus sont moins bons que pour Cleveland. Le modèle est légèrement moins bon quand on augmente le nombre de données.
Une des explications peut être qu'il y avait plus de lignes incomplètes dans les données ajoutées au jeu de données Cleveland, et que pour l'apprentissage on remplace alors les valeurs non connues par les valeurs moyennes du jeu de données. Ceci peut avoir un mauvais impact sur la qualité de l'apprentissage, puisque certaine valeur sont "fausses".

Ces résultats sont les plus satisfaisant que nous ayons obtenus. C'est cet algorithme qui a été retenu pour être utilisé dans l'interface graphique pour les médecins.

2.1.4. Algorithme Naive Bayes

Le deuxième algorithme implémenté est la classification naïve bayesienne à multiple Bernoulli. Cet algorithme suppose que les différents types de mesures sont indépendantes (ce qui est bien sûr très probablement faux). L'avantage de cet algorithme est qu'il a besoin de peu de données puisqu'il ne calcule pas de covariance entre les types mesures, mais seulement des variances pour chaque type de mesure.
Tous nos types de données ne sont pas des variables de Bernoulli (0 ou 1). L'algorithme utilise pour les nombres la moyenne puis affecte une valeur 0 pour les valeurs en dessous de la moyenne et une valeur 1 pour les valeurs au dessus. Cette approximation risque de diminuer la qualité du modèle.

On réalise des premiers tests sur le jeu de données Cleveland :

Moy Accuracy	Moy Recall	Moy Precision	Moy ROC_AUC
0.832	0.821	0.812	0.905

Les résultats sont étonnamment bons.

On réalise ensuite des tests sur le jeu de données complets, en mélangeant le jeu aléatoirement avant de réaliser la validation croisée :

Moy Accuracy	Moy Recall	Moy Precision	Moy ROC_AUC
0.801	0.807	0.825	0.806

Moy Accuracy	Moy Recall	Moy Precision	Moy ROC_AUC
0.799	0.803	0.828	0.843

Moy Accuracy	Moy Recall	Moy Precision	Moy ROC_AUC
0.796	0.804	0.826	0.935

Les résultats ont totalement chutés. Cet algorithme, pour les raisons expliquées auparavant et par sa définition un peu simple, n'est pas performant pour le jeu de données Heart Disease.

2.1.5. Algorithme Gradient Boosting

L'algorithme Gradient Boosting est une méthode d'agrégation de modèles. Le principe du boosting est de réaliser un premier modèle, de mesurer l'erreur, puis de pondérer chaque échantillon pour les générations de modèles suivants. Un échantillon qui a généré une erreur est plus fortement pondéré. Ainsi, on concentre l'amélioration du modèle sur les échantillons qui posent problème. Le gradient boosting utilise un calcul de gradient pour calculer la pondération des échantillons. Ce calcul de gradient est similaire à celui utilisé pour les réseaux de neurones.
L'algorithme agrège donc d'autres algorithmes de classification. Ces mini-classificateurs sont des arbres sur le modèle random forest. C'est à nous de donner la profondeur des arbres *n*, ie le nombre de type de mesure évalué par l'arbre.
Pour donner un exemple, après création du modèle :

Pour un **premier échantillon** : le modèle boost détermine que c'est le **mini-classifieur1** qui est le plus apte à donner la bonne réponse. Le mini-classifieur1 utilise un arbre de décision de profondeur *n* pour classifier l'échantillon.
Pour un **deuxième échantillon** : le modèle boost détermine que c'est le **mini-classifieur2** qui est le plus apte à donner la bonne réponse. Le mini-classifieur2 utilise un arbre de décision de profondeur *n* pour classifier l'échantillon.
Nous avons choisi cet algorithme car des implémentations de celui-ci on donné d'excellents résultats sur d'autres jeux de données récemment et aussi du fait que les tests Weka étaient très encourageants.

Après utilisation de la fonction *model_selection.GridSearchCV* qui sert à optimiser les paramètres d'un classifieur, nous avons choisi comme paramètres :

- nombre d'itérations de sélection de mini-classifieur : 250 (valeur par défaut 100 et recommandation d'augmenter dans la plupart des cas).
- profondeur des arbres : 2. Les résultats n'étaient pas meilleurs avec 3 et devenaient instables (bien que parfois meilleurs) avec les valeurs 4 ou 5.

On réalise des premiers tests sur le jeu de données Cleveland :

Moy Accuracy	Moy Recall	Moy Precision	Moy ROC_AUC
0.795	0.758	0.800	0.914

Les résultats pour Cleveland ne sont pas très bons.

On réalise ensuite des tests sur le jeu de données complets, en mélangeant le jeu aléatoirement avant de réaliser la validation croisée :

Moy Accuracy	Moy Recall	Moy Precision	Moy ROC_AUC
0.808	0.842	0.817	0.908

Moy Accuracy	Moy Recall	Moy Precision	Moy ROC_AUC
0.814	0.862	0.812	0.917

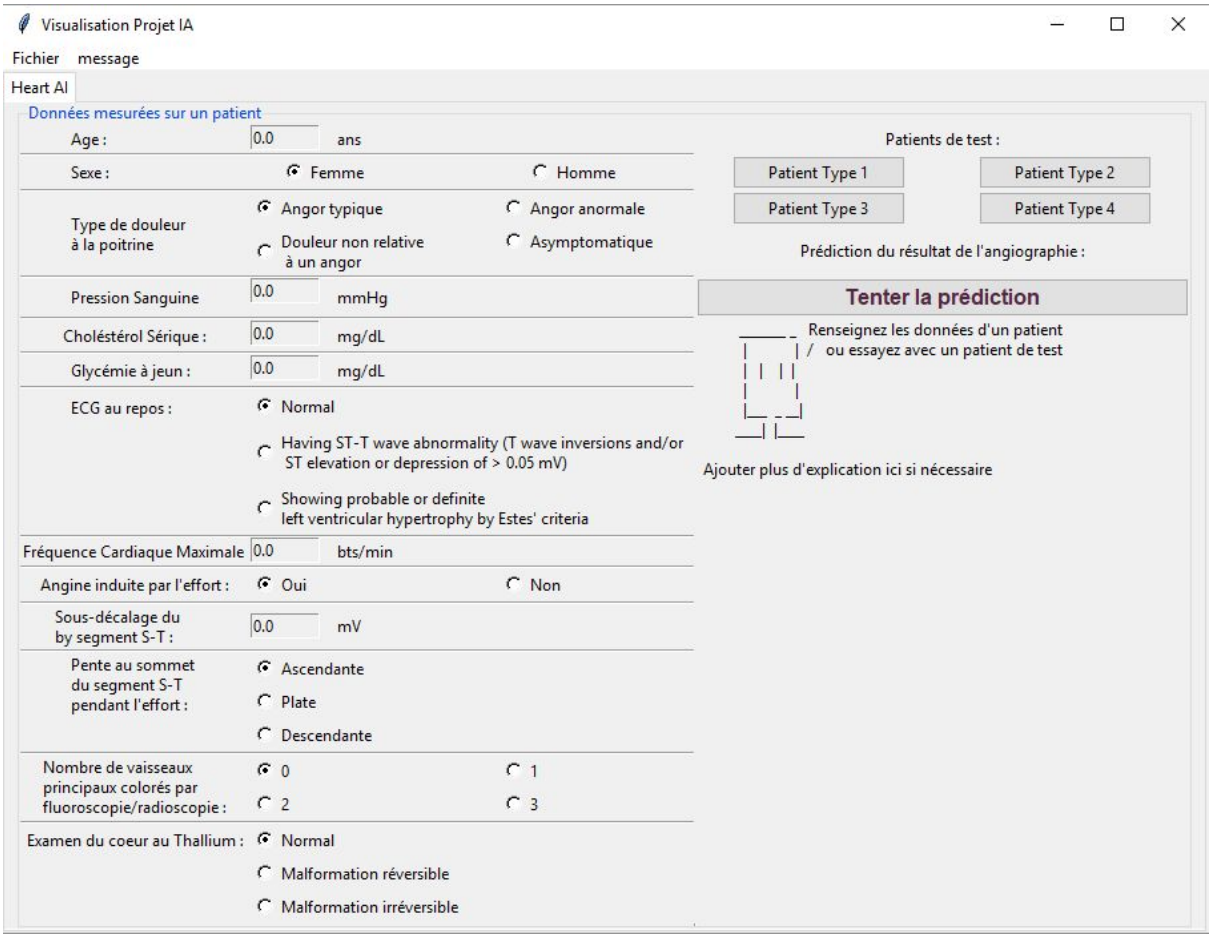
Moy Accuracy	Moy Recall	Moy Precision	Moy ROC_AUC
0.820	0.864	0.821	0.898

Les résultats sont meilleurs de 1 à 2 % ! Si le modèle n'atteint pas encore les performances du SVC, on peut supposer qu'une augmentation de la taille du jeu de données donnera l'avantage à ce modèle. Le modèle Gradient Boosting est donc tout à fait intéressant et notable. De plus, de nombreuses implémentations et variantes commencent à voir le jour. Ces résultats pourraient peut-être être encore améliorés.

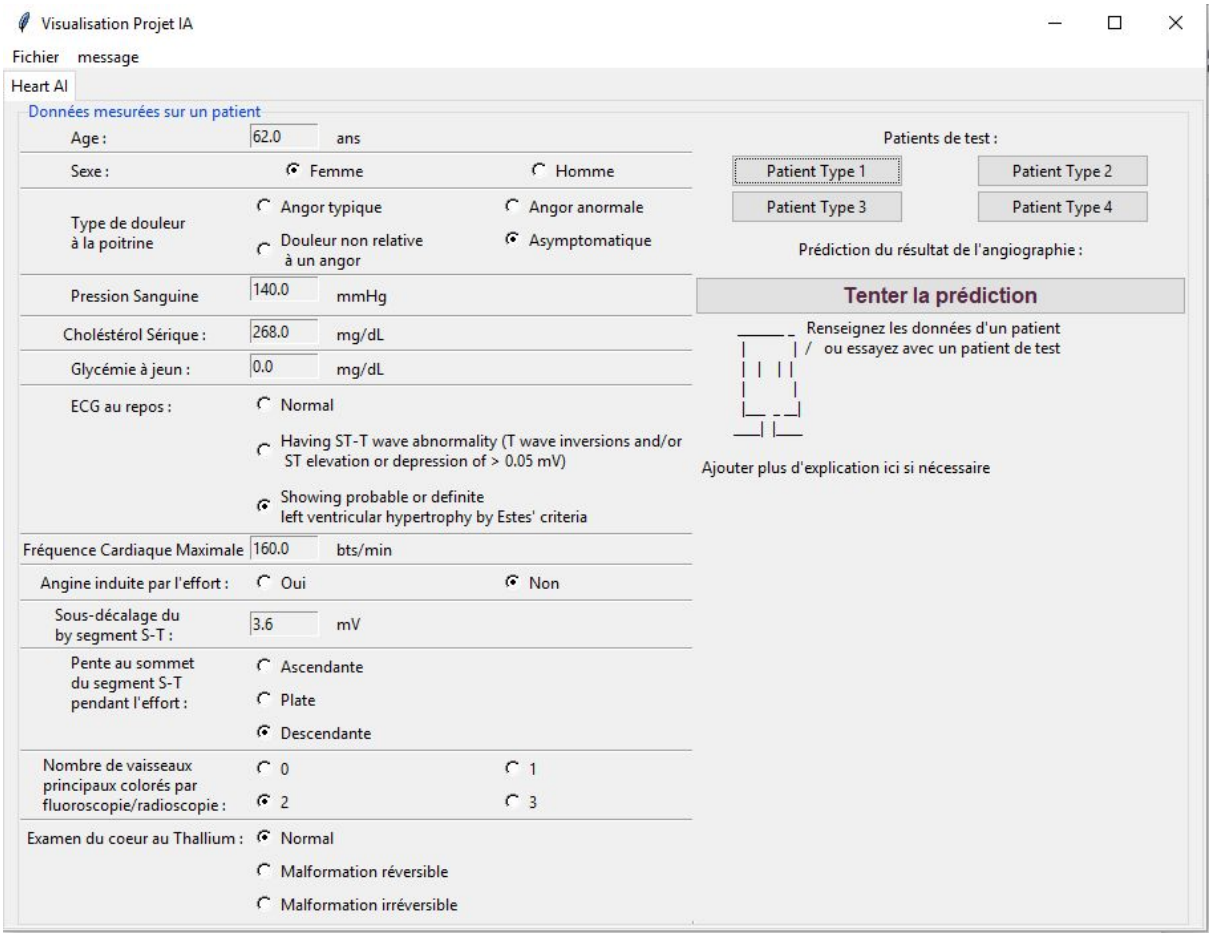
2.1.6. GUI

Pour réaliser la GUI, nous avons utilisé le module tkinter de Python permettant de réaliser facilement des interfaces graphiques. Celle-ci est séparée en 2 fenêtres. La première permet de rentrer

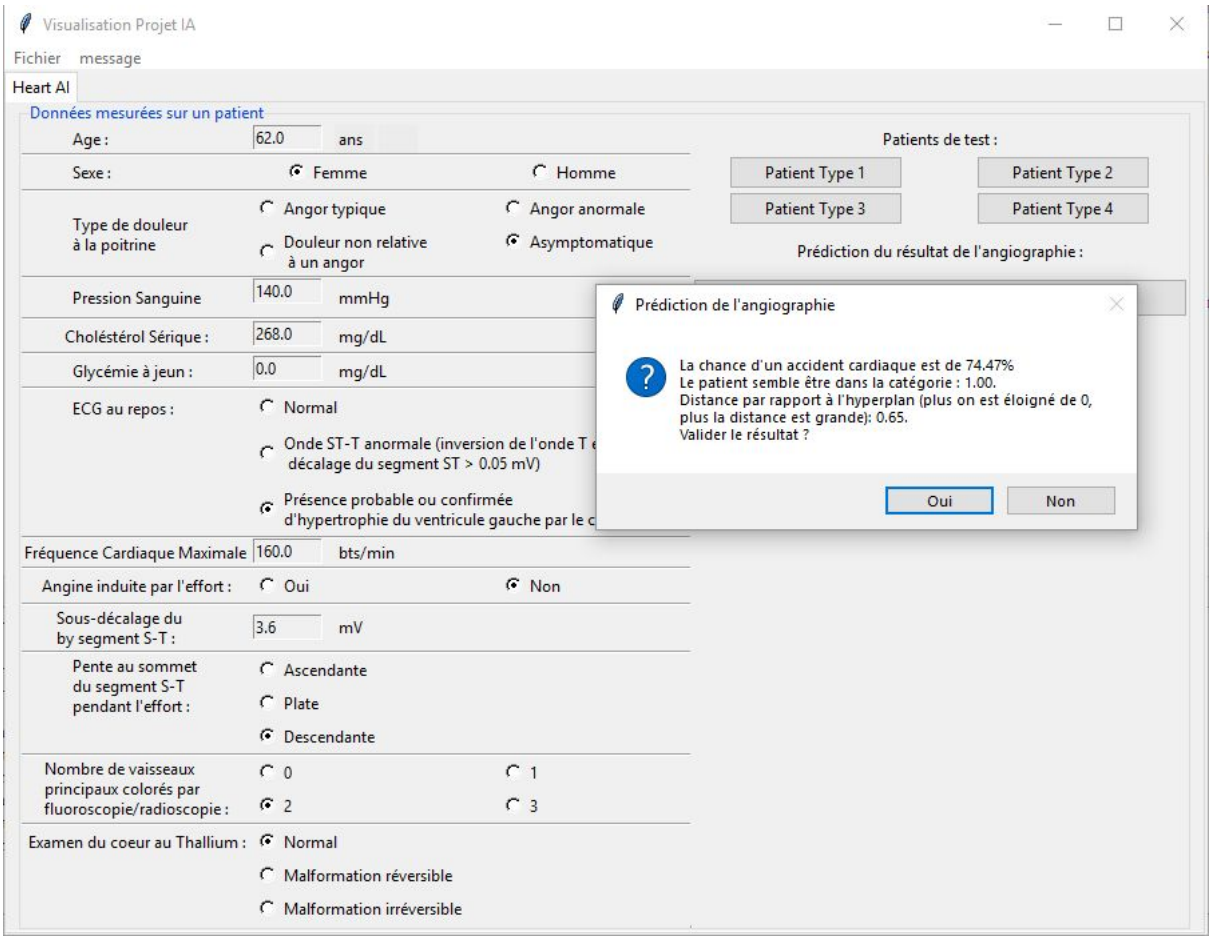
les données d'un nouveau patient et une autre fenêtre de résultats s'affiche lorsqu'on tente une prédiction.



Les boutons " Patient Type " permettent de remplir le formulaire avec des valeurs par défauts afin d'avoir des exemples de résultats possibles. Ainsi, en cliquant sur le bouton " Patient Type 1 ", nous obtenons les valeurs par défaut :



Ce patient type donne le résultat suivant :



Le médecin peut alors valider le résultat pour ajouter le patient dans la base de données et la bonne catégorie afin d'affiner les résultats futurs.

2.2. Multiples diseases

Pour la partie multiples diseases, nous utilisons un jeu de données déjà traité obtenu sur le web. Ce jeu de données, comme son nom l'indique, contient des données relatives à plusieurs maladies et divers symptômes pouvant être la cause de ces maladies.

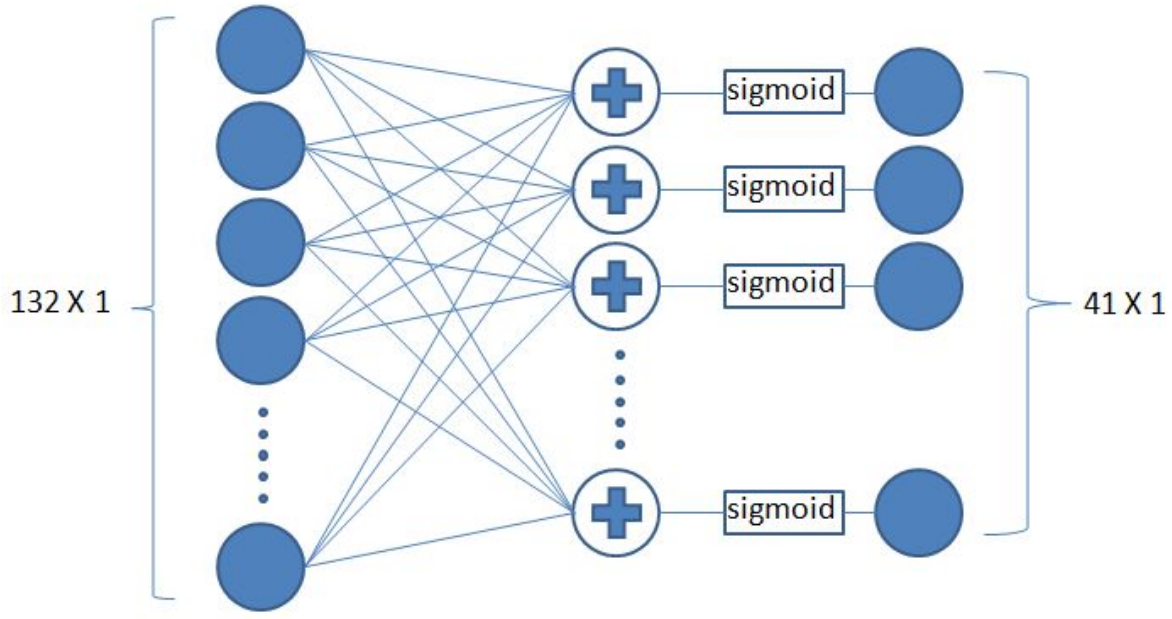
Une partie de ce jeu de données est illustrée dans la figure ci-dessous:

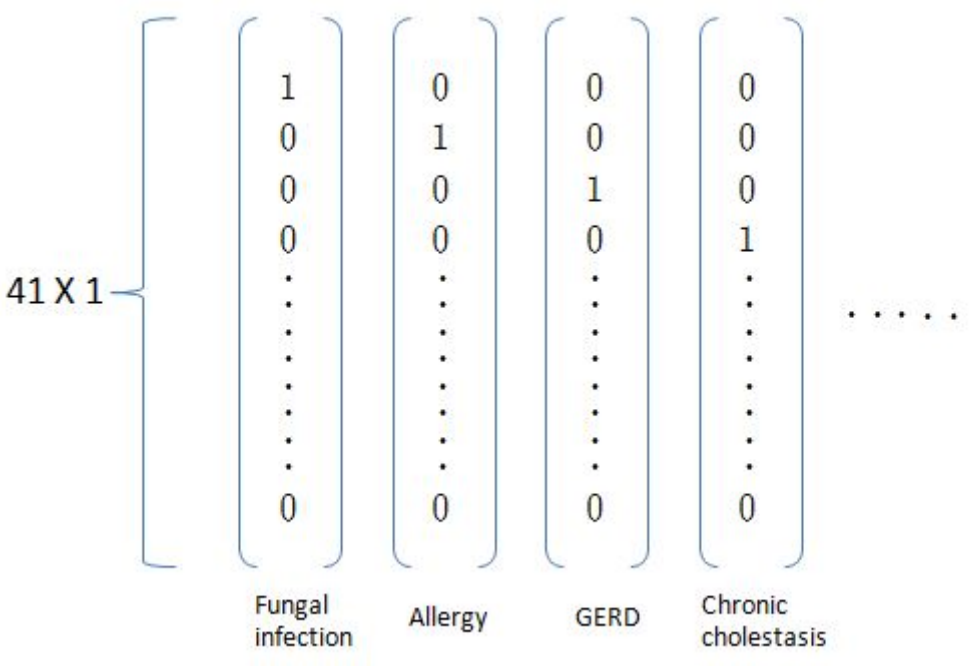
itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain	acidity
1	1	1	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0

Ce tableau présente 133 colonnes, les premières 132 colonnes sont des indicateurs relatifs à divers états symptomatiques qui sont présentés sous forme des variables binaires (0 pour non et 1 pour oui) et la dernière colonne présente la maladie associée.

2.2.1 Algorithmes utilisés

Nous choisissons d'utiliser la méthode d'apprentissage supervisé DNN (Deep Neural Network). L'algorithme DNN prend en entrée un vecteur d'éléments binaires de dimension 132x1 et il a pour sortie un vecteur 41x1, un pour chaque type de maladie qui sont considérés dans notre jeu de données. Ce vecteur de sortie a ses les éléments compris entre 0 et 1,. Dans notre DNN, nous utilisons une seule couche qui comporte 41 neurones, et on applique en sortie une fonction sigmoïde pour obtenir des valeurs entre 0 et 1.

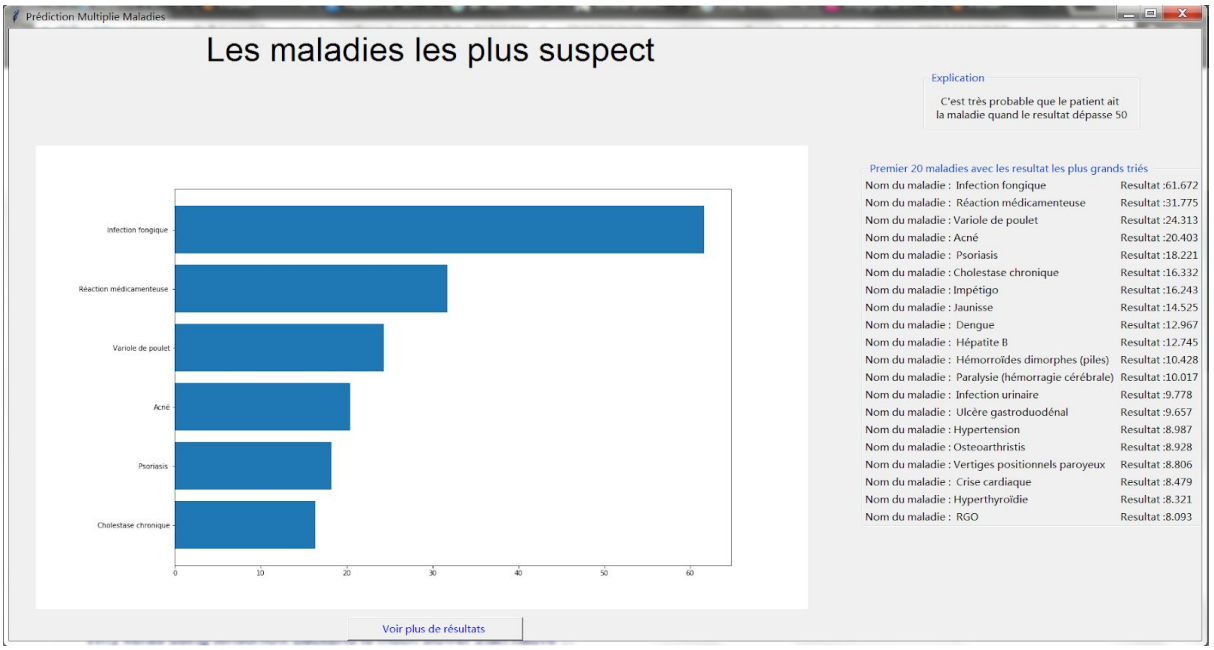




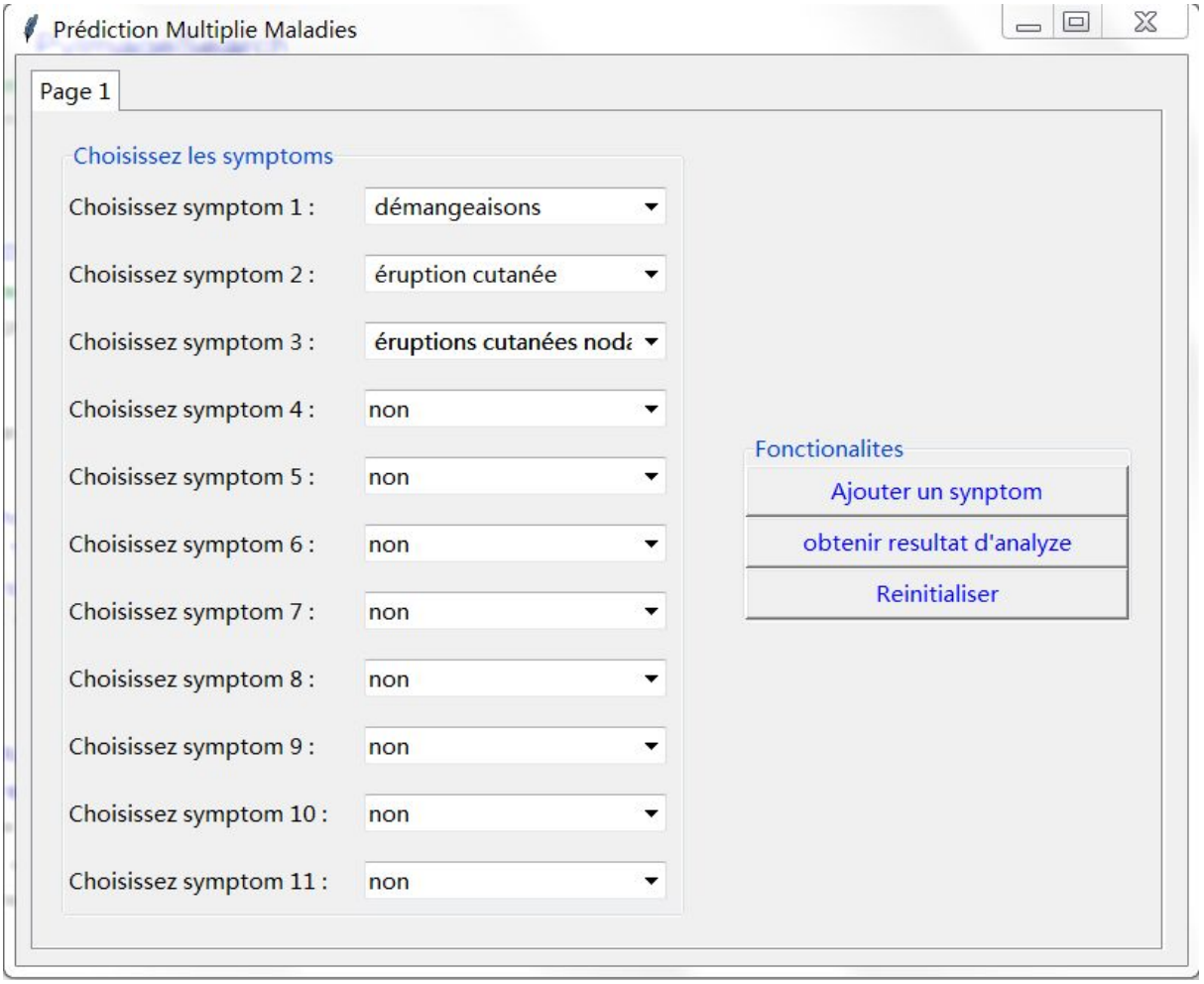
Pour établir ce modèle, nous utilisons le framework *Keras* de Python. Au total, nous avons 4920 lignes: nous utilisons les 3444 premières lignes comme training set et les 1476 lignes restantes comme testing set. Pour la partie training, nous utilisons un vecteur dont les éléments sont les variables binaires pour présenter chaque maladie. Les résultats de la précision sur le training set et le testing set sont tous les deux de 100%.

2.2.2. GUI

Pour faciliter l'utilisation de ce modèle par des médecins, nous mettons en place une GUI pour permettre d'entrer les choix des différentes conditions en entrée et visualiser les résultats d'analyse en sortie. Le module tkinter de Python est utilisé pour mettre en place cette interface.
La GUI contient deux fenêtres. La première fenêtre contient des cases qui permettent de choisir des symptômes que présentent les patients. Dans le cas où ce dernier présenterait plus de dix symptômes, l'utilisateur a la possibilité de cliquer sur *Ajouter symptôme* pour ajouter une case supplémentaire où il pourra sélectionner un symptôme de plus. L'utilisateur a également la possibilité de réinitialiser toutes les cases à l'aide du bouton *Réinitialiser*. Une fois les symptômes désirés sélectionnés, le résultat de l'analyse pourra être visualisé dans une deuxième fenêtre, en cliquant sur *Obtenir résultat d'analyse*.



Dans cette fenêtre, les résultats de l'analyse sont présentés sous la forme d'un histogramme et d'une liste, les deux triés par ordre de probabilité décroissant. L'histogramme permet une vue intuitive et comparative des maladies les plus suspectées. Un bouton *Voir plus de résultats* permet de visualiser le résultat de la prédiction pour les 41 types de maladies.



Gestion de projet

I/ Planning

Semaine / Actions	S02	S03	S04	S05	S06	S07	S08	S09	S10
Nettoyage des données									
Planifié									
Temps réel									
Choix des métriques									
Planifié									
Temps réel									
Préparation des fichiers CSV ou arff									
Planifié									
Temps réel									
Tests d'algorithmes non-supervisés									
Planifié									
Temps réel									
Développement d'algorithmes supervisés									
Planifié									
Temps réel									
Modification des features									
Planifié									
Temps réel									
Modification des algorithmes									
Planifié									
Temps réel									
Interface graphique permettant la visualisation des résultats									
Planifié									
Temps réel									

Il est notable de remarquer que le planning s'arrêtait en semaine 10, soit début mars. En effet, nous devions rendre les livrables applicatifs pour cette semaine car l'option Projet devaient les présenter en semaine 11.

Le planning initial ne prévoyait pas initialement de distinction entre une partie non-supervisée et une partie supervisée. Le traitement des données du CHU a pris plus de temps que prévu, mais l'application de clustering dessus a été rapide ensuite.
Nous sommes passé sans perdre de temps à des méthodes supervisées car les jeux de données fournis étaient déjà pré-traités, nous n'avons donc pas perdu de temps à ce niveau. Néanmoins, nous avons perdu du temps par rapport au planning lors de l'implémentation des algorithmes supervisés car une forte montée en compétence était nécessaire.

Nous avons réussi à livrer des applicatifs fonctionnels dans les délais courts imposé par l'option Projet.

II/ Répartition des tâches et difficultés rencontrées

Deux jeux de données différents été fournis pour l'approche non-supervisée. Nous nous sommes répartis les tâches en 3 groupes :

- Eric pour des recherches sur l'automated machine learning
- Cyril et Camille pour le jeu de données insuffisance cardiaque
- Zheyu, Honglu, Xinran pour le jeu de données diabète

Nous avons repris ce système pour l'approche supervisée.

Comme tout projet exploratoire, ce projet bénéficiait d'une expression du besoin limitée à la réalisation de tests sur des jeux de données. Il n'y avait personne dans le groupe ayant déjà mené des recherches en intelligence artificielle, il était donc difficile au lancement de réaliser un planning et de savoir comment aller se dérouler le projet.

Pour certains algorithmes, nous n'avons parfois pas eu bien le temps de les appréhender, et avons fonctionné en "boîte noire" : on applique les algorithmes en ayant une mauvaise idée des raisons de leur bon ou moins bon fonctionnement.
Aussi, il a parfois été difficile de traduire le vocabulaire technique médical fourni en anglais dans les jeux de données et de comprendre son sens.

Conclusion

L'application du machine learning à la prédiction d'un état de santé permet, à l'aide des différents algorithmes que nous avons implémentés, d'obtenir une première estimation de l'état de santé qui devra être validée par un professionnel de santé.

Une remarque que l'on peut noter concernant notre approche de cette problématique à l'aide du machine learning, et qui est vraie pour tous les problèmes exploitant cette méthode, est que la disponibilité des jeux de données intéressants est un facteur limitant, et limite effectivement la portée de nos résultats. En effet, le faible nombre de jeux de données disponibles et leurs nombres réduits d'entrées nous freine fortement dans nos approches, et il est par conséquent difficile de prendre du recul sur nos résultats, "vrais" pour nos jeux de données peu importants, mais peut être moins lorsque l'on introduit de la donnée supplémentaire. Avec l'essor actuel du domaine de l'intelligence artificielle et du Big Data, cette étape d'acquisition de données ne devrait en revanche plus poser de soucis dans un futur proche, du fait de la croissance toujours importante du nombre de données récupérées.

Pour une optique d'utilisation future de ces méthodes par le CHU, nous pensons qu'une étape de récupération de données par le CHU, dans le but de pallier ce manque de données, éventuellement à l'aide de capteurs et dispositifs mis en place par les équipes de l'option projet, peut permettre l'utilisation de méthodes de machine learning de façon plus sereine pour des prédictions plus précises. En particulier, la précision des modèles prédictifs est cruciale lorsqu'il s'agit de prédire l'état de santé d'un patient. Il est donc important de disposer d'un nombre conséquent de données pour espérer laisser un algorithme prédire de façon correcte un état de santé.

Bibliographie

- Jeu de données Heart Disease :
http://archive.ics.uci.edu/ml/datasets/heart+disease, fourni gracieusement par :

- Hungarian Institute of Cardiology. Budapest: Andras Janosi, M.D.
- University Hospital, Zurich, Switzerland: William Steinbrunn, M.D.
- University Hospital, Basel, Switzerland: Matthias Pfisterer, M.D.
- V.A. Medical Center, Long Beach and Cleveland Clinic Foundation:Robert Detrano, Ph.D.

- Implémentation de l'algorithme SVM sur le jeu de données Heart Disease :
https://github.com/bveber/Heart_Disease_Prediction

- Ressources SVM :
https://fr.wikipedia.org/wiki/Machine_%C3%A0_vecteurs_de_support#R%C3%A9sum%C3%A9_intuitif
http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC

- Ressources Bayes Naif :
https://fr.wikipedia.org/wiki/Classification_na%C3%AFve_bay%C3%A9sienne
http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.html

-Ressources Boost :
https://en.wikipedia.org/wiki/Gradient_boosting
http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

-Ressources automated machine learning :
https://www.kdnuggets.com/2017/01/current-state-automated-machine-learning.html
https://www.cs.ubc.ca/labs/beta/Projects/autoweka/
https://www.cs.waikato.ac.nz/ml/weka/documentation.html
https://www.cs.ubc.ca/labs/beta/Projects/autoweka/manual.pdf
http://www.cs.ubc.ca/labs/beta/Projects/SMAC/
http://chem-eng.utoronto.ca/~datamining/dmc/zeror.htm
https://pdfs.semanticscholar.org/8a29/edb6b9e3863d40ef5223ea35fa566826d1e4.pdf

-Ressources Keras library :
https://keras.io/#keras-the-python-deep-learning-library
https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/

-Ressources GUI tkinter module :
https://openclassrooms.com/courses/apprenez-a-programmer-en-python/des-interfaces-graphiques-avec-tkinter
https://docs.python.org/2/library/tkinter.html

Annexes

Dossier : *Livrables/Non-supervisé/Insuffisance-Cardiaque* :
Ce dossier contient le jeu de données Insuffisance Cardiaque fourni par le CHU a plusieurs niveau de pré-traitement.
insuff_cardiaque-preprocessing_TCD.xlsx : fichier contenant le jeu de données original et le TCD
insuff_cardiaque-preprocessing_incorrect_values.csv : fichier contenant environ 3100 lignes traitées mais avec encore quelques valeurs erronées
insuff_cardiaque-preprocessed.arff : fichier contenant environ 3000 lignes traitées en ayant enlevé les valeurs aberrantes.

Dossier : *Livrables/Non-supervisé/Diabete* :
Ce dossier contient le jeu de données Diabete fourni par le CHU pré-traité.
diabete_traite.csv : fichier des données traitées
diabete_traite.py : script de traitement du jeu de données original

Dossier : *Livrables/Supervisé/Heart-Disease :*
Ce dossier contient les algorithmes de prédiction de maladie cardiaque.

Dossier : *Livrables/Supervisé/Multi-Diseases :*
Ce dossier contient les algorithmes de prédiction multiple diseases.