

REPUBLIQUE DE DJIBOUTI

جمهورية جيبوتي

Ministère de l'Enseignement Supérieur et
de la Recherche

وزارة التعليم العالي والبحث العلمي

UNIVERSITE DE DJIBOUTI



جامعة جيبوتي

THEME : CLASSIFICATION H20

Master 2 Data Science

Encadré par : M. Engelbert MEPHU NGUIFO

Réaliser par :

Nouradine ABDI MAHAMOUD
Ali HASSAN ALI



Année Universitaire : 2020/2021

PLAN

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ALGORITHMES

INTRODUCTION GENERAL

PLANNIFICATION DU PROJET

I. H2O

- a) INTRODUCTION
- b) QU'EST-CE QUE H2O ?
- c) INSTALLATION DE H2O SOUS PYTHON
- d) ARCHITECTURE DE H2O
- e) CONCLUSION

II. H2O POUR LE MACHINE LERANING

- 1. INTROCTION
- 2. DATA PREPARATION
- 3. METHODES SUPERVISEES
- 4. FONCTIONNEMENT DE FLOW
- 5. CONCLUSION

III. ETUDE DES METHODES SUPERVISEES DE H20

- 1. INTRODUCTION
- 2. DONNEE
- 3. RANDOM BOOSTING
- 4. GRADIENT BOOSTING
- 5. NAÏVE BAYES

6. PERCEPTRON SIMPLE
7. H2OAutoML
8. CONCLUSION

IV. REALISATION

1. INTRODUCTION
2. ALGORITHME K-NN POUR TRAITE LE BIG DATA
 - a) PROCESSUS D'APPRENTISSAGE
 - b) PROCESSUS DE TEST
3. EVALUATION ET RESULTAT
 - a. RESULTATS EXPERIMENTAUX DE LC-KNN ET RC-KNN AVEC DIFFERENTES VALEURS DE M
 - b. POUR DETERMINER LE PARAMETRE DE K
 - c. COMPARAISON DES PERFORMANCES DE KNN, RC-KNN ET LC-KNN
4. CONCLUSION

CONCLUSION GENERALE

REFERENCES

IMPLEMENTATION DE L'ALGORITHME DE LSC

❖ Liste des figures

Figure 0 : Planning de notre projet.

Figure 1.1 : Architecture de h2o.

Figure 3.1 : Random Forest - Evolution de la perte en fonction du nombre d'arbres.

Figure 3.2 : Gradient Boosting - Evolution de la perte en fonction du nombre d'arbres.

Figure 4.1 : Précision de classification de LC-kNN sur neuf ensembles de données avec k différents.

❖ Liste des tableaux

Tableau 4.1 : Précision de classification et coût du temps sur l'ensemble de données USPS à différentes valeurs de m.

Tableau 4.2 : Précision de classification et coût du temps sur l'ensemble de données MNIST à différentes valeurs de m.

Tableau 4.3 : Précision de classification et coût du temps sur l'ensemble de données GISETTE à différentes valeurs de m.

Tableau 4.4 : Précision de classification et coût du temps sur l'ensemble de données LETTER à différentes valeurs de m.

Tableau 4.5 : Précision de classification et coût du temps sur l'ensemble de données PENDIGITS à différentes valeurs de m.

Tableau 4.6 : Précision de classification et coût du temps sur l'ensemble de données SATIMAGE à différentes valeurs de m.

Tableau 4.7 : Précision de classification et coût du temps sur l'ensemble de données ADNC à différentes valeurs de m.

Tableau 4.8 : Précision de classification et coût du temps sur l'ensemble de données psMCI à différentes valeurs de m.

Tableau 4.9 : Précision de classification et coût du temps sur l'ensemble de données MCINC à différentes valeurs de m.

Tableau 4.10 : Précision de classification et coût en temps de trois algorithmes sur neuf ensembles de données.

❖ Liste des algorithmes

Algorithme 4.1 : L'algorithme de LSC.

Algorithme 4.2 : L'algorithme de LC-KNN.

INTRODUCTION GENERAL

Dans le cadre de notre études de mastère au sein de la Faculté d'ingénieurs, nous avons été amenés à améliorer nos compétence et notre savoir-faire par divers moyens imposés par notre établissement tels que : les mini-projets, les projets de travail étude et de recherche, les projets de fin d'études

Le présent document constitue le fruit de notre travail accompli dans le cadre de ce projet, dont l'objectif est d'étudier le package H2O sous Python pour l'efficacité de l'algorithme de classification pour le Big Data.

« H2O » est une plate-forme JAVA de machine learning. Elle propose des outils pour la manipulation et la préparation de données, des algorithmes de modélisation, supervisées, non-supervisées ou de réduction de dimensionnalité. Nous pouvons accéder à ses fonctionnalités en mode client-serveur via différents langages de programmation avec le mécanisme des API (application programming interface). Nous nous appuierons sur Python.

Le fait que H2O propose des solutions efficaces est un véritable atout parce que, même aujourd'hui, elles ne sont pas très présentes encore dans les outils de data science.

L'apprentissage supervisé (ou classification) consiste à construire un modèle basé sur un jeu d'apprentissage et à l'utiliser pour classer des données nouvelles. Cette technique est utilisée dans plusieurs applications telles que les diagnostics médicaux, la prédition des pannes et la détection des opinions trompeuses dans les réseaux sociaux.

K plus proche voisins (kNN) est un algorithme d'apprentissage supervisé efficace et a été développé avec succès dans des applications réelles. Il est naturel d'adapter la méthode kNN aux ensembles de données à grande échelle. Les résultats expérimentaux montrent que la classification kNN proposée fonctionne bien en termes de précision et d'efficacité

Ce rapport comporte quatre chapitres : Le premier chapitre présente H2O. Le deuxième chapitre présente H2O pour la machine learning. Troisième chapitre présente l'étude des méthodes supervisées de H2O et le quatrième présente la réalisation du projet. Mon rapport sera achevé par une conclusion générale.

Planning de projet:

Planification du projet

Deroulément du projet

Tâche	DÉBUT DU TACHE	FIN DU TACHE	NOMBRE DE JOUR
1- H2O	24/12/2020	03/01/2021	10
2- H2O POUR LE MACHINE LERANING	04/01/2021	10/01/2021	6
3- ETUDE DES METHODES SUPERVISEES DE H2O	11/01/2021	19/01/2021	8
4- REALISATION	20/01/2021	29/01/2021	9
		TOTAL DE NOMBRE DE JOUR	33

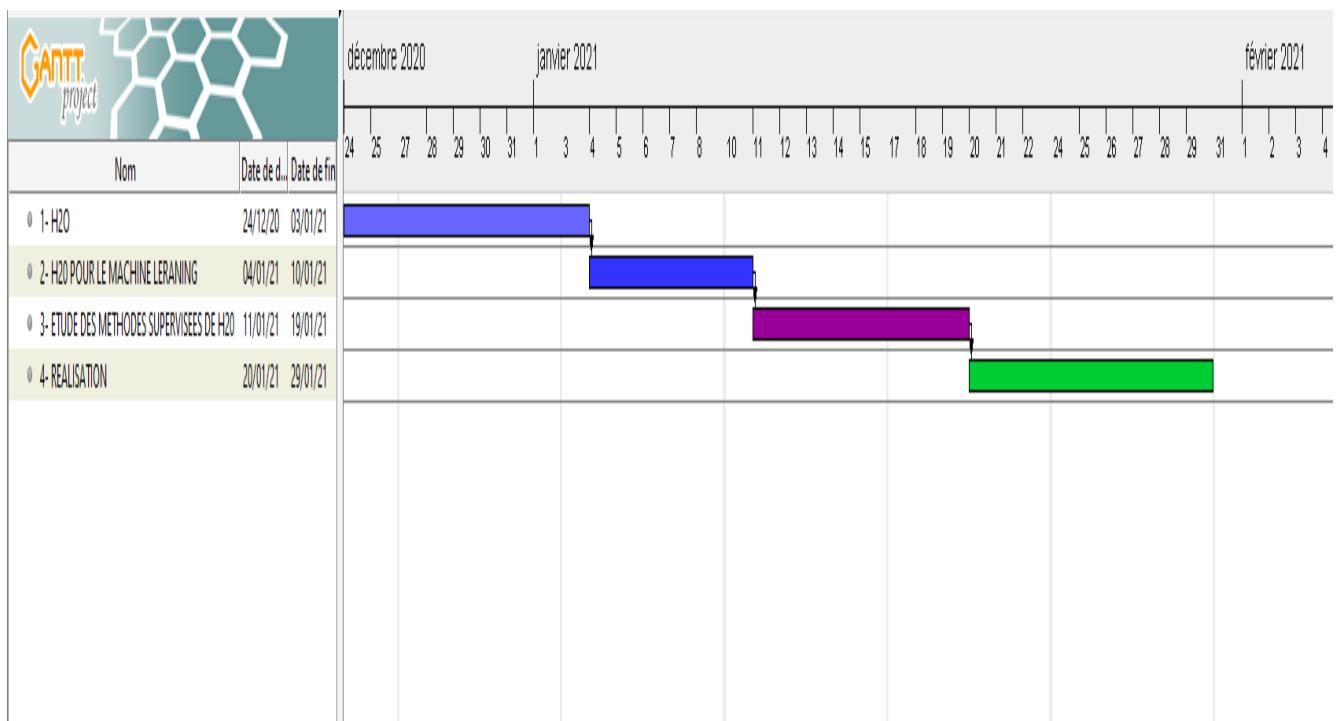
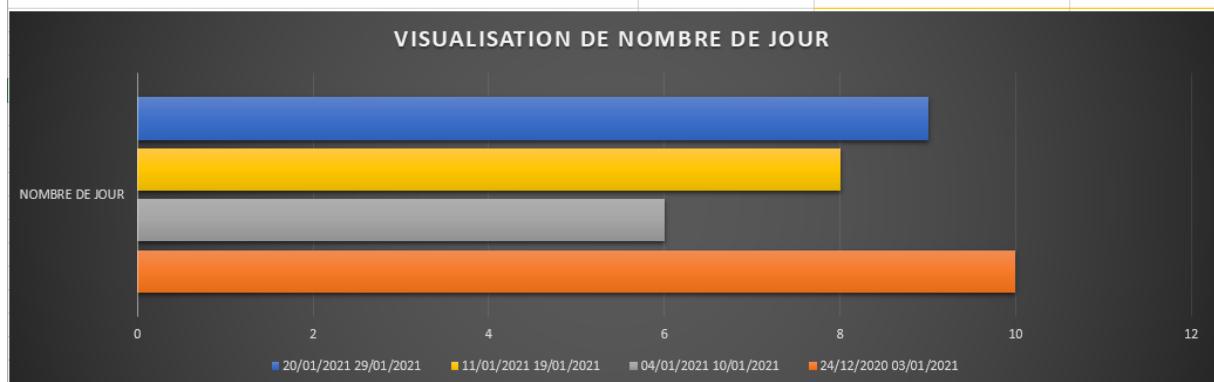


Figure 0 : planning de notre projet.

I. H2O

2. INTRODUCTION

Introduction à l'utilisation de H2O à partir de python pour exécuter des algorithmes d'apprentissage dans différents environnements : simple poste avec plusieurs processeurs, serveur, cluster, cluster avec données distribuées (Spark, Hadoop, Amazon EC2). Plus d'informations sur le système de H2O et des algorithmes de H2O sont disponibles sur le site web de H2O à l'adresse : <http://docs.h2o.ai>.

3. QU'EST-CE QUE H2O?

En utilisant la compression en mémoire, H2O gère des milliards de lignes de données en mémoire, même avec un petit cluster. Pour permettre aux non-ingénieurs de créer plus facilement des flux de travail analytiques complets, la plate-forme H2O comprend des interfaces pour Python, R, Scala, Java, JSON et CoffeeScript / JavaScript, ainsi qu'une interface Web intégrée, Flow. H2O est conçu pour s'exécuter en mode autonome, sur Hadoop ou dans un cluster Spark, et se déploie généralement en quelques minutes.

H2O est un logiciel basé sur Java pour la modélisation de données et le calcul général. Le logiciel H2O est beaucoup de choses, mais le but principal de H2O est en tant que moteur de traitement distribué, parallèle, en mémoire. Il existe deux niveaux de parallélisme:

- dans le nœud
- à travers (ou entre) nœuds

L'objectif de H2O est permettre une simple mise à l'échelle horizontale d'un problème donné afin de produire une solution plus rapidement. Le paradigme conceptuel MapReduce (AKA «diviser et conquérir et combiner»), ainsi qu'une bonne structure d'application simultanée, permettent ce type de mise à l'échelle dans H2O.

H2O comprend de nombreux algorithmes d'apprentissage automatique courants, tels que la modélisation linéaire généralisée (régression linéaire, régression logistique, etc.), Naïve Bayes, l'analyse des composants principaux (ACP), le clustering k-means et word2vec. H2O implémente les meilleurs algorithmes de sa catégorie à grande échelle, tels que la forêt aléatoire distribuée, le renforcement du gradient et l'apprentissage en profondeur. H2O inclut également une méthode d'ensembles empilés, qui trouve la combinaison optimale d'une collection d'algorithmes de prédiction à l'aide d'un

processus appelé «empile». Avec H2O, les clients peuvent créer des milliers de modèles et comparer les résultats pour obtenir les meilleures prévisions.

4. INSTALLATION DE H2O SOUS PYTHON

Le moyen le plus simple d'installer directement H2O consiste à utiliser un package Python. Pour charger un package H2O récent à partir de PyPI, exécutez:

- ✓ [pip installer h2o](#)

Une fois H2O installé, vérifiez l'installation:

```
Entrée [1]: import h2o
# Démarrez H2O sur votre machine Locale
h2o.init()

Checking whether there is an H2O instance running at http://localhost:54321 .... not found.
Attempting to start a local H2O server...
; Java HotSpot(TM) Client VM (build 25.45-b02, mixed mode, sharing)

C:\ProgramData\Anaconda3\lib\site-packages\h2o\backend\server.py:385: UserWarning: You have a 32-bit version of Java. H2O works best with 64-bit Java.
Please download the latest 64-bit Java SE JDK from Oracle.

warn(" You have a 32-bit version of Java. H2O works best with 64-bit Java.\n"

Starting server from C:\ProgramData\Anaconda3\lib\site-packages\h2o\backend\bin\h2o.jar
Ice root: C:\Users\ADMINI~1\AppData\Local\Temp\tmp69c16tfx
JVM stdout: C:\Users\ADMINI~1\AppData\Local\Temp\tmp69c16tfx\h2o_Administrateur_started_from_python.out
JVM stderr: C:\Users\ADMINI~1\AppData\Local\Temp\tmp69c16tfx\h2o_Administrateur_started_from_python.err
Server is running at http://127.0.0.1:54321
Connecting to H2O server at http://127.0.0.1:54321 ... successful.

      H2O_cluster_uptime:          05 secs
      H2O_cluster_timezone:       Europe/Paris
      H2O_data_parsing_timezone:   UTC
      H2O_cluster_version:        3.32.0.2
      H2O_cluster_version_age:    1 month and 24 days
      H2O_cluster_name:          H2O_from_python_Administrateur_404ur6
      H2O_cluster_total_nodes:    1
      H2O_cluster_free_memory:    247.5 Mb
      H2O_cluster_total_cores:    0
      H2O_cluster_allowed_cores:  0
      H2O_cluster_status:         accepting new members, healthy
      H2O_connection_url:        http://127.0.0.1:54321
      H2O_connection_proxy:       {"http": null, "https": null}
      H2O_internal_security:     False
      H2O_API_Extensions:        Amazon S3, Algos, AutoML, Core V3, TargetEncoder, Core V4
      Python_version:            3.7.4 final
```

5. ARCHITECTURE DE H2O

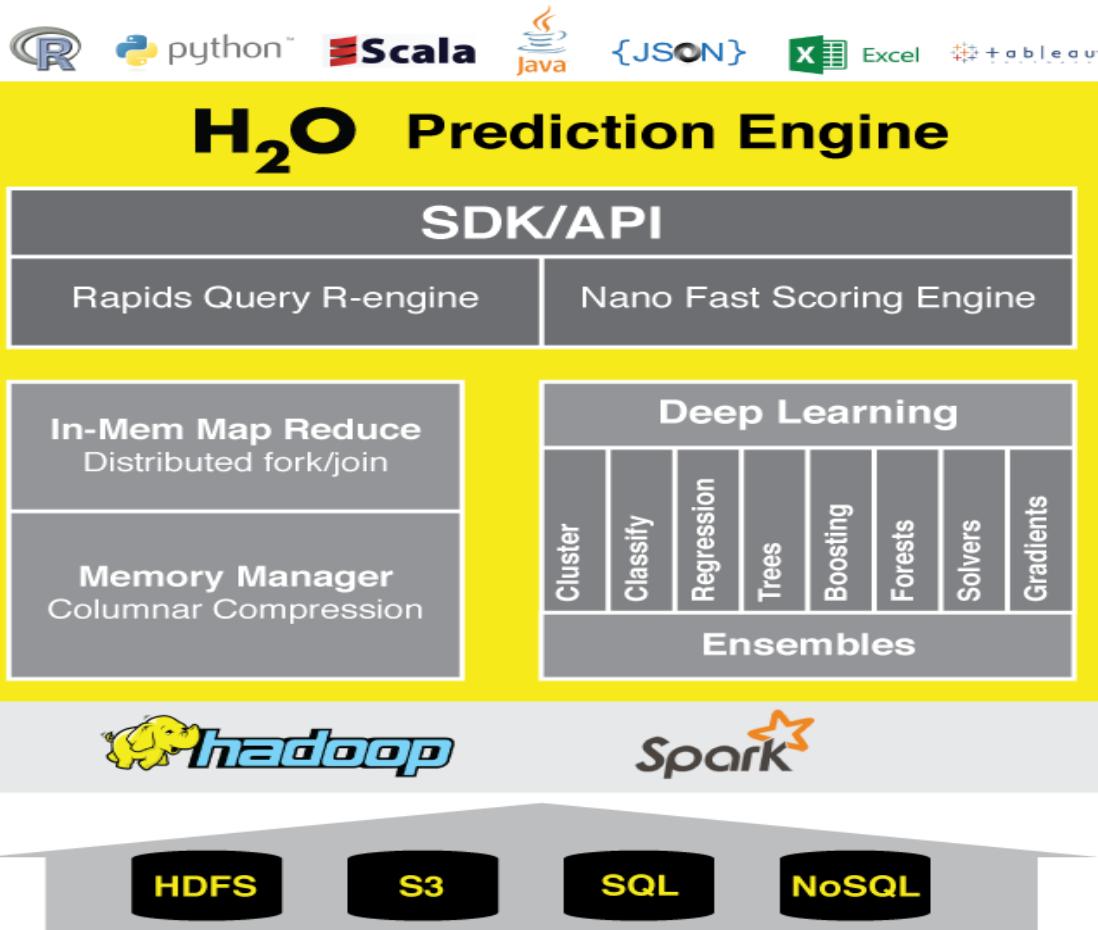


Figure 1.1 : Architecture de h2o.

6. CONCLUSION

Dans ce chapitre nous avons présenté brièvement le H2O. H2O prend en charge de nombreux algorithmes d'apprentissage automatique couramment utilisés. On peut distinguer deux niveaux de parallélisme de H2O:

- dans le nœud
- à travers (ou entre) nœuds

On présente aussi installation de package H2O sous python. H2O permet d'importer des données à partir de plusieurs sources et dispose d'un moteur de calcul rapide, évolutif et distribué écrit en Java.

II. H2O POUR MACHINE LEARNING

1. INTRODUCTION

H2O est une plateforme machine learning open source à laquelle font confiance les data scientifiques et les praticiens de machine learning. Il dispose d'API disponibles en R, Python, Scala, ainsi que d'une interface point et clic basée sur le Web appelée Flow. H2O prend en charge les algorithmes statistiques et machine learning les plus utilisés, y compris les machines à gradient boosté, les modèles linéaires généralisés, deep learning et bien d'autres aussi.

2. PREPARATION DES DONNEES

Avant de voir comment intégrer des données dans H2O, nous avons besoin de prendre un pas en arrière et réfléchir à ce que nous pourrions avoir à faire avec nos données. En gros, cela signifie:

- ✓ Divisez-le en deux ou trois ensembles de données (train / valide / test)
- ✓ Marquer les types de données de champ (numérique / entier / enum)
- ✓ Champs de nom
- ✓ Trier les valeurs manquantes et autres mauvaises données
- ✓ Fusionner les ensembles de données
- ✓ Ajouter de nouvelles colonnes de données

Pour chacun d'entre eux, vous avez une décision importante, avant de charger les données dans H2O, ou après. Plus vos données sont volumineuses, plus vous devez réfléchir à cette décision, et vous devrez prendre en compte des éléments tels que votre budget, vos délais, et s'il s'agit d'un événement ponctuel ou si vous obtenez de nouvelles données chaque mois, chaque jour, chaque heure, chaque milliseconde...

En règle générale, pour utiliser les algorithmes de machine learning H2O, les données doivent être dans le cluster H2O. Cela peut parfois vous défavoriser, mais c'est ce qui vous permet de gérer des ensembles de données volumineux qui ne tiennent pas sur une seule machine de votre cluster, et encore moins sur le petit ordinateur portable sur lequel vous exécutez le client. H2O propose plusieurs façons d'importer des données.

Nous importons et démarrons H2O sur la même machine que le processus Python en cours d'exécution:

```
import h2o
# Démarrer H2O sur votre machine Locale
h2o.init()
```

Pour vous connecter à un cluster H2O :

```

# Connecter à un cluster H2O
h2o.init(ip="127.0.0.1", port=54321)

Checking whether there is an H2O instance running at http://127.0.0.1:54321 . connected.

      H2O_cluster_uptime:          8 mins 28 secs
      H2O_cluster_timezone:        Europe/Paris
      H2O_data_parsing_timezone:   UTC
      H2O_cluster_version:         3.32.0.2
      H2O_cluster_version_age:    1 month and 26 days
      H2O_cluster_name:           H2O_from_python_Administrateur_zgyj6n
      H2O_cluster_total_nodes:    1
      H2O_cluster_free_memory:    88.8 Mb
      H2O_cluster_total_cores:    4
      H2O_cluster_allowed_cores:  4
      H2O_cluster_status:         locked, healthy
      H2O_connection_url:         http://127.0.0.1:54321
      H2O_connection_proxy:       {"http": null, "https": null}
      H2O_internal_security:      False
      H2O_API_Extensions:         Amazon S3, Algos, AutoML, Core V3,
                                  TargetEncoder, Core V4
      Python_version:             3.7.4 final

```

3. METHODE SUPERVISEE

Modèles linéaires généralisés (GLM) : Fournit une généralisation flexible de la régression linéaire ordinaire pour les variables de réponse avec des modèles de distribution d'erreur autres qu'une distribution gaussienne (normale). GLM unifie divers autres modèles statistiques, y compris Poisson, linéaire, logistique et autres lors de l'utilisation des régularisations `1 et` 2.

Forêt aléatoire distribuée: fait la moyenne de plusieurs arbres de décision, chacun étant créé sur différents échantillons aléatoires de lignes et de colonnes. Il est facile à utiliser, non linéaire et fournit des informations sur l'importance de chaque prédicteur dans le modèle, ce qui en fait l'un des algorithmes les plus robustes pour les données bruyantes.

Gradient Boosting Machine (GBM): produit un modèle de prédiction sous la forme d'un ensemble de modèles de prédiction faibles. Il construit le modèle par étapes généralisées en autorisant une fonction de perte différentiable arbitraire. C'est l'une des méthodes les plus puissantes disponibles aujourd'hui.

Deep Learning: Modélise des abstractions de haut niveau dans les données à l'aide de transformations non linéaires dans une méthode couche par couche. L'apprentissage en profondeur est un exemple d'apprentissage supervisé, qui peut utiliser des données non étiquetées que d'autres algorithmes ne peuvent pas.

Naïve Bayes: Génère un classificateur probabiliste qui prend la valeur d'une caractéristique particulière est sans rapport avec la présence ou l'absence de tout autre élément, compte tenu de la variable de classe. Il est souvent utilisé dans la catégorisation de texte.

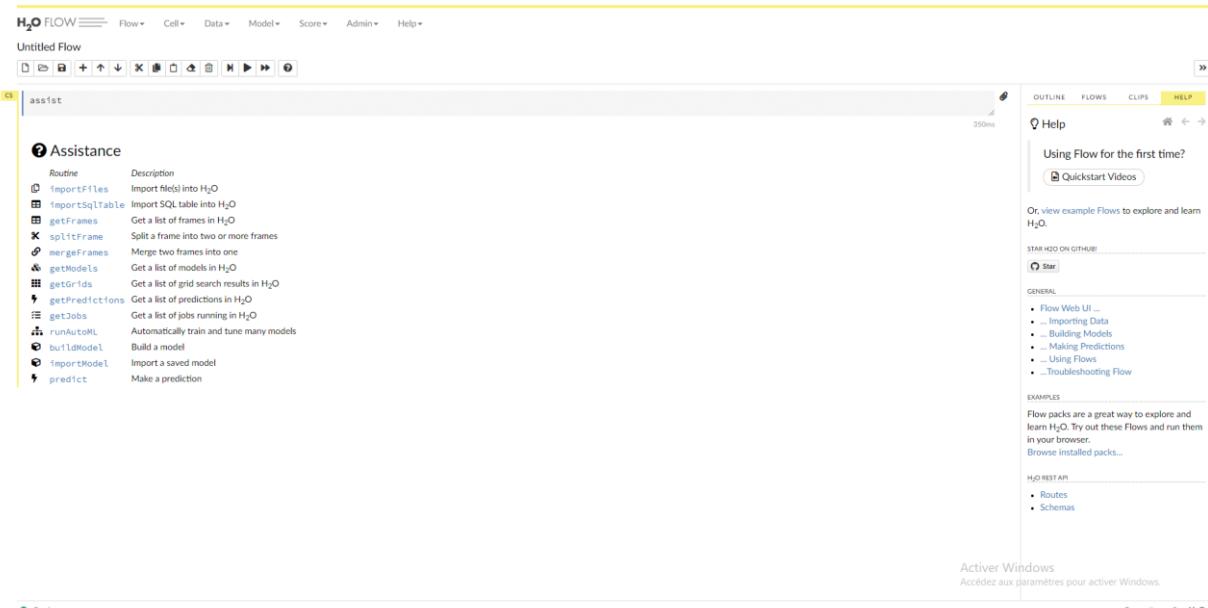
Stacked d'ensemble: L'utilisation de plusieurs modèles construits à partir des algorithmes différents, Stacked ensemble trouve la combinaison optimale d'un ensemble d'algorithme de prédiction en utilisant un processus appelé « empilement ».

4. FONCTIONNEMENT DE FLOW

Flow est le nom de l'interface Web qui fait partie de H2O. Il s'agit en fait d'un autre client, écrit cette fois en CoffeeScript (un langage de type JavaScript), effectuant les mêmes appels de service Web au back-end H2O que les clients Python. Il est complet, ce qui signifie que nous pouvons effectuer toutes les opérations suivantes:

- ❖ Afficher les données que vous avez téléchargées via votre client
- ❖ Télécharger directement les données
- ❖ Afficher les modèles que vous avez créés via votre client
- ❖ Créer des modèles directement
- ❖ Afficher les prédictions que vous avez générées via votre client
- ❖ Exécuter des prédictions directement

Nous pouvons le trouver en pointant notre navigateur vers <http://127.0.0.1:54321>. Bien sûr, si nous démarrons H2O sur un port non standard, modifiez le bit: 54321, et si nous voulons accéder à un cluster H2O distant, remplacez le bit 127.0.0.1 par le nom de serveur de n'importe quel nœud du cluster. Lorsque nous chargeons le Flow pour la première fois, nous verrons le menu Flow, comme illustré à la Figure 1-4.



Importons la donnée que nous avons faite dans l'exemple Python. Depuis l'écran de démarrage, cliquez sur le lien «importFiles», choisissez Data puis Import Files. Collez l'emplacement du fichier csv dans la zone de recherche, puis sélectionnez-le, puis enfin cliquez sur le bouton Importer; voir la figure 1-5.

Import Files

Search: C:\Users\Administrateur\Documents\H2O ipynb\cardio.csv

Search Results: (All files added)

Selected Files: 1 file selected: Clear All

X C:\Users\Administrateur\Documents\H2O ipynb\cardio.csv

Actions: Import

CS importFiles ["C:\\Users\\Administrateur\\Documents\\H2O ipynb\\cardio.csv"]

Cloud 1 / 1 files imported.

Files C:\Users\Administrateur\Documents\H2O ipynb\cardio.csv

Actions Parse these files...

Cliquez maintenant sur «Analyser ces fichiers, «EN : Parse These Files» », et cela nous donne la possibilité de personnaliser les paramètres comme indiqué dans la figure 1-6, mais dans ce cas, il suffit d'accepter les valeurs par défaut.

Setup Parse

PARSE CONFIGURATION

Sources C:\Users\Administrateur\Documents\H2O ipynb\cardio.csv

ID cardio.hex

Parser CSV

Separator ;

Column Headers Auto

First row contains column names (radio button selected)

First row contains data

Options Enable single quotes as a field quotation character (checkbox)

Delete on done (checkbox checked)

EDIT COLUMN NAMES AND TYPES

	1d	Numeric	0	1	2	3	4	8	9	12	13
1	age	Numeric	18393	20228	18857	17623	17474	21914	22113	22584	17668
2	gender	Numeric	2	1	1	2	1	1	1	2	1
3	height	Numeric	168	156	165	169	156	151	157	178	158
4	weight	Numeric	62.0	85.0	64.0	82.0	56.0	67.0	93.0	95.0	71.0
5	ap_hi	Numeric	110	140	130	150	100	120	130	130	110
6	ap_lo	Numeric	80	90	70	100	60	80	80	90	70
7	cholesterol	Numeric	1	3	3	1	1	2	3	3	1
8	gluc	Numeric	1	1	1	1	1	2	1	3	1
9	smoke	Numeric	0	0	0	0	0	0	0	0	0
10	alco	Numeric	0	0	0	0	0	0	0	0	0
11	active	Numeric	1	1	0	1	0	0	1	1	1
12	cardio	Numeric	0	1	1	0	0	0	1	0	0

← Previous page → Next page

Parse

Ready

Activer V Accédez au

Si nous choisissez «getFrames» dans le menu principal, soit après avoir effectué les étapes précédentes, soit après avoir chargé les données de Python, nous verrons une entrée disant «cardio.hex» et qu'elle comporte 70000 lignes et 13 colonnes. Si nous avons cliqué sur le lien «cardio.hex», nous verrons la Figure 1-7.

H2O FLOW Flow ▾ Cell ▾ Data ▾ Model ▾ Score ▾ Admin ▾ Help ▾

Untitled Flow

splitFrame

51ms

Split Frame

Frame: cardio.hex

Splits: Ratio

0.80 cardio.hex_0.80

0.20 cardio.hex_0.20

Add a new split

Seed: 223326

X Create

splitFrame "cardio.hex", [0.8], ["cardio.hex_0.80","cardio.hex_0.20"], 223326

97ms

Split Frames

Type	Key	Ratio
cardio.hex_0.80		0.8
cardio.hex_0.20		0.1999999999999999

getFrameSummary "cardio.hex_0.80"

327ms

H2O FLOW Flow ▾ Cell ▾ Data ▾ Model ▾ Score ▾ Admin ▾ Help ▾

Untitled Flow

splitFrame

51ms

cardio.hex_0.80

Actions: View Data ▾ Split ▾ Build Model ▾ Run AutoML ▾ Predict ▾ Download ▾ Export ▾ Delete

Rows: 56176 Columns: 13 Compressed Size: 808KB

COLUMN SUMMARIES

label	type	Missing	Zeros	+Inf	-Inf	min	max	mean	sigma	cardinality	Actions
id	int	0	1	0	0	99999.0	49974.8783	2816.2857	.	Convert to enum	
age	int	0	0	0	0	10798.0	23713.0	19469.3207	2467.2340	Convert to enum	
gender	int	0	0	0	0	1.0	2.0	1.3591	0.4770	Convert to enum	
height	int	0	0	0	0	57.0	250.0	164.3558	8.2221	Convert to enum	
weight	real	0	0	0	0	21.0	200.0	74.2135	14.3862	.	
ap_hi	int	0	0	0	0	-120.0	16020.0	129.0609	164.3974	Convert to enum	
ap_lo	int	0	14	0	0	-70.0	11000.0	96.5760	189.5880	Convert to enum	
cholesterol	int	0	0	0	0	1.0	3.0	1.3680	0.6806	Convert to enum	
gluc	int	0	0	0	0	1.0	3.0	1.2273	0.5734	Convert to enum	
smoke	int	0	51246	0	0	0	1.0	0.0878	0.2829	Convert to enum	
alco	int	0	53140	0	0	0	1.0	0.0540	0.2261	Convert to enum	
active	int	0	11011	0	0	0	1.0	0.0040	0.1970	Convert to enum	
cardio	int	0	28096	0	0	0	1.0	0.4999	0.5000	Convert to enum	

← Previous 20 Columns → Next 20 Columns

CHUNK COMPRESSION SUMMARY

size	number_of_rows	number_of_chunks_per_column	number_of_chunks
127.0.0.1:54321	808,5 KB	56176,0	16,0
mean	808,5 KB	56176,0	16,0
min	808,5 KB	56176,0	16,0
max	808,5 KB	56176,0	16,0
stddev	0 B	0	0
total	808,5 KB	56176,0	16,0

Activé V
Accédez au

Ready

H2O FLOW Flow Cell Data Model Score Admin Help

Untitled Flow

cardio.hex_0.80

DATA

Previous 20 Columns Next 20 Columns

Row	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
1	0	18393.0	2.0	168.0	62.0	110.0	80.0	1.0	1.0	0	0	1.0	0
2	1.0	20228.0	1.0	156.0	85.0	140.0	90.0	3.0	1.0	0	0	1.0	1.0
3	2.0	18857.0	1.0	165.0	64.0	130.0	70.0	3.0	1.0	0	0	0	1.0
4	3.0	17623.0	2.0	169.0	82.0	150.0	100.0	1.0	1.0	0	0	1.0	1.0
5	8.0	21914.0	1.0	151.0	67.0	120.0	80.0	2.0	2.0	0	0	0	0
6	9.0	22113.0	1.0	157.0	93.0	130.0	80.0	3.0	1.0	0	0	1.0	0
7	12.0	22584.0	2.0	178.0	95.0	130.0	90.0	3.0	3.0	0	0	1.0	1.0
8	13.0	17668.0	1.0	158.0	71.0	110.0	70.0	1.0	1.0	0	0	1.0	0
9	14.0	19834.0	1.0	164.0	68.0	110.0	60.0	1.0	1.0	0	0	0	0
10	15.0	22530.0	1.0	169.0	80.0	120.0	80.0	1.0	1.0	0	0	1.0	0
11	16.0	18815.0	2.0	173.0	60.0	120.0	80.0	1.0	1.0	0	0	1.0	0
12	18.0	14791.0	2.0	165.0	60.0	120.0	80.0	1.0	1.0	0	0	0	0
13	21.0	19809.0	1.0	158.0	78.0	110.0	70.0	1.0	1.0	0	0	1.0	0
14	24.0	16782.0	2.0	172.0	112.0	120.0	80.0	1.0	1.0	0	0	0	1.0
15	28.0	17482.0	1.0	154.0	68.0	100.0	70.0	1.0	1.0	0	0	0	0
16	29.0	21755.0	2.0	162.0	56.0	120.0	70.0	1.0	1.0	1.0	0	1.0	0
17	30.0	19778.0	2.0	163.0	83.0	120.0	80.0	1.0	1.0	0	0	1.0	0
18	31.0	21413.0	1.0	157.0	69.0	130.0	80.0	1.0	1.0	0	0	1.0	0
19	32.0	23946.0	1.0	158.0	90.0	145.0	85.0	2.0	2.0	0	0	1.0	1.0
20	33.0	23376.0	2.0	156.0	45.0	110.0	60.0	1.0	1.0	0	0	1.0	0
21	35.0	16608.0	1.0	170.0	68.0	150.0	90.0	3.0	1.0	0	0	1.0	1.0
22	36.0	14453.0	1.0	153.0	85.0	130.0	100.0	2.0	1.0	0	0	1.0	0
23	37.0	19559.0	1.0	156.0	59.0	130.0	90.0	1.0	1.0	0	0	1.0	0

Previous 20 Columns Next 20 Columns

À la suite de l'exemple précédent, cliquez sur «former», puis sur «Build Model». Dans les algorithmes, nous choisissons Deep Learning.

Nous chargeons le paramètres apparaissent. Il nous suffit de définir un model dans le menu déroulant "response_column", nous choisissons le variable "cardio". Les valeurs par défaut pour tout le reste sont bonnes, alors faites défiler vers le bas et cliquez sur "Créer un modèle". Nous devrions voir quelque chose comme la sortie de la figure 1-8.

H2O FLOW Flow Cell Data Model Score Admin Help

Untitled Flow

cardio.hex_0.80

Build a Model

Select an algorithm: Deep Learning

PARAMETERS

model_id: deeplearning-6b8c8372-b02a Destination id for this model; auto-generated if not specified.

training_frame: cardio.hex_0.80 Id of the training data frame.

validation_frame: (Choose...) Id of the validation data frame.

nfolds: 0 Number of folds for K-fold cross-validation (0 to disable or >= 2).

response_column: cardio Response variable column.

ignored_columns: Search... Names of columns to ignore for training.

Only show columns with more than 0 % missing values.

Ready

Activer V Accédez au

H2O FLOW Flow ▾ Cell ▾ Data ▾ Model ▾ Score ▾ Admin ▾ Help ▾

Untitled Flow

Flow Editor

Only show columns with more than 0 % missing values.

ADVANCED

- ignore_const_cols** Ignore constant columns.
- activation** Rectifier
- hidden** 200, 200
- epochs** 10
- variable_importances** Compute variable importances for input features (Gedeon method) - can be slow for large networks.

GRID?

- fold_column** (Choose...)
- score_each_iteration**
- weights_column** (Choose...)
- offset_column** (Choose...)
- checkpoint**
- use_all_factor_levels**
- standardize**
- train_samples_per_iteration** -2
- adaptive_rate**
- input_dropout_ratio** 0
- L1** 0
- L2** 0
- loss** Automatic
- distribution** AUTO
- huber_alpha** 0.9
- score_interval** 5
- score_trainings_samples** 10000

GRID?

Ready

H2O FLOW Flow ▾ Cell ▾ Data ▾ Model ▾ Score ▾ Admin ▾ Help ▾

Untitled Flow

Flow Editor

EXPERT

- score_training_samples** 10000
- score_duty_cycle** 0.1
- stopping_rounds** 5
- stopping_metric** AUTO
- stopping_tolerance** 0
- max_runtime_secs** 0
- autoencoder**
- categorical_encoding** AUTO
- export_checkpoints_dir**

GRID?

- pretrained_autoencoder**
- overwrite_with_best_model**
- target_ratio_comm_to_comp** 0.05
- seed** -1
- rho** 0.99
- epsilon** 1e-8
- nesterov_accelerated_gradient**
- max_w2** 3.4028235e+38
- initial_weight_distribution** UniformAdaptive
- regression_stop** 0.000001
- diagnostics**
- fast_mode**
- force_load_balance**
- single_node_mode**
- shuffle_training_data**
- missing_values_handling** MeanImputation
- quiet_mode**

GRID?

- missing_values_handling** MeanImputation
- quiet_mode**
- sparse**
- col_major**
- average_activation** 0
- sparsity_beta** 0
- max_categorical_features** 2147483647
- reproducible**
- export_weights_and_biases**
- mini_batch_size** 1
- elastic_averaging**

GRID?

Build Model

Activar

Nous cliquons maintenant sur le bouton «Afficher». Comme nous pouvons le voir sur la figure 1-9, nous obtenons une sortie graphique, d'autres options nous permettent de voir les paramètres avec lesquels le modèle a été construit ou la progression de l'entraînement.

The screenshot shows the Flow interface with the following details:

- Job** (34.6s):
 - Run Time: 00:00:33.285
 - Remaining Time: 00:00:00.0
 - Type: Model
 - Key: `deeplearning-6b8c8372-b02a-484c-a383-f35ab462b32b`
 - Description: DeepLearning
 - Status: DONE
 - Progress: 100%
 - Done.
 - Actions: View
- Model** (404ms):
 - Model ID: deeplearning-6b8c8372-b02a-484c-a383-f35ab462b32b
 - Algorithm: Deep Learning
 - Actions: Refresh, Predict..., Download POJO, Download Model Deployment Package (MOJO), Export, Inspect, Delete, Download Gen Model
 - MODEL PARAMETERS**
 - SCORING HISTORY - DEVIANCE** (Line chart showing training deviation vs epochs from 0 to 10, decreasing from ~0.189 to ~0.179)
 - VARIABLE IMPORTANCES** (Bar chart showing scaled importance for variables: ap_hi, ap_lo, cholesterol, age, weight, gender, active, height, smokes, gluc, id, alco)
 - OUTPUT**
 - COLUMN_TYPES**
 - OUTPUT - STATUS OF NEURON LAYERS (PREDICTING CARDIO, REGRESSION, GAUSSIAN DISTRIBUTION, QUADRATIC LOSS, 43 Å 001 WEIGHTS/BIASES, 514.1 KB, 561 Å 760 TRAINING SAMPLES, MINI-BATCH SIZE 1)**

layer	units	type	dropout	l1	mean_rate	rate_rms	momentum	mean_weight_rms	mean_bias	bias_rms
1	12	Input	0							
2	200	Rectifier	0	0	0.0226	0.0331	0	-0.0049	0.4526	0.0269
3	200	Rectifier	0	0	0.1195	0.1112	0	-0.0463	0.0975	0.8111
4	1	Linear	0	0	0.0014	0.0007	0	0.0041	0.0416	-0.1436
 - OUTPUT - SCORING HISTORY**
 - OUTPUT - TRAINING_METRICS**
 - OUTPUT - VARIABLE_IMPORTANCES**
 - PREVIEW POJO**

Nous pouvons effectuer le cycle complet de prévision du modèle de charge dans Flow. Dans la vue du modèle, cliquez sur «Prédire».

The screenshot shows the H2O Flow interface with three main steps:

- Predict**: A step where a model named "prediction-f8b8afc-4e18" is used to predict on a frame named "cardio.hex_0.80". The "Compute" dropdown is set to "Deep".
- Prediction**: A step showing the detailed prediction results for the "cardio.hex_0.80" frame. It includes metrics like MSE: 0.178925, RMSE: 0.422996, and R-squared: 0.284299.
- bindFrames**: A step that combines the prediction frame with the original "cardio.hex_0.80" frame.

5. CONCLUSION

Dans ce parti nous avons présenté brièvement la préparation de donné sous H2O python et le démarrage de H2O sous jupyter notebook et nous montre aussi comment faire la connecter à un cluster H2O. Ensuite nous avons parlé les différentes modèles de machine learning H2O. H2O prend en charge les modèles suivants:

- ✓ Apprentissage en profondeur
- ✓ Machine de renforcement de gradient (GBM)
- ✓ Naïve Bayes ...etc.

Les commandes Flow que nous voyons peuvent être enregistrées en tant que scripts et chargées ultérieurement. Mais, il y a certaines choses que nous pouvons faire avec les API Python que nous ne pouvons pas faire dans Flow, principalement, la fusion des ensembles de données et la manipulation de données.

Flow peut faire tout ce dont nous avons besoin, mais la plupart d'entre nous voudront utiliser Python. Enfin nous appliquées aux modèles construits dans Flow.

III. ETUDE DES METHODES SUPERVISEES DE H2O

1. INTRODUCTION

Dans ce parti, nous examinons les propriétés des méthodes supervisées disponibles sous H2O. Nous essayons de mettre en évidence les fonctionnalités que l'on doit retrouver dans ce type de librairie, et celles qui seraient un peu plus particulières, moins présentes dans les autres outils.

2. DONNEE

Nous traitons la base « CARDIO » cette fois-ci (prédition de maladie cardio-vasculaire). On souhaite prédire la présence ou l'absence d'une maladie cardio vasculaire (1, 0) chez des patients à partir de leurs caractéristiques (âge, indice de masse corporelle, etc.). Elle est de taille réduite (70000 observations, 13 variables, variable cible binaire), plus facile à manipuler dans ce nouveau contexte. Dans 70000 patients sur lesquels on a mesuré les variables suivantes :

- Age
- Height : taille
- Weight : poids
- Gender : 0 : féminin, 1 : masculin
- Ap_hi pressure : tension systolique
- Ap_lo : tension diastolique
- Cholesterol : 1 : normal, 2 : au-dessus de la normale, 3 : très au-dessus de la normale
- Gluc : taux de glucose 1 : normal, 2 : au-dessus de la normale, 3 : très au-dessus de la normale
- Smoking : 0 : non-fumeur, 1 : fumeur
- Alco : consommation d'alcool 0 : non, 1 : oui
- active : pratique d'une activité physique 0 : non, 1 : oui
- cardio : problème cardio vasculaire, 0 : non, 1 : oui

Démarrage et configuration du serveur. Nous créons un nouveau projet Python, nous importons le package puis nous démarrons le serveur H2O en demandant toute la puissance disponible sur notre machine.

```

Entrée [1]: #importation du package
import h2o

Entrée [2]: #démarrage H2O
h2o.init()

Checking whether there is an H2O instance running at http://localhost:54321 . connected.

H2O_cluster_uptime: 7 hours 50 mins
H2O_cluster_timezone: Europe/Paris
H2O_data_parsing_timezone: UTC
H2O_cluster_version: 3.32.0.2
H2O_cluster_version_age: 2 months and 11 days
H2O_cluster_name: H2O_from_python_Administrateur_mrp89k
H2O_cluster_total_nodes: 1
H2O_cluster_free_memory: 74.6 Mb
H2O_cluster_total_cores: 4
H2O_cluster_allowed_cores: 4
H2O_cluster_status: locked, healthy
H2O_connection_url: http://localhost:54321
H2O_connection_proxy: ("http": null, "https": null)
H2O_internal_security: False
H2O_API_Extensions: Amazon S3, Algos, AutoML, Core V3, TargetEncoder, Core V4
Python_version: 3.7.4 final

```

Importation des données. Nous importons le fichier de données au format CSV (séparateur “; ”) ET nous affichons les 10 premières lignes.

Prétraitement des données

```

Entrée [3]: #changer Le répertoire courant
import os
os.chdir("C:/Users/Administrateur/Documents/H2O ipynb")

Entrée [4]: #chargement des données
cardio = h2o.import_file("cardio.csv")

Parse progress: |██████████| 100%

Entrée [5]: #affichage des premières valeurs
print(cardio.head())

```

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	18393	2	168	62	110	80		1	1	0	0	1	0
1	20228	1	156	85	140	90		3	1	0	0	1	1
2	18857	1	165	64	130	70		3	1	0	0	0	1
3	17623	2	169	82	150	100		1	1	0	0	1	1
4	17474	1	156	56	100	60		1	1	0	0	0	0
8	21914	1	151	67	120	80		2	2	0	0	0	0
9	22113	1	157	93	130	80		3	1	0	0	1	0
12	22584	2	178	95	130	90		3	3	0	0	1	1
13	17668	1	158	71	110	70		1	1	0	0	1	0
14	19834	1	164	68	110	60		1	1	0	0	0	0

Précision importante, en utilisant la commande import_file () de H2O, notre ensemble de données « cardio » est directement typé au format Data Frame spécifique à H2O (h2o.frame.H2OFrame). Il est par conséquent directement reconnu par les différents outils de la librairie.

```

Entrée [6]: #affichage du type
print(type(cardio))

<class 'h2o.frame.H2OFrame'>

```

Nous disposons de 70000 observations et 13 variables.

```

Entrée [7]: #dimension
print(cardio.shape)

(70000, 13)

```

Nous affichons la liste des variables.

```

Entrée [8]: #affichage de La liste des colonnes
print(cardio.col_names)

['id', 'age', 'gender', 'height', 'weight', 'ap_hi', 'ap_lo', 'cholesterol', 'gluc', 'smoke', 'alco', 'active', 'cardio']

```

Nous affichons la résumé de donnée.

Entrée [9]: `# Résumé de donnée
cardio.describe()`

Rows:70000
Cols:13

	id	age	gender	height	weight	ap_hi	ap_lo
type	int	int	int	int	real	int	int
mins	0.0	10798.0	1.0	55.0	10.0	-150.0	-70.0
mean	49972.41989999998	19468.86581428571	1.3495714285714273	164.35922857142862	74.20568999999999	128.81728571428567	96.63041428571428
maxs	99999.0	23713.0	2.0	250.0	200.0	16020.0	11000.0
sigma	28851.30232317291	2467.2516672414017	0.4768380155828637	8.210126364538034	14.395756678511377	154.01141945609132	188.4725302963903
zeros	1	0	0	0	0	0	21
missing	0	0	0	0	0	0	0
0	0.0	18393.0	2.0	168.0	62.0	110.0	80.0
1	1.0	20228.0	1.0	156.0	85.0	140.0	90.0
2	2.0	18857.0	1.0	165.0	64.0	130.0	70.0
3	3.0	17623.0	2.0	169.0	82.0	150.0	100.0
4	4.0	17474.0	1.0	156.0	56.0	100.0	60.0
5	8.0	21914.0	1.0	151.0	67.0	120.0	80.0
6	9.0	22113.0	1.0	157.0	93.0	130.0	80.0
7	12.0	22584.0	2.0	178.0	95.0	130.0	90.0
8	13.0	17668.0	1.0	158.0	71.0	110.0	70.0
9	14.0	19834.0	1.0	164.0	68.0	110.0	60.0

Nous vérifions les types des données à analyser par H2O.

Entrée [10]: `# vérifier les types de données analysés par H2O
cardio.types`

Out[10]: `{'id': 'int',
'age': 'int',
'gender': 'int',
'height': 'int',
'weight': 'real',
'ap_hi': 'int',
'ap_lo': 'int',
'cholesterol': 'int',
'gluc': 'int',
'smoke': 'int',
'alco': 'int',
'active': 'int',
'cardio': 'int'}`

Nous nous assurons que la variable cible « cardio » est bien reconnue comme type factor, avec les modalités, dans l'ordre alphabétique, {0, 1}

Entrée [11]: `cardio["cardio"] = cardio["cardio"].asfactor()`

Entrée [12]: `#cardio est bien un type facteur
cardio['cardio'].isfactor()`

Out[12]: `[True]`

Entrée [13]: `#nombre de niveaux (modalités) de "diabète"
cardio['cardio'].levels()`

Out[13]: `[['0', '1']]`

Subdivision en échantillons d'apprentissage et de test. L'étape suivante consiste à scinder les données en ensembles d'apprentissage et de test. La fonction `split_frame()` fait l'affaire. Nous indiquons la proportion des données (ratios) dévolue à l'apprentissage. L'option `seed` assure la reproductibilité à l'identique de l'opération. Nous vérifions ensuite les dimensions des sous échantillons.

Subdivision en échantillons d'apprentissage et de test

Entrée [14]: `#subdivision
cardioTrain,cardioTest = cardio.split_frame(ratios=[0.8],seed=1)`

Entrée [15]: `#vérification train
cardioTrain.shape`

Out[15]: `(56007, 13)`

Entrée [16]: `#vérification test
cardioTest.shape`

Out[16]: `(13993, 13)`

3. RANDOM FOREST

La méthode Random Forest est basée sur l'agrégation d'arbres de décision élaborés de façon à maximiser leur décorrélation. Ainsi, nous maximisons leur efficacité lorsque nous les faisons coopérer. Initialisation et apprentissage. Sous H2O, nous utilisons la classe H2ORandomForestEstimator. Nous l'instancions avec (ntrees = 200) arbres, de profondeur maximum (max_depth = 30) et avec des feuilles comportant au moins (min_rows = 1) observation. Pour rappel, plus les arbres sont grands, plus Random Forest est performant.

```
In [17]: #random forest
from h2o.estimators import H2ORandomForestEstimator
```

```
In [18]: x = cardioTrain.col_names[:-1]
x
```

```
Out[18]: ['id',
'age',
'gender',
'height',
'weight',
'ap_hi',
'ap_lo',
'cholesterol',
'gluc',
'smoke',
'alco',
'active']
```

```
In [19]: y=cardio.col_names[-1]
y
```

```
Out[19]: 'cardio'
```

```
In [20]: #instanciation
rf = H2ORandomForestEstimator(seed=1, nfolds=5, model_id="rf",
                               ntrees=200,
                               max_depth=30,
                               stopping_rounds=2,
                               stopping_tolerance=0.01,
                               score_each_iteration=True)
```

```
In [21]: #apprentissage
rf.train(x=x, y=y, training_frame=cardioTrain)
```

```
drf Model Build progress: |███████████| 100%
```

Nous pouvons suivre l'évolution de la fonction de perte avec plot () .

```
In [22]: #évolution de l'apprentissage
rf.plot()
```

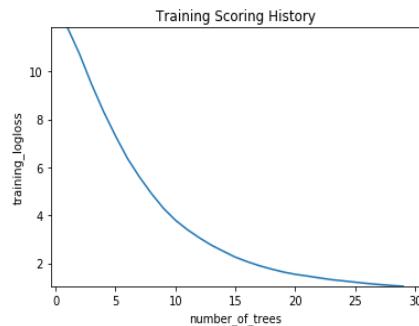


Figure 3.1 : Random Forest - Evolution de la perte en fonction du nombre d'arbres

Une vingt-neuf d'arbre aurait suffi apparemment. Mais Random Forest possède la propriété très avantageuse d'être robuste au sur apprentissage. L'empilement des arbres ne lui porte pas préjudice. Ce n'est pas le cas des autres méthodes ensemblistes.

Un résumé du modèle est disponible.

```
In [23]: #résumé  
rf.summary()  
  
Model Summary:  
  
number_of_trees number_of_internal_trees model_size_in_bytes min_depth max_depth mean_depth min_leaves max_leaves mean_leaves  
0 29.0 29.0 4029837.0 30.0 30.0 30.0 10438.0 11578.0 11052.207
```

Out[23]:

29 arbres ont bien été générés, leur profondeur moyenne est de 30.0, avec un nombre de feuilles moyen de 11052.207 (nombre de règles par arbre). Globalement, la méthode Random Forest est parfaite.

Nous affichons les résultats de l'apprentissage.

```
In [24]: #affichage  
rf.show()  
  
Model Details  
=====  
H2ORandomForestEstimator : Distributed Random Forest  
Model Key: rf  
  
Model Summary:  
  
number_of_trees number_of_internal_trees model_size_in_bytes min_depth max_depth mean_depth min_leaves max_leaves mean_leaves  
0 29.0 29.0 4029837.0 30.0 30.0 30.0 10438.0 11578.0 11052.207  
  
ModelMetricsBinomial: drf  
** Reported on train data. **  
  
MSE: 0.2036909877312715  
RMSE: 0.4513213796523177  
LogLoss: 1.0472604630956661  
Mean Per-Class Error: 0.29515391831013504  
AUC: 0.7605602495399105  
AUCPR: 0.7447028114740423  
Gini: 0.521120499679821  
  
Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.36365493542527855:  
  
 0   1   Error      Rate  
---  
0  0 15236.0 12739.0 0.4554 (12739.0/27975.0)  
1  1 5221.0 22811.0 0.1863 (5221.0/28032.0)  
2 Total 20457.0 35550.0 0.3207 (17960.0/56007.0)
```

Maximum Metrics: Maximum metrics at their respective thresholds

	metric	threshold	value	idx
0	max f1	0.363655	0.717530	239.0
1	max f2	0.004385	0.834968	398.0
2	max f0point5	0.571446	0.713256	153.0
3	max accuracy	0.541815	0.704805	165.0
4	max precision	0.958566	0.844556	14.0
5	max recall	0.000000	1.000000	399.0
6	max specificity	0.999997	0.984808	0.0
7	max absolute_mcc	0.541815	0.411036	165.0
8	max min_per_class_accuracy	0.500010	0.693155	181.0
9	max mean_per_class_accuracy	0.541815	0.704846	165.0
10	max tns	0.999997	27550.000000	0.0
11	max fns	0.999997	25774.000000	0.0
12	max fps	0.000000	27975.000000	399.0
13	max tps	0.000000	28032.000000	399.0
14	max tnr	0.999997	0.984808	0.0
15	max fnr	0.999997	0.919449	0.0
16	max fpr	0.000000	1.000000	399.0
17	max tpr	0.000000	1.000000	399.0

Gains/Lift Table: Avg response rate: 50,05 %, avg score: 50,12 %

group	cumulative_data_fraction	lower_threshold	lift	cumulative_lift	response_rate	score	cumulative_response_rate	cumulative_score	capture_r
0	1	0.047833	1.000000	1.681752	1.681752	0.841732	1.000000	0.841732	1.000000
1	2	0.050012	0.988974	1.768692	1.685539	0.885246	0.993372	0.843627	0.999711
2	3	0.105433	0.900000	1.665187	1.674841	0.833441	0.922284	0.838273	0.959011
3	4	0.153124	0.857143	1.614231	1.655964	0.807937	0.874778	0.828825	0.932777
4	5	0.215598	0.800000	1.544013	1.623524	0.772792	0.821789	0.812588	0.900616
5	6	0.300498	0.714286	1.441645	1.572137	0.721556	0.751290	0.786869	0.858426
6	7	0.408270	0.600000	1.252536	1.487771	0.626905	0.652044	0.744643	0.803947
7	8	0.510115	0.500000	1.039265	1.398227	0.520161	0.537453	0.699825	0.750741
8	9	0.599996	0.394724	0.822365	1.311961	0.411601	0.437420	0.656648	0.703805
9	10	0.701823	0.300000	0.695767	1.222558	0.348238	0.342843	0.611901	0.651433
10	11	0.810292	0.200000	0.572586	1.135550	0.286584	0.244911	0.568353	0.597015
11	12	0.900012	0.103030	0.494223	1.071618	0.247363	0.149782	0.536354	0.552431
12	13	1.000000	0.000000	0.355353	1.000000	0.177857	0.039794	0.500509	0.501174

4 15

```
ModelMetricsBinomial: drf
** Reported on cross-validation data. **

MSE: 0.19346623584480488
RMSE: 0.4398479699129362
LogLoss: 0.6190823543606799
Mean Per-Class Error: 0.2818953119070353
AUC: 0.7762727124573066
AUCPR: 0.7597919488310206
Gini: 0.5525454249146131
```

Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.38223638286126355:

	0	1	Error	Rate
0	0	16244.0	11731.0	0.4193 (11731.0/27975.0)
1	1	5292.0	22740.0	0.1888 (5292.0/28032.0)
2 Total	21536.0	34471.0	0.3039 (17023.0/56007.0)	

Maximum Metrics: Maximum metrics at their respective thresholds

	metric	threshold	value	idx
0	max f1	0.382236	0.727645	242.0
1	max f2	0.100615	0.837320	366.0
2	max f0point5	0.592447	0.730107	154.0
3	max accuracy	0.546509	0.718053	170.0
4	max precision	0.967484	0.865052	6.0
5	max recall	0.000046	1.000000	399.0
6	max specificity	0.999977	0.998820	0.0
7	max absolute_mcc	0.546509	0.438448	170.0
8	max min_per_class_accuracy	0.489900	0.714710	195.0
9	max mean_per_class_accuracy	0.546509	0.718105	170.0
10	max tns	0.999977	27942.000000	0.0
11	max fns	0.999977	27837.000000	0.0
12	max fps	0.000046	27975.000000	399.0
13	max tps	0.000046	28032.000000	399.0
14	max tnr	0.999977	0.998820	0.0
15	max fnr	0.999977	0.993044	0.0
16	max fpr	0.000046	1.000000	399.0
17	max tpr	0.000046	1.000000	399.0

Gains/Lift Table: Avg response rate: 50,05 %, avg score: 50,20 %

group		cumulative_data_fraction	lower_threshold	lift	cumulative_lift	response_rate	score	cumulative_response_rate	cumulative_score	capture_r
0	1	0.010159	0.967742	1.727592	1.727592	0.864675	0.985670	0.864675	0.985670	0.017
1	2	0.020176	0.960000	1.713052	1.720373	0.857398	0.962248	0.861062	0.974042	0.017
2	3	0.031996	0.935484	1.699177	1.712543	0.850453	0.943788	0.857143	0.962865	0.020
3	4	0.042727	0.925926	1.655553	1.698230	0.828619	0.928079	0.849979	0.954129	0.017
4	5	0.050208	0.920000	1.688020	1.696708	0.844869	0.920869	0.849218	0.949173	0.012
5	6	0.100095	0.875000	1.677605	1.687188	0.839656	0.893709	0.844452	0.921530	0.083
6	7	0.150071	0.838235	1.661046	1.678482	0.831368	0.853374	0.840095	0.898833	0.083
7	8	0.203885	0.794118	1.599566	1.657653	0.800597	0.813201	0.829670	0.876231	0.086
8	9	0.301623	0.705882	1.507782	1.609089	0.754658	0.748948	0.805363	0.834986	0.147
9	10	0.401164	0.600000	1.292675	1.530577	0.646996	0.652840	0.766067	0.789790	0.128
10	11	0.500009	0.490196	1.022442	1.430126	0.511741	0.543769	0.715791	0.741155	0.101
11	12	0.599996	0.393701	0.842357	1.332176	0.421607	0.441973	0.666766	0.691297	0.084
12	13	0.700377	0.306452	0.664567	1.236492	0.332622	0.350108	0.618875	0.642397	0.066
13	14	0.800043	0.223214	0.551928	1.151212	0.276245	0.264550	0.576192	0.595326	0.055
14	15	0.899995	0.137703	0.481825	1.076871	0.241158	0.180910	0.538983	0.549302	0.048
15	16	1.000000	0.000000	0.308203	1.000000	0.154258	0.076733	0.500509	0.502042	0.030

Cross-Validation Metrics Summary:

		mean	sd	cv_1_valid	cv_2_valid	cv_3_valid	cv_4_valid	cv_5_valid
0	accuracy	0.6965689	0.00500723	0.68885326	0.696102	0.7024347	0.6964141	0.6990405
1	auc	0.776263	0.004009069	0.7756584	0.7730693	0.77467716	0.77467036	0.78323966
2	aucpr	0.7597324	0.0045673577	0.7624629	0.7576347	0.7533	0.76004446	0.76522
3	err	0.3034311	0.00500723	0.31114677	0.30389795	0.29756528	0.3035859	0.30095956
4	err_count	3398.8	55.782616	3492.0	3407.0	3361.0	3378.0	3356.0
5	f0point5	0.6857879	0.0047237575	0.67857784	0.6858292	0.6918133	0.6867532	0.6859662
6	f1	0.72801626	0.004813609	0.72268105	0.72539693	0.72805244	0.72836924	0.73558146
7	f2	0.7758623	0.009927481	0.77291566	0.7698098	0.7682981	0.7753544	0.7929336
8	lift_top_group	1.7168936	0.034274522	1.7159022	1.7191662	1.6674864	1.7175714	1.7643415
9	logloss	0.6190586	0.011921326	0.61782265	0.6138325	0.6326343	0.6283477	0.602656
10	max_per_class_error	0.41858542	0.018358262	0.43287572	0.41042113	0.39257294	0.41845766	0.43859965
11	mcc	0.4039378	0.009937571	0.3893179	0.4013658	0.41261637	0.4025771	0.41381177
12	mean_per_class_accuracy	0.6964416	0.005059308	0.688799	0.69607353	0.702561	0.69586957	0.6989048
13	mean_per_class_error	0.3035584	0.005059308	0.31120095	0.30392647	0.29743895	0.30413043	0.3010952
14	mse	0.1934632	0.0018592428	0.19344145	0.19502705	0.19415678	0.19439487	0.19029588
15	pr_auc	0.7597324	0.0045673577	0.7624629	0.7576347	0.7533	0.76004446	0.76522
16	precision	0.66028196	0.0065765353	0.6520493	0.6617647	0.6695937	0.6615542	0.65644777
17	r2	0.2261432	0.007439009	0.22623402	0.21989174	0.22337154	0.22240287	0.23881578
18	recall	0.8114686	0.014946328	0.8104738	0.8025682	0.79769504	0.81019676	0.8364093
19	rmse	0.43984044	0.0021188757	0.4398198	0.44161868	0.44063225	0.44090235	0.43622914

See the whole table with `table.as_data_frame()`

Scoring History:

	timestamp	duration	number_of_trees	training_rmse	training_logloss	training_auc	training_pr_auc	training_lift	training_classification_error
0	2021-01-28 20:20:30	37.490 sec	0.0	NaN	NaN	NaN	NaN	NaN	NaN
1	2021-01-28 20:20:30	37.736 sec	1.0	0.595357	11.820245	0.639459	0.607521	1.280344	0.502930
2	2021-01-28 20:20:31	37.908 sec	2.0	0.582589	10.722862	0.646468	0.615802	1.299885	0.500783
3	2021-01-28 20:20:31	38.074 sec	3.0	0.567442	9.468607	0.655637	0.624506	1.325145	0.501488
4	2021-01-28 20:20:31	38.232 sec	4.0	0.553045	8.329476	0.665199	0.634159	1.349055	0.373494
5	2021-01-28 20:20:31	38.389 sec	5.0	0.540499	7.311625	0.673797	0.644044	1.377476	0.377739
6	2021-01-28 20:20:31	38.648 sec	6.0	0.528180	6.373432	0.683367	0.653458	1.403100	0.368252
7	2021-01-28 20:20:32	38.883 sec	7.0	0.518854	5.599549	0.689959	0.661036	1.427178	0.379031
8	2021-01-28 20:20:32	39.141 sec	8.0	0.509595	4.909078	0.697837	0.669048	1.449207	0.365118
9	2021-01-28 20:20:32	39.319 sec	9.0	0.500646	4.295484	0.706597	0.678693	1.476650	0.359882
10	2021-01-28 20:20:32	39.481 sec	10.0	0.493900	3.798060	0.712705	0.685384	1.496073	0.356660
11	2021-01-28 20:20:32	39.653 sec	11.0	0.488737	3.402248	0.71434	0.691288	1.514332	0.355083
12	2021-01-28 20:20:33	39.852 sec	12.0	0.484087	3.063719	0.722092	0.696603	1.530289	0.352601
13	2021-01-28 20:20:33	40.030 sec	13.0	0.479819	2.757080	0.726400	0.702362	1.548330	0.351775
14	2021-01-28 20:20:33	40.296 sec	14.0	0.476096	2.503598	0.730547	0.707242	1.563059	0.347842
15	2021-01-28 20:20:33	40.567 sec	15.0	0.472560	2.257751	0.734510	0.712770	1.583384	0.347303
16	2021-01-28 20:20:34	40.823 sec	16.0	0.469250	2.069434	0.738626	0.717696	1.599167	0.345253
17	2021-01-28 20:20:34	41.088 sec	17.0	0.466733	1.902069	0.741331	0.720765	1.608862	0.343442
18	2021-01-28 20:20:34	41.370 sec	18.0	0.464577	1.763658	0.743916	0.723345	1.615277	0.341257
19	2021-01-28 20:20:34	41.646 sec	19.0	0.462719	1.638882	0.745926	0.726268	1.628309	0.326548

```
See the whole table with table.as_data_frame()
```

```
Variable Importances:
```

	variable	relative_importance	scaled_importance	percentage
0	age	51419.511719	1.000000	0.200374
1	ap_hi	47065.410156	0.915322	0.183406
2	id	40776.289062	0.793012	0.158899
3	weight	35104.593750	0.682710	0.136797
4	height	31632.037109	0.615176	0.123265
5	ap_lo	25020.437500	0.486594	0.097501
6	cholesterol	10056.023438	0.195568	0.039187
7	gender	4304.793945	0.083719	0.016775
8	gluc	3964.414307	0.077099	0.015449
9	active	3127.366943	0.060821	0.012187
10	smoke	2423.509521	0.047132	0.009444
11	alco	1723.715454	0.033523	0.006717

Cette dernière partie attire notre attention. Random Forest sait évaluer la contribution des variables dans le modèle. AGE et AP_HI se démarquent à nouveau. Une partie des variables seulement sont affichées lorsqu'elles sont nombreuses (les meilleures et les pires).

Nous utilisons varimp () pour obtenir la liste exhaustive.

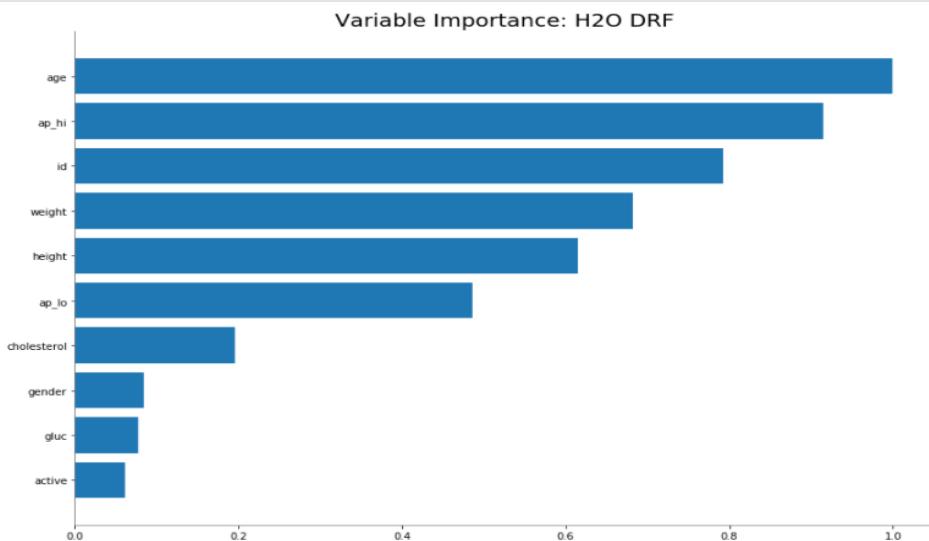
```
In [25]: #ou tabulaire  
import pandas as pd  
pd.DataFrame(rf.varimp())
```

Out[25]:

	0	1	2	3
0	age	51419.511719	1.000000	0.200374
1	ap_hi	47065.410156	0.915322	0.183406
2	id	40776.289062	0.793012	0.158899
3	weight	35104.593750	0.682710	0.136797
4	height	31632.037109	0.615176	0.123265
5	ap_lo	25020.437500	0.486594	0.097501
6	cholesterol	10056.023438	0.195568	0.039187
7	gender	4304.793945	0.083719	0.016775
8	gluc	3964.414307	0.077099	0.015449
9	active	3127.366943	0.060821	0.012187
10	smoke	2423.509521	0.047132	0.009444
11	alco	1723.715454	0.033523	0.006717

Un graphique peut faire l'affaire également.

```
In [26]: #importance - graphique  
rf.varimp_plot()
```



Evaluation en test. De nouveau, `model_performance()` propose une vue globale des performances sur l'échantillon passé en paramètre.

```
In [27]: #evaluation
rf.model_performance(cardioTest)

ModelMetricsBinomial: drf
** Reported on test data. **

MSE: 0.1932603156185623
RMSE: 0.4396138255543862
LogLoss: 0.61689784771367
Mean Per-Class Error: 0.27953074494813557
AUC: 0.7759074822259334
AUCPR: 0.7604561529082023
Gini: 0.5518149644518668

Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.39063150461377766:

      0   1   Error    Rate
0  4245.0 2801.0  0.3975 (2801.0/7046.0)
1  1433.0 5514.0  0.2063 (1433.0/6947.0)
2 Total 5678.0 8315.0  0.3026 (4234.0/13993.0)

Maximum Metrics: Maximum metrics at their respective thresholds

      metric threshold  value  idx
0       max f1  0.390632  0.722579 233.0
1       max f2  0.076872  0.834184 380.0
2     max f0point5  0.597450  0.731661 145.0
3     max accuracy  0.540081  0.720860 166.0
4     max precision  0.982796  0.888889  4.0
5     max recall  0.000040  1.000000 399.0
6     max specificity  0.999839  0.998865  0.0
7   max absolute_mcc  0.568946  0.444900 156.0
8 max min_per_class_accuracy  0.482751  0.714732 188.0
9 max mean_per_class_accuracy  0.540081  0.720469 166.0
10      max tns  0.999839 7038.000000  0.0
11      max fns  0.999839 6892.000000  0.0
12      max fps  0.000040 7046.000000 399.0
13      max tps  0.000040 6947.000000 399.0
14      max tnr  0.999839  0.998865  0.0
15      max fnr  0.999839  0.992083  0.0
16      max fpr  0.000040  1.000000 399.0
17      max tpr  0.000040  1.000000 399.0

Gains/Lift Table: Avg response rate: 49,65 %, avg score: 49,83 %

      group cumulative_data_fraction lower_threshold      lift cumulative_lift response_rate    score cumulative_response_rate cumulative_score capture_r
0       1           0.018438        0.965517  1.741000  1.741000  0.864341  0.976373  0.864341  0.976373  0.032
1       2           0.020010        0.958966  1.922694  1.755276  0.954545  0.962219  0.871429  0.975261  0.003
2       3           0.044594        0.931034  1.739048  1.746330  0.863372  0.935853  0.866987  0.953536  0.042
3       4           0.050382        0.919540  1.666109  1.737113  0.827160  0.925000  0.862411  0.950258  0.009
4       5           0.126635        0.862069  1.680115  1.702792  0.834114  0.882168  0.845372  0.909258  0.128
5       6           0.169871        0.827586  1.634706  1.685463  0.811570  0.835729  0.836769  0.890543  0.070
6       7           0.208962        0.793103  1.598144  1.669128  0.793419  0.802029  0.828659  0.873984  0.062
7       8           0.300150        0.698276  1.548574  1.632502  0.768809  0.744823  0.810476  0.834744  0.141
8       9           0.408919        0.586207  1.274457  1.537265  0.632720  0.642201  0.763195  0.783529  0.138
9      10           0.500036        0.479119  0.965261  1.433034  0.479216  0.525988  0.711448  0.736600  0.087
10     11           0.600086        0.381226  0.797068  1.327002  0.395714  0.429710  0.658807  0.685433  0.079
11     12           0.699993        0.308045  0.662772  1.232199  0.329041  0.343810  0.611741  0.636675  0.066
12     13           0.799971        0.222057  0.583110  1.151078  0.289492  0.263751  0.571467  0.590068  0.058
13     14           0.909240        0.137931  0.458443  1.067839  0.227600  0.178508  0.530142  0.540608  0.050
14     15           1.000000        0.000000  0.320377  1.000000  0.159055  0.074079  0.496463  0.498266  0.029
```

Out[27]:

La modulation des seuils d'affectation ne donne pas une image objective des performances du modèle. Nous préférons réaliser manuellement la séquence prédition et confrontation sur l'échantillon test.

```
In [28]: #prediction - de nouveau voir le seuil d'affectation
predRf = rf.predict(cardioTest).as_data_frame()
print(predRf.head(10))

drf prediction progress: |██████████| 100%
   predict      p0      p1
0        1  0.206897  0.793103
1        0  0.655172  0.344828
2        0  0.724138  0.275862
3        1  0.467884  0.532116
4        1  0.176012  0.823988
5        1  0.052874  0.947126
6        1  0.258621  0.741379
7        0  0.775862  0.224138
8        0  0.671182  0.328818
9        0  0.741379  0.258621

In [29]: #scikit-Learn
from sklearn import metrics
#f1-score
print(metrics.f1_score(cardioTest.as_data_frame()[["cardio"]],predRf.predict, pos_label=1))

0.7211862788773626
```

4. GRADIENT BOOSTING

Gradient boosting est une variante du boosting où l'on exploite les écarts entre les probabilités d'affectation estimées et les classes d'appartenance pour corriger les modèles successifs. Par rapport à Random Forest, elle est plus efficace car nécessite peu d'arbres, mais elle est autrement plus sujette au sur apprentissage. Le paramétrage joue un rôle essentiel pour cette méthode.

Initialisation et apprentissage. Justement, dans cette section, nous allons sciemment pénaliser la méthode en créant trop d'arbres (ntrees = 50) dimensionnés raisonnablement (max_depth = 5).

Modèle 2 : Gradient boosting

```
In [30]: #gradient boosting
from h2o.estimators import H2OGradientBoostingEstimator

In [31]: #instanciation
gb = H2OGradientBoostingEstimator(distribution="bernoulli", ntrees=50, max_depth=3, min_rows=2, learn_rate=0.2, nfolds=5)

In [32]: #apprentissage
gb.train(x=x, y=y, training_frame=cardioTrain)

gbm Model Build progress: |██████████| 100%

In [33]: #évolution
gb.plot()
```

Manifestement, nous avons trop d'arbres. Pour un Random Forest, l'information était anecdotique. Pour Gradient Boosting, c'est un problème potentiel.

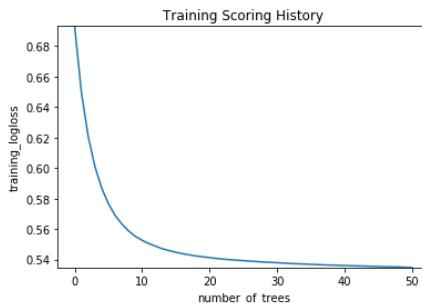


Figure 3.2 : Gradient Boosting - Evolution de la perte en fonction du nombre d'arbres

Un résumé du modèle est disponible.

```
In [34]: #résumé  
gb.summary()  
Model Summary:
```

	number_of_trees	number_of_internal_trees	model_size_in_bytes	min_depth	max_depth	mean_depth	min_leaves	max_leaves	mean_leaves
0	50.0	50.0	7891.0	3.0	3.0	3.0	5.0	8.0	7.94

```
Out[34]:
```

L'affichage des résultats (show) n'apporte rien de particulier par rapport aux autres approches. On notera quand même la valeur invraisemblable du seuil d'affectation proposé (0.37615134585429205) pour maximiser le F1-Score.

```
In [35]: #affichage  
gb.show()
```

```
Model Details  
=====  
H2GradientBoostingEstimator : Gradient Boosting Machine  
Model Key: GBM_model_python_1611861553922_342
```

```
Model Summary:
```

	number_of_trees	number_of_internal_trees	model_size_in_bytes	min_depth	max_depth	mean_depth	min_leaves	max_leaves	mean_leaves
0	50.0	50.0	7891.0	3.0	3.0	3.0	5.0	8.0	7.94

```
ModelMetricsBinomial: gbm  
** Reported on train data. **
```

```
MSE: 0.1779928606140518  
RMSE: 0.42189200112594194  
LogLoss: 0.5347327499388773  
Mean Per-Class Error: 0.2600348816213106  
AUC: 0.8078490616877022  
AUCPR: 0.7959357192924584  
Gini: 0.6156981233754044
```

```
Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.37615134585429205:
```

	0	1	Error	Rate
0	0	17480.0	10495.0	0.3752 (10495.0/27975.0)
1	1	5110.0	22922.0	0.1823 (5110.0/28032.0)
2	Total	22590.0	33417.0	0.2786 (15605.0/56007.0)

```
Maximum Metrics: Maximum metrics at their respective thresholds
```

	metric	threshold	value	idx
0	max f1	0.376151	0.746050	243.0
1	max f2	0.201341	0.844849	336.0
2	max f0point5	0.566070	0.758085	154.0
3	max accuracy	0.508041	0.739908	181.0
4	max precision	0.944773	1.000000	0.0
5	max recall	0.050051	1.000000	398.0
6	max specificity	0.944773	1.000000	0.0
7	max absolute_mcc	0.508041	0.483008	181.0
8	max min_per_class_accuracy	0.450503	0.736550	208.0
9	max mean_per_class_accuracy	0.508041	0.739965	181.0
10	max tns	0.944773	27975.000000	0.0
11	max fns	0.944773	28031.000000	0.0
12	max fps	0.044145	27975.000000	399.0
13	max tps	0.050051	28032.000000	398.0
14	max tnr	0.944773	1.000000	0.0
15	max fnr	0.944773	0.999964	0.0
16	max fpr	0.044145	1.000000	399.0
17	max tpr	0.050051	1.000000	398.0

Gains/Lift Table: Avg response rate: 50,05 %, avg score: 50,05 %

group	cumulative_data_fraction	lower_threshold	lift	cumulative_lift	response_rate	score	cumulative_response_rate	cumulative_score	capture_r	
0	1	0.010017	0.888560	1.827018	0.914439	0.896885	0.914439	0.896885	0.018	
1	2	0.020015	0.881535	1.776763	1.801913	0.889286	0.884859	0.901873	0.890877	0.017
2	3	0.030014	0.876789	1.776763	1.793535	0.889286	0.878977	0.897680	0.886913	0.017
3	4	0.040013	0.873017	1.748221	1.782211	0.875000	0.874867	0.892012	0.883903	0.017
4	5	0.050012	0.869935	1.816009	1.788968	0.908929	0.871379	0.895395	0.881399	0.018
5	6	0.100005	0.856414	1.753929	1.771452	0.877857	0.863104	0.886627	0.872253	0.087
6	7	0.149999	0.843515	1.696844	1.746586	0.849286	0.849893	0.874182	0.864801	0.084
7	8	0.200029	0.829058	1.664972	1.726173	0.833333	0.836734	0.863965	0.857781	0.083
8	9	0.299998	0.745857	1.581887	1.678092	0.791749	0.798425	0.839900	0.838001	0.158
9	10	0.400004	0.553611	1.321990	1.589063	0.661668	0.633630	0.795340	0.786906	0.132
10	11	0.500009	0.450198	1.012717	1.473789	0.506874	0.500444	0.737645	0.729612	0.101
11	12	0.600014	0.372446	0.829722	1.366441	0.415283	0.408698	0.683916	0.676125	0.082
12	13	0.700002	0.298699	0.664324	1.266152	0.332500	0.336462	0.633720	0.627608	0.066
13	14	0.800096	0.244927	0.532815	1.174409	0.266679	0.270992	0.587802	0.582994	0.053
14	15	0.899995	0.173842	0.398165	1.088247	0.199285	0.213845	0.544677	0.542019	0.039
15	16	1.000000	0.039607	0.205825	1.000000	0.103017	0.126817	0.500509	0.500497	0.020

```
ModelMetricsBinomial: gbm
** Reported on cross-validation data. **

MSE: 0.18011811705367514
RMSE: 0.4244032481657924
LogLoss: 0.540282743234226
Mean Per-Class Error: 0.26406883133179093
AUC: 0.8820242619439649
AUCPR: 0.7836853261952996
Gini: 0.6040485238879298
```

Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.3620963591044405:

	0	1	Error	Rate
0	16551.0	11424.0	0.4084	(11424.0/27975.0)
1	1	4715.0	23317.0	0.1682
2 Total	21266.0	34741.0	0.2882	(16139.0/56007.0)

Maximum Metrics: Maximum metrics at their respective thresholds

	metric	threshold	value	idx
0		max f1	0.362096	0.742899 250.0
1		max f2	0.182671	0.843558 343.0
2		max f0point5	0.586581	0.753918 149.0
3		max accuracy	0.494711	0.735890 188.0
4		max precision	0.931745	1.000000 0.0
5		max recall	0.039888	1.000000 399.0
6		max specificity	0.931745	1.000000 0.0
7		max absolute_mcc	0.539356	0.475488 168.0
8		max min_per_class_accuracy	0.452064	0.732726 207.0
9	max mean_per_class_accuracy	0.494711	0.735931 188.0	
10		max tns	0.931745	27975.000000 0.0
11		max fns	0.931745	28030.000000 0.0
12		max fps	0.039888	27975.000000 399.0
13		max tps	0.039888	28032.000000 399.0
14		max tnr	0.931745	1.000000 0.0
15		max fnr	0.931745	0.999929 0.0
16		max fpr	0.039888	1.000000 399.0
17		max tpr	0.039888	1.000000 399.0

Gains/Lift Table: Avg response rate: 50,05 %, avg score: 50,05 %

group	cumulative_data_fraction	lower_threshold	lift	cumulative_lift	response_rate	score	cumulative_response_rate	cumulative_score	capture_r
0	1	0.010017	0.890688	1.691683	1.691683	0.846702	0.897455	0.846702	0.897455
1	2	0.020015	0.882903	1.748221	1.719927	0.875000	0.886502	0.860839	0.891983
2	3	0.030014	0.877721	1.705407	1.715090	0.853571	0.880279	0.858418	0.888084
3	4	0.040031	0.873576	1.737982	1.720818	0.869875	0.875571	0.861285	0.884953
4	5	0.050012	0.870352	1.722755	1.721204	0.862254	0.871957	0.861478	0.882360
5	6	0.100005	0.856223	1.733236	1.727219	0.867500	0.863059	0.864488	0.872711
6	7	0.149999	0.843068	1.698272	1.717571	0.850000	0.849896	0.859660	0.865107
7	8	0.200011	0.827085	1.673413	1.706530	0.837558	0.835460	0.854133	0.857694
8	9	0.299998	0.746540	1.600514	1.671195	0.801071	0.798230	0.836448	0.837875
9	10	0.400039	0.552414	1.310107	1.580895	0.655720	0.629558	0.791252	0.785779
10	11	0.500045	0.452042	1.011290	1.466978	0.506160	0.502667	0.734236	0.729159
11	12	0.599996	0.371698	0.823028	1.359704	0.411933	0.409342	0.680544	0.675882
12	13	0.701073	0.301386	0.667048	1.259841	0.333863	0.336657	0.630562	0.626974
13	14	0.799989	0.246447	0.536277	1.170375	0.268412	0.272017	0.585783	0.583085
14	15	0.899995	0.174150	0.418071	1.086780	0.209248	0.214098	0.543943	0.542084
15	16	1.000000	0.031794	0.219024	1.000000	0.109623	0.125786	0.500509	0.500452

Cross-Validation Metrics Summary:

	mean	sd	cv_1_valid	cv_2_valid	cv_3_valid	cv_4_valid	cv_5_valid
0	accuracy	0.713533	0.010911124	0.7193683	0.7068981	0.718502	0.724953
1	auc	0.80207485	0.0033918307	0.80702245	0.8035309	0.801722	0.79975295
2	aucpr	0.78383553	0.0050883815	0.7908352	0.78446555	0.7766403	0.7846924
3	err	0.28646702	0.010911124	0.28063172	0.2931019	0.281498	0.275047
4	err_count	3208.8	122.41814	3163.0	3259.0	3157.0	3072.0
5	f0point5	0.70015275	0.013392236	0.7066986	0.6890486	0.70088416	0.7185495
6	f1	0.7439748	0.004676877	0.7512778	0.7393426	0.7424329	0.74561113
7	f2	0.7940172	0.012515685	0.80185986	0.7975566	0.78921807	0.7747909
8	lift_top_group	1.6891943	0.027254036	1.7354293	1.6814317	1.6891711	1.6740953
9	logloss	0.54028404	0.003744378	0.5353291	0.5376631	0.54110277	0.54290056
10	max_per_class_error	0.40465933	0.043739296	0.40315357	0.4246624	0.3836964	0.34754992
11	mcc	0.44000912	0.016034953	0.45031226	0.43205336	0.44923967	0.4530191
12	mean_per_class_accuracy	0.713448	0.011078682	0.7181947	0.7085393	0.7200656	0.7239985
13	mean_per_class_error	0.28655204	0.011078682	0.28180525	0.2914607	0.2799344	0.27600148
14	mse	0.18011838	0.001637343	0.1780708	0.17895761	0.18026386	0.18116416
15	pr_auc	0.78383553	0.0050883815	0.7908352	0.78446555	0.7766403	0.7846924
16	precision	0.67382556	0.019115344	0.6798065	0.6591557	0.6756757	0.70157397
17	r2	0.2794201	0.006553257	0.2876502	0.28406087	0.2787808	0.27521437
18	recall	0.83155525	0.023330817	0.83954304	0.841741	0.8238276	0.7955469
19	rmse	0.42440006	0.0019292196	0.42198434	0.42303383	0.4245749	0.42563382

See the whole table with `table.as_data_frame()`

Scoring History:

	timestamp	duration	number_of_trees	training_rmse	training_logloss	training_auc	training_pr_auc	training_lift	training_classification_error
0	2021-01-28 20:21:04	13.856 sec	0.0	0.500000	0.693147	0.500000	0.500509	1.000000	0.499491
1	2021-01-28 20:21:04	13.950 sec	1.0	0.477974	0.649862	0.768415	0.759181	1.683106	0.308283
2	2021-01-28 20:21:04	13.966 sec	2.0	0.463113	0.621085	0.776567	0.765168	1.686497	0.306158
3	2021-01-28 20:21:04	13.997 sec	3.0	0.452938	0.601170	0.782346	0.769594	1.698518	0.306230
4	2021-01-28 20:21:04	14.012 sec	4.0	0.445803	0.586991	0.785560	0.778271	1.769033	0.306230
5	2021-01-28 20:21:04	14.091 sec	5.0	0.440786	0.576763	0.792143	0.782168	1.772632	0.293731
6	2021-01-28 20:21:04	14.132 sec	6.0	0.437096	0.568982	0.795319	0.783539	1.772632	0.294445
7	2021-01-28 20:21:04	14.145 sec	7.0	0.434378	0.563328	0.796666	0.783960	1.772632	0.296356
8	2021-01-28 20:21:05	14.176 sec	8.0	0.432240	0.558718	0.798093	0.784416	0.000000	0.292928
9	2021-01-28 20:21:05	14.254 sec	9.0	0.430739	0.555236	0.798334	0.784872	1.774966	0.292713
10	2021-01-28 20:21:05	14.285 sec	10.0	0.429581	0.552646	0.798922	0.785272	1.744935	0.290303
11	2021-01-28 20:21:05	14.317 sec	11.0	0.428602	0.550500	0.800032	0.785600	1.744935	0.291517
12	2021-01-28 20:21:05	14.395 sec	12.0	0.427844	0.548793	0.800666	0.785709	1.744935	0.285464
13	2021-01-28 20:21:05	14.442 sec	13.0	0.427126	0.547067	0.801034	0.786437	1.743492	0.293053
14	2021-01-28 20:21:05	14.489 sec	14.0	0.426621	0.545919	0.801397	0.786514	1.737221	0.295160
15	2021-01-28 20:21:05	14.567 sec	15.0	0.426183	0.544801	0.801805	0.787229	1.745312	0.291696
16	2021-01-28 20:21:05	14.629 sec	16.0	0.425780	0.543840	0.801999	0.787687	1.766217	0.291089
17	2021-01-28 20:21:05	14.676 sec	17.0	0.425442	0.543090	0.802360	0.787536	1.769326	0.290607
18	2021-01-28 20:21:05	14.765 sec	18.0	0.425143	0.542358	0.802655	0.787311	1.762912	0.289232
19	2021-01-28 20:21:05	14.810 sec	19.0	0.424908	0.541810	0.802942	0.787638	1.760446	0.278572

See the whole table with `table.as_data_frame()`

Variable Importances:

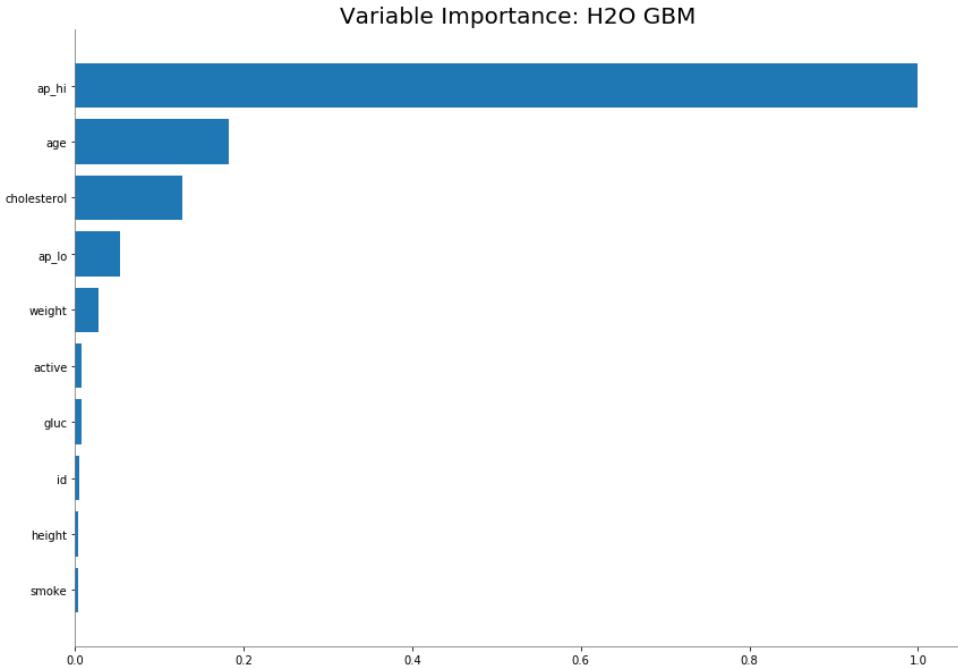
	variable	relative_importance	scaled_importance	percentage
0	ap_hi	8017.527344	1.000000	0.703689
1	age	1463.536255	0.182542	0.128453
2	cholesterol	1024.267700	0.127754	0.089899
3	ap_lo	426.289978	0.053170	0.037415
4	weight	219.486008	0.027376	0.019264
5	active	65.719299	0.008197	0.005768
6	gluc	56.285793	0.007020	0.004940
7	id	35.211563	0.004392	0.003090
8	height	31.477167	0.003926	0.002763
9	smoke	24.670198	0.003077	0.002165
10	gender	16.081520	0.002006	0.001411
11	alco	13.020377	0.001624	0.001143

Pour cette méthode également, nous pouvons obtenir l'importance relative des variables. AP_HI et AGE se démarquent encore une fois.

In [36]: `#ou tabulaire
pd.DataFrame(gb.varimp())`

	0	1	2	3
0	ap_hi	8017.527344	1.000000	0.703689
1	age	1463.536255	0.182542	0.128453
2	cholesterol	1024.267700	0.127754	0.089899
3	ap_lo	426.289978	0.053170	0.037415
4	weight	219.486008	0.027376	0.019264
5	active	65.719299	0.008197	0.005768
6	gluc	56.285793	0.007020	0.004940
7	id	35.211563	0.004392	0.003090
8	height	31.477167	0.003926	0.002763
9	smoke	24.670198	0.003077	0.002165
10	gender	16.081520	0.002006	0.001411
11	alco	13.020377	0.001624	0.001143

In [37]: `#plotting var importance
gb.varimp_plot()`



Performances en test. Nous réalisons la séquence prédiction – confrontation sur l'échantillon test pour évaluer la pertinence du modèle.

```
In [38]: #evaluation
gb.model_performance(cardioTest)

ModelMetricsBinomial: gbm
** Reported on test data. **

MSE: 0.18091714833680023
RMSE: 0.4253435650586479
LogLoss: 0.5426995239762797
Mean Per-Class Error: 0.26198891195210194
AUC: 0.7998337417961328
AUCPR: 0.779368098152081
Gini: 0.5996674835922655

Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.39869118889470845:
```

	0	1	Error	Rate
0	0	4747.0	2299.0	0.3263 (2299.0/7046.0)
1	1	1515.0	5432.0	0.2181 (1515.0/6947.0)
2 Total	6262.0	7731.0	0.2726	(3814.0/13993.0)

Maximum Metrics: Maximum metrics at their respective thresholds

	metric	threshold	value	idx
0	max f1	0.398691	0.740155	234.0
1	max f2	0.197359	0.840697	340.0
2	max f0point5	0.562324	0.755092	156.0
3	max accuracy	0.514580	0.738512	177.0
4	max precision	0.925676	1.000000	0.0
5	max recall	0.056597	1.000000	397.0
6	max specificity	0.925676	1.000000	0.0
7	max absolute_mcc	0.533089	0.481493	169.0
8	max min_per_class_accuracy	0.445343	0.730531	212.0
9	max mean_per_class_accuracy	0.514580	0.738011	177.0
10	max tns	0.925676	7046.000000	0.0
11	max fns	0.925676	6944.000000	0.0
12	max fps	0.049274	7046.000000	399.0
13	max tps	0.056597	6947.000000	397.0
14	max tnr	0.925676	1.000000	0.0
15	max fnr	0.925676	0.999568	0.0
16	max fpr	0.049274	1.000000	399.0
17	max tpr	0.056597	1.000000	397.0

```
Gains/Lift Table: Avg response rate: 49,65 %, avg score: 49,65 %
```

group	cumulative_data_fraction	lower_threshold	lift	cumulative_lift	response_rate	score	cumulative_response_rate	cumulative_score	capture_r
0	1	0.010005	0.887596	1.755276	1.755276	0.871429	0.896160	0.871429	0.896160
1	2	0.020010	0.881031	1.740888	1.740888	0.864286	0.883930	0.867857	0.890045
2	3	0.030015	0.876803	1.812826	1.769663	0.900000	0.878810	0.878571	0.886300
3	4	0.040020	0.872666	1.712113	1.755276	0.850000	0.874599	0.871429	0.883375
4	5	0.050025	0.869153	1.654563	1.735133	0.821429	0.870820	0.861429	0.880864
5	6	0.100050	0.854647	1.714991	1.725062	0.851429	0.862029	0.856429	0.871447
6	7	0.150147	0.841727	1.712544	1.720885	0.850214	0.848410	0.854355	0.863760
7	8	0.200029	0.825815	1.636218	1.699771	0.812321	0.834076	0.843873	0.856358
8	9	0.300007	0.728444	1.626950	1.675503	0.807720	0.790859	0.831825	0.834530
9	10	0.399986	0.541594	1.343314	1.592471	0.666905	0.616010	0.790602	0.779910
10	11	0.500036	0.442796	0.956769	1.465276	0.475000	0.492739	0.727455	0.722451
11	12	0.600014	0.369439	0.830752	1.359547	0.412437	0.402215	0.674964	0.669091
12	13	0.701708	0.298642	0.662452	1.258522	0.328883	0.332552	0.624809	0.620319
13	14	0.800329	0.245276	0.544432	1.170528	0.270290	0.270780	0.581123	0.577247
14	15	0.899950	0.177129	0.397359	1.084941	0.197274	0.215068	0.538633	0.537155
15	16	1.000000	0.046538	0.235955	1.000000	0.117143	0.130507	0.496463	0.496470

```
Out[38]:
```

Encore une fois ici, la modulation des seuils d'affectation ne donne pas une image objective des performances du modèle. Nous préférions réaliser manuellement la séquence prédiction et confrontation sur l'échantillon test.

```
In [39]: #prediction - de nouveau voir le seuil d'affectation
predGb = gb.predict(cardioTest).as_data_frame()
print(predGb.head(10))
```

```
gbm prediction progress: |██████████| 100%
 predict      p0      p1
0       1  0.252751  0.747249
1       0  0.765738  0.234262
2       0  0.657139  0.342861
3       1  0.458098  0.541902
4       1  0.172428  0.827572
5       1  0.141920  0.858880
6       1  0.474475  0.525525
7       1  0.480001  0.519999
8       0  0.631144  0.368856
9       1  0.574867  0.425133
```

```
In [40]: #F1-score
print(metrics.f1_score(cardioTest.as_data_frame()[["cardio"]], predGb.predict, pos_label=1))
```

```
0.7387030806781449
```

5. NAÏVE BAYES

Naïve Bayes est un algorithme s'appuyant sur l'indépendance des descripteurs conditionnellement aux valeurs de la cible. Sous H2O, les calculs internes se limitent aux estimations des moyennes et écarts-type conditionnels pour l'ensemble des descripteurs. Tu m'étonnes que la méthode soit si rapide, y compris sur des très grandes bases de données. La complexité est linéaire tant en nombre de variables qu'en nombre d'observations, tous les paramètres peuvent être estimés en une seule passe sur l'ensemble d'apprentissage.

Initialisation et apprentissage. Pas de paramètres particuliers à spécifier pour linstanciation.

Modèle 3 : Naive Bayes

```
In [41]: #naive bayes
from h2o.estimators import H2ONaiveBayesEstimator

In [42]: #instanciation
nb = H2ONaiveBayesEstimator(seed=100)

In [43]: nb.train(x=x, y=y, training_frame=cardioTrain)
naivebayes Model Build progress: |██████████| 100%

In [44]: #résumé
nb.summary()

Model Summary:



| number_of_response_levels | min_apriori_probability | max_apriori_probability |
|---------------------------|-------------------------|-------------------------|
| 0                         | 2.0                     | 0.499491                |
|                           |                         | 0.500509                |



Out[44]:

In [45]: #affichage
nb.show()

Model Details
=====
H2ONaiveBayesEstimator : Naive Bayes
Model Key: NaiveBayes_model_python_1611861553922_631

Model Summary:



| number_of_response_levels | min_apriori_probability | max_apriori_probability |
|---------------------------|-------------------------|-------------------------|
| 0                         | 2.0                     | 0.499491                |
|                           |                         | 0.500509                |



ModelMetricsBinomial: naivebayes
** Reported on train data. **

MSE: 0.28169345048885763
RMSE: 0.5307480103484681
LogLoss: 0.8690611112379693
Mean Per-Class Error: 0.35698028182268904
AUC: 0.6913777373286651
AUCPR: 0.6713060091463037
Gini: 0.3827554746573303

Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.11540451312199568:



|         | 0       | 1       | Error  | Rate              |
|---------|---------|---------|--------|-------------------|
| 0       | 9081.0  | 18894.0 | 0.6754 | (18894.0/27975.0) |
| 1       | 3392.0  | 24640.0 | 0.121  | (3392.0/28032.0)  |
| 2 Total | 12473.0 | 43534.0 | 0.3979 | (22286.0/56007.0) |



Maximum Metrics: Maximum metrics at their respective thresholds



|    | metric                      | threshold     | value        | idx   |
|----|-----------------------------|---------------|--------------|-------|
| 0  | max f1                      | 1.154045e-01  | 0.688595     | 338.0 |
| 1  | max f2                      | 4.504013e-02  | 0.834766     | 386.0 |
| 2  | max f0point5                | 1.977122e-01  | 0.642725     | 283.0 |
| 3  | max accuracy                | 1.977122e-01  | 0.643045     | 283.0 |
| 4  | max precision               | 9.999240e-01  | 0.839120     | 0.0   |
| 5  | max recall                  | 2.368810e-157 | 1.000000     | 399.0 |
| 6  | max specificity             | 9.999240e-01  | 0.995031     | 0.0   |
| 7  | max absolute_mcc            | 1.977122e-01  | 0.286386     | 283.0 |
| 8  | max min_per_class_accuracy  | 2.072635e-01  | 0.640054     | 276.0 |
| 9  | max mean_per_class_accuracy | 1.977122e-01  | 0.643020     | 283.0 |
| 10 | max tns                     | 9.999240e-01  | 27836.000000 | 0.0   |
| 11 | max fns                     | 9.999240e-01  | 27307.000000 | 0.0   |
| 12 | max fps                     | 2.368810e-157 | 27975.000000 | 399.0 |
| 13 | max tps                     | 2.368810e-157 | 28032.000000 | 399.0 |
| 14 | max tnr                     | 9.999240e-01  | 0.995031     | 0.0   |
| 15 | max fnr                     | 9.999240e-01  | 0.974137     | 0.0   |
| 16 | max fpr                     | 2.368810e-157 | 1.000000     | 399.0 |
| 17 | max tpr                     | 2.368810e-157 | 1.000000     | 399.0 |


```

Gains/Lift Table: Avg response rate: 50,05 %, avg score: 32,94 %

group	cumulative_data_fraction	lower_threshold	lift	cumulative_lift	response_rate	score	cumulative_response_rate	cumulative_score	capture_r
0	1	0.010017	0.999982	1.698806	1.698806	0.850267	0.999996	0.850267	0.999996
1	2	0.020015	0.998017	1.580534	1.639723	0.791071	0.999211	0.820696	0.999603
2	3	0.030014	0.996548	1.555560	1.611685	0.778571	0.997264	0.806663	0.998824
3	4	0.040013	0.995218	1.473500	1.577154	0.737500	0.995873	0.789380	0.998087
4	5	0.050012	0.993469	1.316517	1.525046	0.658929	0.994434	0.763299	0.997356
5	6	0.100005	0.941324	1.502043	1.513546	0.751786	0.971091	0.757543	0.984226
6	7	0.149999	0.777058	1.447812	1.491637	0.724643	0.880227	0.746578	0.949564
7	8	0.200011	0.540389	1.303921	1.444700	0.652624	0.648716	0.723085	0.874339
8	9	0.299998	0.337618	1.265498	1.384973	0.633393	0.422320	0.693191	0.723684
9	10	0.400004	0.250205	1.184654	1.334891	0.592930	0.288578	0.668125	0.614902
10	11	0.500009	0.205825	1.073359	1.282583	0.537225	0.225911	0.641944	0.537101
11	12	0.599996	0.172021	0.965089	1.229673	0.483036	0.188539	0.615462	0.479015
12	13	0.700002	0.140146	0.852551	1.175796	0.426710	0.155940	0.588496	0.432859
13	14	0.799989	0.107062	0.708208	1.117354	0.354464	0.123702	0.559246	0.394218
14	15	0.899995	0.070649	0.632814	1.063513	0.316729	0.089494	0.532298	0.360358
15	16	1.000000	0.000000	0.428416	1.000000	0.214426	0.050929	0.500509	0.329414

L'objet est peu disert quand même. Nous n'avons pas d'information sur l'importance des variables, encore moins sur les coefficients estimés. Or, contrairement à ce que l'on peut croire, avec l'estimation paramétrique des distributions conditionnelles (hypothèse gaussienne), il est possible de produire un modèle explicite.

Prédiction et évaluation en test. Voyons ce que donne le modèle sur l'échantillon test.

```
In [46]: #prediction - de nouveau voir le seuil d'affectation
predNb = nb.predict(cardioTest).as_data_frame()
print(predNb.head(10))

naivebayes prediction progress: |██████████| 100%
   predict    p0      p1
0      1 0.084216 0.915784
1      1 0.798881 0.201119
2      1 0.791668 0.208332
3      0 0.894575 0.105425
4      0 0.890269 0.109731
5      1 0.832762 0.167238
6      1 0.705882 0.294118
7      1 0.697926 0.302074
8      1 0.823376 0.176624
9      1 0.789719 0.210281
```

```
In [47]: #F1-score
print(metrics.f1_score(cardioTest.as_data_frame()[ "cardio"],predNb.predict, pos_label=1))

0.6871371147833855
```

Malgré sa rusticité, Naïve Bayes tient la route par rapport aux autres approches étudiées jusqu'à présent.

6. PERCEPTRON SIMPLE

Nous étudions le perceptron dans cette section, avec la classe de calcul H2ODeepLearningEstimator avec le paramètre hidden = [] défini un perceptron sans couche cachée. Dans les variables sont standardisées (standardize=True) ; nous passons 1250 fois sur la base (epochs = 1250) ; la variable cible étant binaire, nous choisissons (distribution = bernoulli) ; (export_weights_and_biases = True) permet de produire les coefficients estimés.

Initialisation et apprentissage. Après instanciation, nous lançons l'apprentissage. Pas de remarques particulières à ce stade.

Modèle 4 : Perceptron simple

```
In [48]: #deep Learning
from h2o.estimators import H2ODeepLearningEstimator

In [49]: #instanciation
ps = H2ODeepLearningEstimator(seed=100,epochs=1250,standardize=True,hidden=[],distribution="bernoulli")

In [50]: #apprentissage
ps.train(x=x, y=y, training_frame=cardioTrain)

deeplearning Model Build progress: |██████████| 100%
```

Structure du réseau. La méthode summary () permet d'afficher l'architecture du réseau.

```
In [51]: #structure du réseau
ps.summary()

Status of Neuron Layers: predicting cardio, 2-class classification, bernoulli distribution, CrossEntropy loss, 26 weights/biases, 4,6 KB, 39À 295À 113 training samples, mini-batch size 1



| layer | units | type | dropout | I1 | I2 | mean_rate  | rate_rms   | momentum | mean_weight | weight_rms | mean_bias | bias_rms  |
|-------|-------|------|---------|----|----|------------|------------|----------|-------------|------------|-----------|-----------|
| 0     | 1     | 12   | Input   | 0  |    |            |            |          |             |            |           |           |
| 1     | 2     | 2    | Softmax | 0  | 0  | 0.00195111 | 0.00179441 | 0        | -0.150255   | 1.86727    | -0.111204 | 0.0887527 |



Out[51]:
```

Nous observons 12 neurones dans la couche d'entrée (layer = 1) parce que nous avons 12 variables explicatives ; 2 neurones dans la couche de sortie (layer = 2), parce que la variable cible est binaire. Au total, 26 coefficients ont été estimés en prenant en compte du biais : $(12 + 1) \times 2 = 26$.

Importance des variables. La sortie standard des modèles show () propose des sorties additionnelles qui ont attiré notre attention.

```
In [52]: #affichage
ps.show()

Model Details
=====
H2ODeepLearningEstimator : Deep Learning
Model Key: DeepLearning_model_python_1611861553922_633

Status of Neuron Layers: predicting cardio, 2-class classification, bernoulli distribution, CrossEntropy loss, 26 weights/biases, 4,6 KB, 39À 295À 113 training samples, mini-batch size 1



| layer | units | type | dropout | I1 | I2 | mean_rate  | rate_rms   | momentum | mean_weight | weight_rms | mean_bias | bias_rms  |
|-------|-------|------|---------|----|----|------------|------------|----------|-------------|------------|-----------|-----------|
| 0     | 1     | 12   | Input   | 0  |    |            |            |          |             |            |           |           |
| 1     | 2     | 2    | Softmax | 0  | 0  | 0.00195111 | 0.00179441 | 0        | -0.150255   | 1.86727    | -0.111204 | 0.0887527 |



ModelMetricsBinomial: deeplearning
** Reported on train data. **

MSE: 0.1877152429946841
RMSE: 0.4332611718059721
LogLoss: 0.5730983249942956
Mean Per-Class Error: 0.2702326578605818
AUC: 0.7900637098155333
AUCPR: 0.7751398261430212
Gini: 0.5801274196310666

Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.3916398143706861:



|         | 0      | 1      | Error                  | Rate                   |
|---------|--------|--------|------------------------|------------------------|
| 0       | 0      | 3078.0 | 1900.0                 | 0.3817 (1900.0/4978.0) |
| 1       | 1      | 954.0  | 4010.0                 | 0.1922 (954.0/4964.0)  |
| 2 Total | 4032.0 | 5910.0 | 0.2871 (2854.0/9942.0) |                        |


```

Maximum Metrics: Maximum metrics at their respective thresholds

	metric	threshold	value	idx
0	max f1	0.391640	0.737539	249.0
1	max f2	0.149998	0.837336	357.0
2	max f0point5	0.532697	0.743192	184.0
3	max accuracy	0.487945	0.729833	203.0
4	max precision	0.977246	0.898552	8.0
5	max recall	0.001663	1.000000	399.0
6	max specificity	0.999342	0.999397	0.0
7	max absolute_mcc	0.512028	0.462207	193.0
8	max min_per_class_accuracy	0.453200	0.726027	219.0
9	max mean_per_class_accuracy	0.487945	0.729767	203.0
10	max tns	0.999342	4975.000000	0.0
11	max fns	0.999342	4948.000000	0.0
12	max fps	0.001663	4978.000000	399.0
13	max tps	0.001663	4964.000000	399.0
14	max tnr	0.999342	0.999397	0.0
15	max fnr	0.999342	0.996777	0.0
16	max fpr	0.001663	1.000000	399.0
17	max tpr	0.001663	1.000000	399.0

group	cumulative_data_fraction	lower_threshold	lift	cumulative_lift	response_rate	score	cumulative_response_rate	cumulative_score	capture_r	
0	1	0.010058	0.987615	1.762482	1.762482	0.880000	0.993598	0.880000	0.993598	0.017
1	2	0.020016	0.970178	1.780285	1.771339	0.888889	0.978990	0.884422	0.986331	0.017
2	3	0.030074	0.956114	1.662341	1.734884	0.830000	0.963481	0.866221	0.978689	0.016
3	4	0.040032	0.941471	1.780285	1.746178	0.888889	0.948741	0.871859	0.971239	0.017
4	5	0.050091	0.928197	1.722425	1.741408	0.860000	0.934107	0.869478	0.963783	0.017
5	6	0.100080	0.862976	1.716703	1.729068	0.857143	0.895728	0.863317	0.929790	0.085
6	7	0.150070	0.811097	1.644166	1.700786	0.820926	0.837799	0.849196	0.899147	0.082
7	8	0.200060	0.757373	1.652226	1.688652	0.824950	0.784118	0.843137	0.870404	0.082
8	9	0.300040	0.648628	1.551481	1.642944	0.774648	0.701320	0.820315	0.814061	0.155
9	10	0.400020	0.538989	1.317751	1.561666	0.657948	0.593448	0.779733	0.758922	0.131
10	11	0.500000	0.451050	1.019544	1.453263	0.509054	0.491644	0.725609	0.705477	0.101
11	12	0.599980	0.386716	0.834173	1.350099	0.416499	0.418080	0.674099	0.657586	0.083
12	13	0.699960	0.327624	0.660890	1.251655	0.329980	0.358082	0.624946	0.614805	0.066
13	14	0.799940	0.265382	0.519847	1.160190	0.259557	0.297390	0.579278	0.575133	0.051
14	15	0.899920	0.189513	0.433206	1.079423	0.216298	0.230770	0.538952	0.536875	0.043
15	16	1.000000	0.000002	0.285830	1.000000	0.142714	0.124526	0.499296	0.495607	0.028

Scoring History:

	timestamp	duration	training_speed	epochs	iterations	samples	training_rmse	training_logloss	training_r2	training_auc	training_pr_auc	trainin
0	2021-01-28 20:21:56	0.000 sec	None	0.000000	0	0.0	NaN	NaN	NaN	NaN	NaN	NaN
1	2021-01-28 20:21:56	0.455 sec	529063 obs/sec	1.785366	1	99993.0	0.434226	0.579296	0.245788	0.788752	0.771537	1.74
2	2021-01-28 20:22:01	5.471 sec	753827 obs/sec	69.626118	39	3899550.0	0.433826	0.575025	0.247179	0.789223	0.773371	1.78
3	2021-01-28 20:22:06	10.467 sec	768257 obs/sec	139.270484	78	7800122.0	0.433400	0.574383	0.248657	0.789842	0.775138	1.80
4	2021-01-28 20:22:11	15.503 sec	772554 obs/sec	208.880836	117	11698789.0	0.433278	0.573832	0.249078	0.789949	0.773920	1.74
5	2021-01-28 20:22:16	20.549 sec	773781 obs/sec	278.499009	156	15597894.0	0.433478	0.573416	0.248386	0.789630	0.774267	1.74
6	2021-01-28 20:22:21	25.658 sec	776583 obs/sec	349.918189	196	19597868.0	0.433261	0.573098	0.249138	0.790064	0.775140	1.76
7	2021-01-28 20:22:26	30.700 sec	777261 obs/sec	419.544093	235	23497406.0	0.432951	0.575570	0.250211	0.790895	0.774913	1.76
8	2021-01-28 20:22:31	35.812 sec	780054 obs/sec	490.941329	275	27496151.0	0.433456	0.574392	0.248462	0.790153	0.774511	1.74
9	2021-01-28 20:22:36	40.936 sec	779458 obs/sec	560.569161	314	31395797.0	0.433914	0.576297	0.246872	0.789754	0.775015	1.80

10	2021-01-28 20:22:42	45.948 sec	779836 obs/sec	630.182049	353	35294606.0	0.432963	0.574980	0.250170	0.791001	0.774664	1.78
11	2021-01-28 20:22:47	51.121 sec	780625 obs/sec	701.610745	393	39295113.0	0.433058	0.573994	0.249840	0.790672	0.775436	1.80
12	2021-01-28 20:22:47	51.152 sec	780625 obs/sec	701.610745	393	39295113.0	0.433261	0.573098	0.249138	0.790064	0.775140	1.76

Variable Importances:

variable	relative_importance	scaled_importance	percentage
0 ap_hi	1.000000	1.000000	0.248718
1 alco	0.545083	0.545083	0.135572
2 smoke	0.523521	0.523521	0.130209
3 weight	0.409243	0.409243	0.101786
4 ap_lo	0.360910	0.360910	0.089765
5 cholesterol	0.280050	0.280050	0.069653
6 height	0.209698	0.209698	0.052156
7 age	0.167012	0.167012	0.041539
8 active	0.166398	0.166398	0.041386
9 gender	0.161450	0.161450	0.040155
10 id	0.119461	0.119461	0.029712
11 gluc	0.077798	0.077798	0.019350

L'outil présente une mesure de l'impact des variables et, patatras, nous n'avons pas le même classement qu'avec l'analyse des coefficients. Par exemple, "alco" devient bien placé, a contrario, "age" est reléguée en 7ème position...

Nous trouvons la description suivante sur le site de H2O : « Variable importances for Neural Network models are notoriously difficult to compute, and there are many pitfalls. H2O Deep Learning has implemented the method of Gedeon, and returns relative variable importances in descending order of importance. ». L'article cité (T. Gédéon, "Data Mining of Inputs : analysing magnitude and functional measures", Int. J. of Neural Syst., 8(2):209-208, 1997), décrit le calcul d'une mesure (functional measure) de l'influence des entrées via les caractéristiques du réseau. L'idée est de pouvoir écarter les descripteurs peu pertinents, c'est la moindre des choses, mais surtout de mettre en lumière les descripteurs qui agissent, même faiblement sur la prédiction, mais en étant peu corrélée aux autres, au détriment de ceux qui sont redondants. Pourquoi pas. Il n'en reste pas moins que les résultats proposés ne sont pas complètement (le rôle primordial de AP_HI a bien été détecté) cohérent avec l'analyse des coefficients qui fait référence dans un modèle linéaire. L'énorme avantage de cette approche.

Performances en test. Voyons ce que donne le perceptron simple sur l'échantillon test.

```
In [53]: #prediction - de nouveau voir le seuil d'affectation
predps = ps.predict(cardioTest).as_data_frame()
print(predps.head(10))

deeplearning prediction progress: |██████████| 100%
   predict      p0      p1
0      1  0.255547  0.744453
1      0  0.729474  0.270526
2      1  0.558120  0.441880
3      1  0.546701  0.453299
4      1  0.491929  0.598071
5      1  0.227847  0.772153
6      1  0.407444  0.592556
7      1  0.481153  0.518847
8      1  0.577451  0.422549
9      0  0.683000  0.317000
```

```
In [54]: performance = ps.model_performance(cardioTest)
performance.show()

ModelMetricsBinomial: deeplearning
** Reported on test data. **

MSE: 0.18899419564027498
RMSE: 0.4347346266865281
LogLoss: 0.5869692943396676
Mean Per-Class Error: 0.27062963565712095
AUC: 0.7875997051762216
AUCPR: 0.7665084120455569
Gini: 0.5751994103524432

Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.4007850108718426:
```

	0	1	Error	Rate
0	0	4485.0	2561.0	0.3635 (2561.0/7046.0)
1	1	1423.0	5524.0	0.2048 (1423.0/6947.0)
2	Total	5908.0	8085.0	0.2847 (3984.0/13993.0)

Maximum Metrics: Maximum metrics at their respective thresholds

	metric	threshold	value	idx
0	max f1	0.400785	0.734965	242.0
1	max f2	0.182941	0.835183	347.0
2	max f0point5	0.546029	0.739567	177.0
3	max accuracy	0.474248	0.729579	208.0
4	max precision	0.991087	0.864407	3.0
5	max recall	0.001352	1.000000	399.0
6	max specificity	0.999411	0.998723	0.0
7	max absolute_mcc	0.474248	0.459621	208.0
8	max min_per_class_accuracy	0.451860	0.727505	218.0
9	max mean_per_class_accuracy	0.474248	0.729370	208.0
10	max tns	0.999411	7037.000000	0.0
11	max fns	0.999411	6925.000000	0.0
12	max fps	0.001352	7046.000000	399.0
13	max tps	0.001352	6947.000000	399.0
14	max tnr	0.999411	0.998723	0.0
15	max fnr	0.999411	0.996833	0.0
16	max fpr	0.001352	1.000000	399.0
17	max tpr	0.001352	1.000000	399.0

Gains/Lift Table: Avg response rate: 49,65 %, avg score: 49,51 %

group	cumulative_data_fraction	lower_threshold	lift	cumulative_lift	response_rate	score	cumulative_response_rate	cumulative_score	capture_r
0	1	0.010005	9.878694e-01	1.697726	1.697726	0.842857	0.994258	0.842857	0.994258
1	2	0.020010	9.730481e-01	1.625788	1.661757	0.807143	0.980230	0.825000	0.987244
2	3	0.030015	9.581317e-01	1.726501	1.683338	0.857143	0.965762	0.835714	0.980083
3	4	0.040020	9.435820e-01	1.812826	1.715710	0.900000	0.951003	0.851786	0.972813
4	5	0.050025	9.280242e-01	1.740888	1.720746	0.864286	0.935716	0.854286	0.965394
5	6	0.100050	8.654409e-01	1.729378	1.725062	0.858571	0.896148	0.856429	0.930771
6	7	0.150004	8.079646e-01	1.645404	1.698534	0.816881	0.836560	0.843259	0.899398
7	8	0.200029	7.512582e-01	1.668951	1.691136	0.828571	0.779018	0.839586	0.869292
8	9	0.300007	6.415994e-01	1.546323	1.642876	0.767691	0.698003	0.815626	0.812209
9	10	0.399986	5.364349e-01	1.318838	1.561881	0.654753	0.589484	0.775415	0.756538
10	11	0.500036	4.504170e-01	1.047410	1.458943	0.520000	0.490087	0.724310	0.703225
11	12	0.600014	3.863177e-01	0.791978	1.347792	0.393138	0.417062	0.669128	0.655543
12	13	0.699993	3.286128e-01	0.640702	1.246800	0.318084	0.357151	0.618999	0.612924
13	14	0.799971	2.677154e-01	0.565833	1.161694	0.280915	0.298693	0.576738	0.573652
14	15	0.899950	1.947684e-01	0.411777	1.078383	0.204432	0.232155	0.535377	0.535714
15	16	1.000000	5.684382e-07	0.294944	1.000000	0.146429	0.129859	0.496463	0.495108

```
In [55]: #F1-score
print(metrics.f1_score(cardioTest.as_data_frame()[ "cardio"],predps.predict, pos_label=1))
```

0.7326002502139988

7. H2OAutoML

L'outil H2OAutoML (Automatic Machine Learning) correspond au même état d'esprit. Mieux même, il prétend détecter pour nous le meilleur modèle possible. Il procède toujours par validation croisée pour l'exploration de l'espace des solutions mais, au-delà de la recherche des paramètres optimaux, permet aussi de tester différentes familles d'algorithmes et de les combiner. « H2O's AutoML can be used for automating the machine learning workflow, which includes automatic training and tuning of many models within a user-specified time-limit. Stacked Ensembles – one based on all previously trained models, another one on the best model of each family – will be automatically trained on collections of individual models to produce highly predictive ensemble models which, in most cases, will be the top performing models in the AutoML Leaderboard. ». Le discours est séduisant, c'est le moins qu'on puisse dire. H2O met à la portée des néophytes toute la puissance de l'intelligence artificielle (je suis ébloui par ce que je viens d'écrire, argh...). Bon, trêve de plaisanterie, regardons ce que cela donne sur notre base « cardio ». Rien ne vaut l'épreuve du feu.

Paramétrage et instantiation. Il y a deux manières de limiter la recherche des solutions sous AutoML : fixer le nombre de modèles à essayer, ou fixer une durée limite des calculs. Nous trouvons cette seconde option particulièrement intéressante dans un contexte d'études réelles. On peut toujours essayer d'optimiser, mais on ne peut pas le faire indéfiniment. Sachant que la durée de calcul pour chaque méthode n'est pas toujours maîtrisée, pouvoir borner le temps que l'on consacre à l'exploration correspond à la pratique réelle des data scientifiques(Nous). Nous choisissons d'allouer 180 secondes (3 minutes) à AutoML puis nous faisons appel à train(). La performance de chaque modèle est mesurée en (nfolds = 5) validation croisée.

Modèle 5 : H2OAutoML

```
In [56]: #chargement de la classe
from h2o.automl import H2OAutoML

In [57]: #instanciation
aml = H2OAutoML(seed=100,nfolds=5,max_runtime_secs=180)

In [58]: #lancement des calculs
aml.train(x=x, y=y, training_frame=cardioTrain)

AutoML progress: |
20:23:05.31: AutoML: XGBoost is not available; skipping it.

██████████
20:23:27.531: GBM_1_AutoML_20210128_202305 [GBM def_1] failed: water.exceptions.H2OModelBuilderIllegalArgumentException: Illega
l argument(s) for GBM model: GBM_1_AutoML_20210128_202305_cv_1. Details: ERRR on field: _ntrees: The tree model will not fit i
n the driver node's memory ( 770 B per tree x 10000 > 2,0 MB) - try decreasing ntrees and/or max_depth or increasing min_rows!

██████████
20:23:31.242: GBM_2_AutoML_20210128_202305 [GBM def_2] failed: water.exceptions.H2OModelBuilderIllegalArgumentException: Illega
l argument(s) for GBM model: GBM_2_AutoML_20210128_202305_cv_1. Details: ERRR on field: _ntrees: The tree model will not fit i
n the driver node's memory (1,2 KB per tree x 10000 > 11,7 MB) - try decreasing ntrees and/or max_depth or increasing min_rows!

██████████
20:23:33.742: GBM_3_AutoML_20210128_202305 [GBM def_3] failed: water.exceptions.H2OModelBuilderIllegalArgumentException: Illega
l argument(s) for GBM model: GBM_3_AutoML_20210128_202305_cv_1. Details: ERRR on field: _ntrees: The tree model will not fit i
n the driver node's memory (2,1 KB per tree x 10000 > 13,7 MB) - try decreasing ntrees and/or max_depth or increasing min_rows!

██████████
20:23:36.576: GBM_4_AutoML_20210128_202305 [GBM def_4] failed: water.exceptions.H2OModelBuilderIllegalArgumentException: Illega
l argument(s) for GBM model: GBM_4_AutoML_20210128_202305_cv_1. Details: ERRR on field: _ntrees: The tree model will not fit i
n the driver node's memory (4,9 KB per tree x 10000 > 28,5 MB) - try decreasing ntrees and/or max_depth or increasing min_rows!
```

5 modèles ont été évalués, nous affichons leurs identifiants avec la valeur de l'AUC correspondante.

```
In [59]: #récupérer le tableau des modèles
lb = aml.leaderboard

In [60]: #nombre de modèles
print(lb.nrow) #35
5

In [61]: #afficher les modèles -- tri par défaut AUC pour le classement binaire
result = lb.head(rows=lb.nrow).as_data_frame()
result.loc[:,["model_id","auc"]]

Out[61]:
   model_id      auc
0  StackedEnsemble_BestOfFamily_AutoML_20210128_202305  0.783956
1  StackedEnsemble_AllModels_AutoML_20210128_202305  0.783908
2          GBM_5_AutoML_20210128_202305  0.765696
3          DRF_1_AutoML_20210128_202305  0.735211
4          GLM_1_AutoML_20210128_202305  0.712019
```

Un petit mot sur la manière de procéder d'AutoML pour bien comprendre ces résultats. Il essaie différentes familles d'algorithmes, couplés avec différentes valeurs des paramètres clés. Jusque-là pas de surprise. Ensuite il « empile », au sens du stacking1, des groupes de modèles : tous les modèles (StackedEnsemble_AllModels_AutoML), les meilleurs modèles de chaque famille (StackedEnsemble_BestOfFamily). Pour notre jeu de données, c'est ce dernier qui s'est révélé le plus intéressant au sens de l'AUC (0.8447 en validation croisée).

Nous pouvons afficher les propriétés du modèle leader.

```
In [62]: #meilleur modèle
aml.leader

Model Details
=====
H2StackedEnsembleEstimator : Stacked Ensemble
Model Key: StackedEnsemble_BestOfFamily_AutoML_20210128_202305

No model summary for this model

ModelMetricsBinomialGLM: stackedensemble
** Reported on train data. **

MSE: 0.15071948299201818
RMSE: 0.38822607201477105
LogLoss: 0.4751235241757313
Null degrees of freedom: 9939
Residual degrees of freedom: 9936
Null deviance: 13779.853592633595
Residual deviance: 9445.455660613537
AIC: 9453.455660613537
AUC: 0.885643242227588
AUCPR: 0.8870851323966201
Gini: 0.7712864484455175

Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.4635565176469616:

    0   1   Error      Rate
0   0  3934.0  1055.0  0.2115  (1055.0/4989.0)
1   1   872.0  4079.0  0.1761  (872.0/4951.0)
2 Total 4806.0  5134.0  0.1939  (1927.0/9940.0)
```

Maximum Metrics: Maximum metrics at their respective thresholds

	metric	threshold	value	idx
0	max f1	0.463557	0.808924	219.0
1	max f2	0.289765	0.869052	311.0
2	max f0point5	0.580201	0.828292	164.0
3	max accuracy	0.482159	0.806942	210.0
4	max precision	0.919440	1.000000	0.0
5	max recall	0.122218	1.000000	394.0
6	max specificity	0.919440	1.000000	0.0
7	max absolute_mcc	0.500040	0.613947	202.0
8	max min_per_class_accuracy	0.476800	0.805171	212.0
9	max mean_per_class_accuracy	0.482159	0.806924	210.0
10	max tns	0.919440	4989.000000	0.0
11	max fns	0.919440	4943.000000	0.0
12	max fps	0.103505	4989.000000	399.0
13	max tps	0.122218	4951.000000	394.0
14	max tnr	0.919440	1.000000	0.0
15	max fnr	0.919440	0.998384	0.0
16	max fpr	0.103505	1.000000	399.0
17	max tpr	0.122218	1.000000	394.0

Gains/Lift Table: Avg response rate: 49,81 %, avg score: 50,01 %

group	cumulative_data_fraction	lower_threshold	lift	cumulative_lift	response_rate	score	cumulative_response_rate	cumulative_score	capture_r	
0	1	0.01006	0.883787	2.007675	2.007675	1.000000	0.898123	1.000000	0.898123	0.020
1	2	0.02002	0.870832	1.987396	1.997586	0.989899	0.877054	0.994975	0.887641	0.019
2	3	0.03008	0.860339	1.967522	1.987531	0.980000	0.865244	0.989967	0.880151	0.019
3	4	0.04004	0.850436	2.007675	1.992542	1.000000	0.855378	0.992462	0.873989	0.019
4	5	0.05000	0.840975	1.906277	1.975359	0.949495	0.845608	0.983903	0.868335	0.018
5	6	0.10000	0.805348	1.934963	1.955161	0.963783	0.822788	0.973843	0.845562	0.096
6	7	0.15000	0.7711333	1.866290	1.925537	0.929577	0.788759	0.959088	0.826627	0.093
7	8	0.20000	0.739209	1.846092	1.905676	0.919517	0.755287	0.949195	0.808792	0.092
8	9	0.30000	0.668062	1.684508	1.831953	0.839034	0.704440	0.912475	0.774008	0.168
9	10	0.40000	0.574115	1.478489	1.743587	0.736419	0.622559	0.868461	0.736146	0.147
10	11	0.50000	0.475829	1.100788	1.615027	0.548290	0.525035	0.804427	0.693924	0.110
11	12	0.60000	0.398051	0.789739	1.477479	0.393360	0.436400	0.735915	0.651003	0.078
12	13	0.70000	0.334196	0.494850	1.337104	0.246479	0.365372	0.665996	0.610199	0.049
13	14	0.80000	0.275371	0.359523	1.214906	0.179074	0.304452	0.605131	0.571980	0.035
14	15	0.90000	0.217906	0.226217	1.105052	0.112676	0.248169	0.550414	0.536001	0.022
15	16	1.00000	0.102514	0.054534	1.000000	0.027163	0.177405	0.498089	0.500142	0.005

ModelMetricsBinomialGLM: stackedensemble
** Reported on cross-validation data. **

MSE: 0.18929272668449465
RMSE: 0.43507783979937964
LogLoss: 0.562530512850381
Null degrees of freedom: 56006
Residual degrees of freedom: 56003
Null deviance: 77644.3490855284
Residual deviance: 63011.29286642258
AIC: 63019.29286642258
AUC: 0.7839564658136137
AUCPR: 0.762930486611559
Gini: 0.5679129316272273

Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.3806691560716026:

	0	1	Error	Rate
0	0	16114.0	11861.0	0.424 (11861.0/27975.0)
1	1	4935.0	23097.0	0.176 (4935.0/28032.0)
2	Total	21049.0	34958.0	0.2999 (16796.0/56007.0)

Maximum Metrics: Maximum metrics at their respective thresholds

	metric	threshold	value	idx
0	max f1	0.380669	0.733355	257.0
1	max f2	0.202481	0.840255	350.0
2	max f0point5	0.575935	0.733556	170.0
3	max accuracy	0.505319	0.723142	199.0
4	max precision	0.920925	0.878788	5.0
5	max recall	0.073142	1.000000	398.0
6	max specificity	0.942585	0.999964	0.0
7	max absolute_mcc	0.526661	0.447685	190.0
8	max min_per_class_accuracy	0.479657	0.720962	211.0
9	max mean_per_class_accuracy	0.505319	0.723168	199.0
10	max tns	0.942585	27974.000000	0.0
11	max fns	0.942585	28026.000000	0.0
12	max fps	0.067912	27975.000000	399.0
13	max tps	0.073142	28032.000000	398.0
14	max tnr	0.942585	0.999964	0.0
15	max fnr	0.942585	0.999786	0.0
16	max fpr	0.067912	1.000000	399.0
17	max tpr	0.073142	1.000000	398.0

Gains/Lift Table: Avg response rate: 50,05 %, avg score: 50,05 %

group	cumulative_data_fraction	lower_threshold	lift	cumulative_lift	response_rate	score	cumulative_response_rate	cumulative_score	capture_r
0	1	0.010017	0.901120	1.688122	0.844920	0.914486	0.844920	0.914486	0.0161
1	2	0.020015	0.885228	1.687568	0.844643	0.892721	0.844781	0.903613	0.0161
2	3	0.030014	0.874301	1.684000	0.842857	0.879566	0.844140	0.895602	0.0161
3	4	0.040013	0.865285	1.666161	0.833929	0.869610	0.841589	0.889107	0.0161
4	5	0.050012	0.856760	1.687568	0.844643	0.860985	0.842199	0.883485	0.0161
5	6	0.100005	0.819206	1.691136	0.846429	0.837355	0.844314	0.860424	0.0841
6	7	0.149999	0.785273	1.646895	0.824286	0.801964	0.837638	0.840940	0.0821
7	8	0.200011	0.752190	1.590670	0.852844	0.796144	0.768917	0.827263	0.822931
8	9	0.299998	0.676185	1.532012	1.612571	0.766786	0.715138	0.807106	0.787004
9	10	0.400004	0.581822	1.312715	1.537604	0.657026	0.630318	0.769584	0.747831
10	11	0.500009	0.478432	1.058733	1.441826	0.529905	0.529929	0.721647	0.704249
11	12	0.599996	0.397153	0.840930	1.341689	0.420893	0.435549	0.671527	0.659471
12	13	0.700002	0.329106	0.697023	1.249589	0.348866	0.362513	0.625430	0.617046
13	14	0.799989	0.265196	0.568707	1.164488	0.284643	0.296510	0.582837	0.576984
14	15	0.899995	0.199907	0.426989	1.082539	0.213712	0.233751	0.541820	0.538844
15	16	1.000000	0.065114	0.257192	1.000000	0.128727	0.155460	0.500509	0.500504

Out[62]:

Les résultats en validation croisée sont ceux qui nous intéressent le plus. On nous annonce un F1-Score de 0.7310. Voyons ce qu'il en est en test.

Performance en test du meilleur modèle. Nous appliquons le modèle leader sur notre échantillon test et nous mesurons le F1-Score.

```
In [63]: #prédition sur Le test
predAml = aml.predict(cardioTest).as_data_frame()
print(predAml.head(10))

stackedensemble prediction progress: |███████████| 100%
   predict      p0      p1
0       0  0.380239  0.619761
1       0  0.621306  0.378694
2       0  0.694760  0.305240
3       1  0.561358  0.438642
4       1  0.299229  0.700771
5       1  0.252403  0.747597
6       1  0.380064  0.619936
7       1  0.523493  0.476507
8       1  0.595332  0.404668
9       1  0.511398  0.488602

In [64]: #F1-score
print(metrics.f1_score(cardioTest.as_data_frame()[ "cardio"],predAml.predict, pos_label=1))

0.7310084825636193
```

Empiler 5 modèles pour obtenir des performances équivalentes aux autres approches « simples », on se demande où est l'intérêt.... Encore une fois, c'est la démarche ici qui est intéressante. Après arrive toujours une limite où, malgré tous nos efforts, nous ne pouvons plus tirer de l'information utile (pour la prédition) des données, même en les triturant dans tous les sens.

8. CONCLUSION

Les résultats obtenus dans cette partie nous donnent de valeur de preuve, loin pour la prédition. Il s'agit d'une expérimentation sur notre base (CARDIO), avec une version de la subdivision apprentissage-test des données, où ce dernier (échantillon test) est de taille assez réduite quand même (13993 observations). Rappelons aussi que nous sciemment pénalisé le gradient boosting pour mieux mettre en valeur les outils additionnels de H2O.

Bon, il reste qu'un petit tableau récapitulatif permet d'avoir une vue globale de notre étude.

ML Méthode - Cardio Dataset	F1-Score
Random Forest	0.7212
Gradient Boosting	0.7387
Naïve Bayes	0.6871
Simple Perceptron	0.7326
H2OAutoML	0.7310

Le gradient boosting est le plus performant semble-t-il. Mais les écarts entre les modèles ne sont pas mirifiques non plus. Estimés sur un échantillon test de taille réduite, ils ne sont certainement pas significatifs.

IV. REALISATION

1. INTRODUCTION

Le problème de classification automatique, clustering, est un problème central pour explorer et analyser les larges volumes de données. Le Big Data est un concept qui se propose de traiter une énorme quantité de données selon les temps de réponses souhaités dans un système informatique distribué (cluster, cloud). L'algorithme K plus proche voisins (kNN) a une complexité temporelle et spatiale linéaire. L'algorithme standard séquentiel est inadapté pour la classification des Big Data.

Dans [Zhenyun Deng, Xiaoshu Zhun , Debo Cheng, Ming Zong, et Shichao Zhang, 2016], les auteurs ont proposé deux processus dans notre algorithme K-NN pour le classification de grands volumes de données. Le processus d'apprentissage est conçu pour sélectionner un cluster le plus proche pour chaque échantillon de test en tant que nouvel ensemble de données d'apprentissage, et le processus de test est utilisé pour classer chaque échantillon de test par algorithme kNN dans son cluster le plus proche.

2. ALGORITHME DE K-NN POUR TRAITE LE BIG DATA

Inspiré par les progrès de Big Data, cet article a conçu une nouvelle méthode kNN pour traiter le Big Data. Nous proposons de procéder d'abord à un clustering de k-means pour séparer l'ensemble de données en plusieurs parties. Ensuite, nous sélectionnons le cluster le plus proche comme échantillons d'apprentissage et effectuons la classification kNN. La complexité temporelle de l'algorithme proposé est linéaire à la taille de l'échantillon.

a) Processus d'apprentissage

Le clustering est conçu pour regrouper un ensemble d'échantillons de telle sorte que les échantillons du même groupe (cluster) soient plus similaires les uns aux autres qu'à ceux des autres groupes. Autrement dit, les échantillons présentent une forte similitude au sein d'un cluster et une faible similitude entre les clusters. Les méthodes de clustering récentes peuvent être réparties dans les catégories suivantes: clustering basé sur la densité, clustering basé sur le cluster, clustering de partitionnement et clustering hiérarchique, respectivement. Même si les méthodes de clustering précédentes ont montré de bonnes performances, mais leur applicabilité au big data est limitée en raison de leur complexité de calcul élevée. Pour résoudre ce problème, cet article envisage d'employer un algorithme de clustering satisfaisant les deux avantages suivants:

- Faible complexité.
- Échelles linéairement.

Nous avons utilisé le clustering spectral basé sur le Landmark (LSC) dans cet article. La raison d'être de LSC est sélectionner p (n) échantillon représentatif comme points de repère et représente les échantillons originaux sous forme de combinaisons linéaires de ces points de repère. Différent de cette méthode traditionnelle de regroupement spectral qui utilise des échantillons entiers pour représenter chaque échantillon, la LSC significative réduit la complexité de la matrice d'affinité. La complexité de la résolution de la valeur propre à des échelles linéaires. Tout d'abord, nous traitons chaque échantillon comme un vecteur de base

pour construire la matrice de repère Z. Notez que LSC utilise p points de repère pour représenter chaque échantillon $X = [X_1, X_2, \dots, X_n] \in R^{(m \times n)}$. Nous devons trouver la matrice W qui est la matrice de projection de X à la matrice de repère Z.

$$w_{ji} = \frac{K_h(\mathbf{x}_i, \mathbf{z}_j)}{\sum_{j' \in Z_{*}} K_h(\mathbf{x}_i, \mathbf{z}_{j'})}, \quad j \in Z_{*}**$$

Ensuite, nous effectuons une analyse spectrale sur un graphique basé sur des points de repère et calculons la matrice du graphique comme suit:

$$\mathbf{G} = \hat{\mathbf{W}}^T \hat{\mathbf{W}}$$

Soit la décomposition en valeur singulière (SVD) de \mathbf{W}^T est la suivante:

$$\hat{\mathbf{W}} = \mathbf{U} \Sigma \mathbf{V}^T$$

Nous calculons U dans O (p3), linéaire à la taille de l'échantillon. V peut être calculé comme:

$$\mathbf{V}^T = \Sigma^{-1} \mathbf{U}^T \hat{\mathbf{W}}$$

En raison de la complexité temporelle de O (n³) à O (n), l'algorithme LSC réduit considérablement le temps de calcul. Ainsi, l'algorithme proposé à faible complexité est très adapté pour être appliqué dans le domaine du big data. Enfin, le pseudo de LSC est présenté dans l'algorithme 1.

Algorithm 1. The pseudo of LSC.

Input: n data points $x_1, x_2, \dots, x_n \in R^m$; Cluster number k ;

Output: k clusters;

- 1 Produce p landmark points using the k -means method;
- 2 Construct a landmark matrix Z between data points and landmark samples, with the affinity calculated according to Eq. (1);
- 3 Compute the first k eigenvectors of $\mathbf{W}\mathbf{W}^T$, denoted by $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_k]$;
- 4 Compute $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k]$ according to Eq. (4);
- 5 Each row of \mathbf{V} is a data point and apply k -means to get the clusters.

Algorithme 4.1 : L'algorithme de LSC.

b) Processus de test

Nous produisions déjà k clusters et centres de cluster par algorithme LSC, nous trouvons alors le centre de cluster le plus proche pour chaque échantillon de test et utilisons le cluster

correspondant comme nouvel ensemble de données d'apprentissage pour chaque échantillon de test. En raison de la réalisation des clusters avec une forte similitude au sein d'un cluster, nous appliquons kNN pour classer les échantillons de test dans le nouvel ensemble de données d'apprentissage. L'algorithme proposé garantit toujours une précision de classification relativement élevée. Nous listons le pseudo de notre méthode proposée dans l'algorithme 2.

Algorithm 2. The pseudo of LC-kNN algorithm.

Input: Training dataset, test samples Y ;

Output: Class label;

- 1 Produce m cluster centers using LSC algorithm, denoted by $C_1, C_2, C_3, \dots, C_m$;
- 2 Compute the distance $D(y, C_i)$ between test sample y and each cluster center, denoted by $D(y, C_i)$, $i = 1, 2, \dots, m$;
- 3 Compute the nearest cluster center C_i to y , $C_i = \min\{D(y, C_i)\}$, $i = 1, 2, \dots, m$;
- 4 Using the corresponding cluster of C_i as the training dataset, denoted by $NewX_i$;
- 5 Apply to kNN algorithm to predict y in the training dataset.

Algorithme 4.2 : L'algorithme de LC-KNN.

À partir de l'algorithme 2, le nombre de $NewX_i$ est beaucoup plus petit que la taille de l'ensemble de données d'apprentissage. Lorsque la taille de m est grande, LCkNN permet de réduire facilement le calcul de kNN et améliore la qualité de la classification. En général, plus la valeur de m est élevée, plus la classification, l'efficacité, ce qui conduit à une précision de classification plus élevée.

3. EVALUATION ET RESULTAT

Comme notre algorithme proposé est une extension de kNN. Ainsi, afin de montrer l'efficacité de l'algorithme LC-kNN, nous avons pris le kNN comme base de référence et avons fait une comparaison entre kNN, LC-kNN et RC-kNN, et plusieurs ensembles de données réels provenant d'UCI, d'ensembles de données d'imagerie médicale et d'ensembles de données LIBSVM.

a) Résultats expérimentaux de LC-kNN et RC-kNN avec différentes valeurs de m

La valeur de m est très importante pour l'algorithme LC-kNN, car elle affecte directement les performances finales et les applications réelles. Ainsi, afin de sélectionner m approprié pour apporter un grand effet à LC-kNN, un groupe d'expériences ont été menées sur des ensembles de données en choisissant différentes valeurs de m pour LC-kNN et RC-kNN. Les expériences LC-kNN et RC-kNN dans ce groupe ont été réalisées sur les ensembles de données avec $m = 10, 15, 20, 25$ et 30 .

Tableau 4.1 : Précision de classification et coût du temps sur l'ensemble de données USPS à différentes valeurs de m.

<i>m</i>	Criterion	RC-kNN	LC-kNN
10	Accuracy	$0.9027 \pm 1.6498e-005$	$0.9355 \pm 7.1306e-006$
	Time	3.5589 ± 0.0107	3.7605 ± 0.0242
15	Accuracy	$0.8964 \pm 5.1803e-005$	$0.9338 \pm 4.1625e-006$
	Time	2.4857 ± 0.0032	2.7260 ± 0.0077
20	Accuracy	$0.8770 \pm 7.4889e-005$	$0.9300 \pm 4.9238e-006$
	Time	2.3202 ± 0.0010	2.5157 ± 0.01928
25	Accuracy	$0.8793 \pm 4.9917e-005$	$0.9284 \pm 1.0637e-005$
	Time	1.8586 ± 0.0008	1.9971 ± 0.0042
30	Accuracy	$0.8607 \pm 4.6629e-005$	$0.9275 \pm 1.1596e-005$
	Time	1.6441 ± 0.0002	1.9249 ± 0.0023

Tableau 4.2 : Précision de classification et coût du temps sur l'ensemble de données MNIST à différentes valeurs de m.

<i>m</i>	Criterion	RC-kNN	LC-kNN
10	Accuracy	$0.7221 \pm 4.8878e-005$	$0.8389 \pm 3.1656e-005$
	Time	2.9369 ± 0.0508	3.5504 ± 0.0927
15	Accuracy	$0.6840 \pm 2.3333e-004$	$0.8364 \pm 2.3136e-005$
	Time	2.8905 ± 0.0456	3.1222 ± 0.01397
20	Accuracy	$0.6657 \pm 2.4739e-004$	$0.8353 \pm 3.3233e-005$
	Time	2.0564 ± 0.0011	2.1490 ± 0.0065
25	Accuracy	$0.6478 \pm 2.2689e-004$	$0.8338 \pm 8.7844e-005$
	Time	1.8240 ± 0.0020	2.1148 ± 0.0094
30	Accuracy	$0.6396 \pm 6.9156e-005$	$0.8313 \pm 3.8678e-005$
	Time	1.5457 ± 0.0002	1.7274 ± 0.0011

Tableau 4.3 : Précision de classification et coût du temps sur l'ensemble de données GISETTE à différentes valeurs de m.

<i>m</i>	Criterion	RC-kNN	LC-kNN
10	Accuracy	$0.9311 \pm 5.0989e-005$	$0.9526 \pm 1.4511e-005$
	Time	23.3933 ± 0.9677	28.5940 ± 3.2405
15	Accuracy	$0.9252 \pm 1.0573e-004$	$0.9494 \pm 1.3378e-005$
	Time	18.0106 ± 0.2434	23.1904 ± 1.0894
20	Accuracy	$0.9166 \pm 2.8267e-005$	$0.9411 \pm 5.4699e-004$
	Time	12.7685 ± 0.0966	16.2759 ± 0.8880
25	Accuracy	$0.9150 \pm 7.0000e-005$	$0.9321 \pm 6.4810e-004$
	Time	9.9201 ± 0.3696	13.8645 ± 1.5093
30	Accuracy	$0.9079 \pm 1.0366e-004$	$0.9192 \pm 5.3796e-004$
	Time	8.4064 ± 0.0784	11.3922 ± 0.0658

Tableau 4.4 : Précision de classification et coût du temps sur l'ensemble de données LETTER à différentes valeurs de m.

<i>m</i>	Criterion	RC- <i>k</i> NN	LC- <i>k</i> NN
10	Accuracy	$0.7892 \pm 3.8822e-005$	$0.9495 \pm 1.0760e-006$
	Time	3.2391 ± 0.0015	3.2994 ± 0.0010
15	Accuracy	$0.7932 \pm 3.7106e-005$	$0.9469 \pm 5.5751e-006$
	Time	3.3808 ± 0.0435	3.4334 ± 0.0585
20	Accuracy	$0.6815 \pm 1.3812e-004$	$0.9451 \pm 1.9756e-006$
	Time	$3.0938 \pm 5.8392e-004$	$3.1285 \pm 3.1243e-004$
25	Accuracy	$0.7279 \pm 5.6480e-005$	$0.9423 \pm 5.2818e-006$
	Time	3.3950 ± 0.0018	3.4813 ± 0.0054
30	Accuracy	$0.6214 \pm 9.8480e-005$	$0.9403 \pm 3.9204e-006$
	Time	$3.0889 \pm 1.3000e-003$	$3.1168 \pm 3.8514e-004$

Tableau 4.5 : Précision de classification et coût du temps sur l'ensemble de données PENDIGITS à différentes valeurs de m.

<i>m</i>	Criterion	RC- <i>k</i> NN	LC- <i>k</i> NN
10	Accuracy	$0.9452 \pm 3.5382e-005$	$0.9721 \pm 4.7991e-006$
	Time	2.3380 ± 0.0041	2.4056 ± 0.0101
15	Accuracy	$0.9316 \pm 1.0341e-004$	$0.9711 \pm 6.0196e-006$
	Time	2.5451 ± 0.0011	2.5709 ± 0.0089
20	Accuracy	$0.9163 \pm 1.5515e-004$	$0.9700 \pm 2.5390e-006$
	Time	$2.2233 \pm 6.4795e-005$	$2.2554 \pm 2.1569e-004$
25	Accuracy	$0.9216 \pm 1.5677e-004$	$0.9687 \pm 3.5642e-006$
	Time	2.5270 ± 0.0056	2.5468 ± 0.0083
30	Accuracy	$0.9088 \pm 1.8409e-004$	$0.9683 \pm 1.5809e-006$
	Time	$2.1805 \pm 7.4785e-005$	$2.2022 \pm 8.9611e-005$

Tableau 4.6 : Précision de classification et coût du temps sur l'ensemble de données SATIMAGE à différentes valeurs de m.

<i>m</i>	Criterion	RC- <i>k</i> NN	LC- <i>k</i> NN
10	Accuracy	$0.8603 \pm 8.9122e-005$	$0.8883 \pm 8.1139e-006$
	Time	$1.2868 \pm 6.5495e-005$	$1.3027 \pm 1.1429e-004$
15	Accuracy	$0.7917 \pm 3.1680e-005$	$0.9468 \pm 3.7244e-006$
	Time	3.8583 ± 0.0332	3.9337 ± 0.0152
20	Accuracy	$0.8418 \pm 8.8847e-005$	$0.8884 \pm 6.8028e-006$
	Time	$1.2292 \pm 3.2277e-005$	$1.2463 \pm 4.3126e-005$
25	Accuracy	$0.7283 \pm 1.1039e-004$	$0.9421 \pm 8.8449e-006$
	Time	3.5062 ± 0.0061	3.6287 ± 0.0052
30	Accuracy	$0.8312 \pm 3.5146e-004$	$0.8878 \pm 4.9556e-006$
	Time	$1.2225 \pm 1.8176e-005$	$1.2396 \pm 1.7711e-005$

Tableau 4.7 : Précision de classification et coût du temps sur l'ensemble de données ADNC à différentes valeurs de m.

<i>m</i>	Criterion	RC- <i>k</i> NN	LC- <i>k</i> NN
10	Accuracy	0.7024 ± 0.0010	0.7667 ± 0.0019
	Time	$0.0310 \pm 2.7390e-004$	$0.0333 \pm 1.9201e-005$
15	Accuracy	0.6857 ± 0.0014	0.7500 ± 0.0013
	Time	$0.0308 \pm 7.9360e-006$	$0.0325 \pm 3.3635e-005$
20	Accuracy	0.6619 ± 0.0029	0.7143 ± 0.0028
	Time	$0.0295 \pm 3.0063e-006$	$0.0313 \pm 6.8908e-006$
25	Accuracy	0.6548 ± 0.0039	0.7071 ± 0.0038
	Time	$0.0281 \pm 4.8219e-007$	$0.0286 \pm 7.0881e-007$
30	Accuracy	0.6619 ± 0.0015	0.7190 ± 0.0031
	Time	$0.0306 \pm 1.2641e-005$	$0.0326 \pm 5.3013e-006$

Tableau 4.8 : Précision de classification et coût du temps sur l'ensemble de données psMCI à différentes valeurs de m.

<i>m</i>	Criterion	RC- <i>k</i> NN	LC- <i>k</i> NN
10	Accuracy	0.4792 ± 0.0082	0.5833 ± 0.0089
	Time	$0.0168 \pm 4.5147e-007$	$0.0171 \pm 1.4130e-006$
15	Accuracy	0.5125 ± 0.0019	0.6042 ± 0.0040
	Time	$0.0167 \pm 9.4536e-007$	$0.0173 \pm 5.6957e-007$
20	Accuracy	0.5458 ± 0.0060	0.6500 ± 0.0035
	Time	$0.0170 \pm 1.2416e-006$	$0.0188 \pm 1.1793e-005$
25	Accuracy	0.5333 ± 0.0053	0.6417 ± 0.0035
	Time	$0.0163 \pm 4.9822e-007$	$0.0286 \pm 7.0881e-007$
30	Accuracy	0.5000 ± 0.0042	0.6125 ± 0.0019
	Time	$0.0166 \pm 8.6385e-007$	$0.0250 \pm 2.2840e-004$

Tableau 4.9 : Précision de classification et coût du temps sur l'ensemble de données MCINC à différentes valeurs de m.

<i>m</i>	Criterion	RC- <i>k</i> NN	LC- <i>k</i> NN
10	Accuracy	0.5476 ± 0.0049	0.6159 ± 0.0045
	Time	$0.0468 \pm 2.8577e-005$	$0.0506 \pm 5.9543e-005$
15	Accuracy	0.5397 ± 0.0032	0.5633 ± 0.0336
	Time	$0.0505 \pm 2.7585e-005$	$0.0538 \pm 4.6465e-005$
20	Accuracy	0.5458 ± 0.0060	0.6500 ± 0.0035
	Time	$0.0452 \pm 4.9222e-006$	$0.0480 \pm 1.1608e-005$
25	Accuracy	0.5222 ± 0.0015	0.5746 ± 0.0008
	Time	$0.0489 \pm 1.5349e-005$	$0.0540 \pm 4.5819e-005$
30	Accuracy	0.5016 ± 0.0245	0.5984 ± 0.0021
	Time	$0.0490 \pm 2.6160e-005$	$0.0536 \pm 4.5625e-005$

Tableau 4.10 : Précision de classification et coût en temps de trois algorithmes sur neuf ensembles de données.

Dataset	RC-kNN		LC-kNN		kNN	
	Accuracy	Time	Accuracy	Time	Accuracy	Time
USPS	0.9027	3.5589	0.9355	3.7605	0.9482	32.8764
MNIST	0.7221	2.9369	0.8389	3.5504	0.8635	24.1575
GISETTE	0.9311	23.3933	0.9526	28.594	0.9660	217.3327
LETTER	0.7892	3.2391	0.9495	3.2994	0.9518	19.8246
PENDIGITS	0.9452	2.3380	0.9721	2.4056	0.9780	7.2982
SATIMAGE	0.8603	1.2868	0.8883	1.3027	0.9065	3.5525
ADNC	0.7024	0.0310	0.7667	0.0333	0.7857	0.0355
psMCI	0.4792	0.0168	0.5833	0.0171	0.6042	0.0176
MCINC	0.5476	0.0468	0.6159	0.0506	0.6413	0.0575

Nous avons constaté que deux algorithmes ont une précision de classification élevée avec le nombre plus important de clusters m, tandis que ces deux algorithmes ont une précision de classification faible pour le petit nombre de m.

b) Pour déterminer le paramètre k.

Afin de sélectionner la valeur k appropriée, un groupe d'expériences a été mené sur neuf ensembles de données avec m = 1 à 20, chaque point représentant la moyenne de 10 résultats dans les figures X. Avec l'augmentation de la valeur k, la précision globale de la classification diminue. En effet, plus le jeu de données d'apprentissage est petit, plus la précision de classification est élevée. Par conséquent, la différence entre les échantillons est significative et la précision de classification réduira. La précision de classification était de 0,8986 avec k = 3, ce qui était supérieur à k = 1 sur l'ensemble de données satimage. Mais la performance LC-kNN est supérieure à kNN avec k = 1 dans le tableau 10.

c) Comparaison des performances de kNN, RC-kNN et LC-kNN

Dans cette expérience, selon l'analyse précédente, nous fixons m = 10, k = 1. Ensuite, nous utilisons la précision de classification et la durée d'exécution comme évaluations pour la tâche de classification. Plus algorithme est court et précis, meilleures sont les performances. Nous pouvons observer que le RC-kNN et le LC-kNN sont améliorés de 7 à 9 fois par rapport au kNN, en termes de coût en temps.

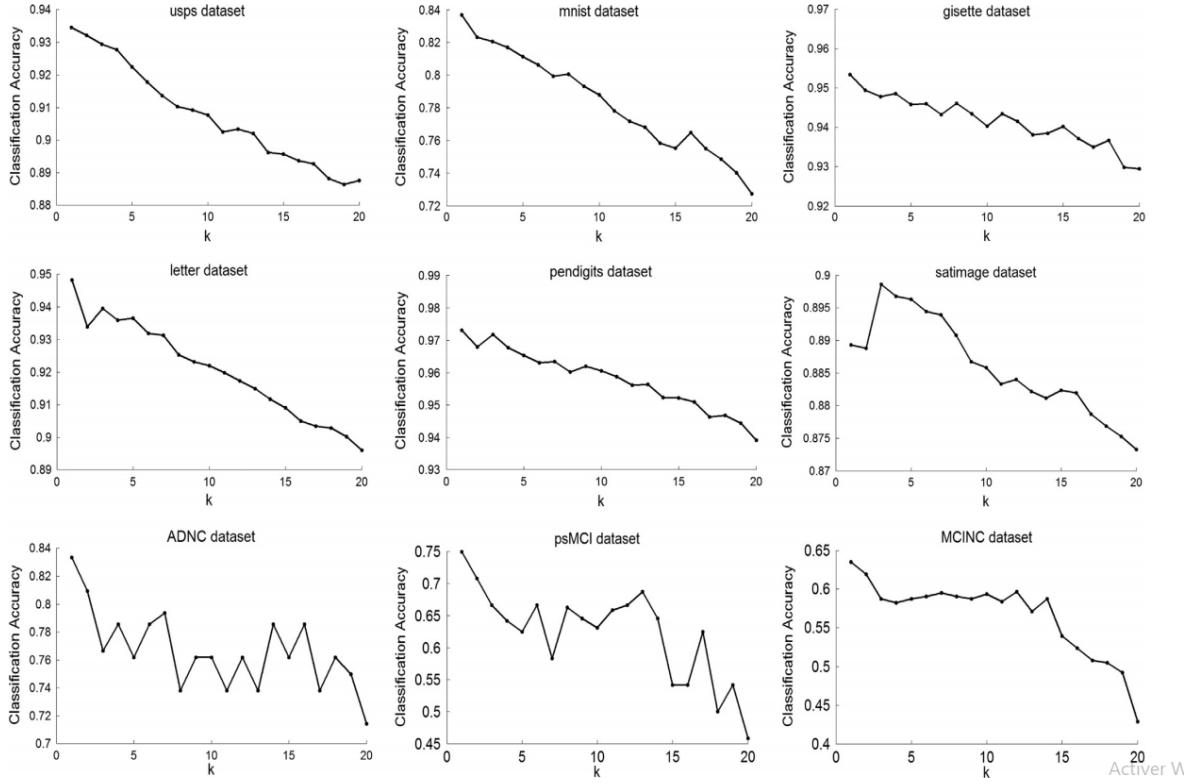


Figure 4.1 : Précision de classification de LC-kNN sur neuf ensembles de données avec k différents.

Selon les résultats expérimentaux, nous pouvons conclure que le LC-kNN fonctionne bien en termes de précision de classification et de temps.

4. CONCLUSION

Nous avons proposé une classification kNN efficace pour effectuer un regroupement de k -means pour séparer l'ensemble de données en plusieurs parties. Nous avons ensuite procédé à une classification kNN pour chaque partie. Pour ce faire, nous avons divisé la méthode conventionnelle kNN en deux processus, à savoir le processus d'apprentissage et le processus de test. De plus, nous avons analysé la valeur appropriée pour les paramètres m et k . Nous avons pris le kNN comme base de référence et mené un groupe d'expériences comparées entre kNN, LC-kNN et RC-kNN. Les résultats expérimentaux ont montré que la classification kNN proposée fonctionnait bien en termes de précision et d'efficacité, et qu'il était approprié de traiter de big data.

CONCLUSION GENERAL

Nous avons entendu parler de H2O depuis un moment déjà. Après un rapide coup d'œil, nous n'avons pas constatés d'éléments qui pourraient justifier l'écriture d'un tutoriel spécifique à son sujet. Plus récemment, nous l'avons de nouveau inspectée de plus près en nous intéressant aux packages pour la classification. Nous finalement décidé de nous pencher attentivement sur la plateforme avec notre premier objectif l'évaluation de sa capacité à paralléliser les algorithmes de machine learning. Dans un deuxième temps, à force de nous escrimer dessus, nous avons rendu compte que H2O proposait des fonctionnalités intéressantes pour la machine learning. Les structures de données H2O et ses algorithmes ont été construits avec des clusters à l'esprit.

Nous avons proposé une classification kNN efficace pour effectuer un regroupement de k-means pour séparer l'ensemble de données en plusieurs parties. Nous avons ensuite procédé à une classification kNN pour chaque partie. Pour ce faire, nous avons divisé la méthode conventionnelle kNN en deux processus, à savoir le processus d'apprentissage et le processus de test. De plus, nous avons analysé la valeur appropriée pour les paramètres m et k. Nous avons pris le kNN comme base de référence et mené un groupe d'expériences comparées entre kNN, LC-kNN et RC-kNN. Les résultats expérimentaux ont montré que la classification kNN proposée fonctionnait bien en termes de précision et d'efficacité, et qu'il était approprié de traiter de big data.

REFERENCES

- [1] Zhenyun Deng, Xiaoshu Zhun , Debo Cheng, Ming Zong, et Shichao Zhang, 2016: Efficient kNN classification algorithm for big data
- [2] Nicole Tache 2017 : Practical Machine Learning with H2O
- [3] H2O.ai – 3.22.1.1, <http://docs.h2o.ai/h2o/latest-stable/h2o-docs/index.html> (© Copyright 2016-2018 H2O.ai. Last updated on Dec 28, 2018.)
- [4] H2O Tutorials, <http://docs.h2o.ai/h2o-tutorials/latest-stable/index.html>
- [5] The H2O Python Module, <http://h2o.release.s3.amazonaws.com/h2o/master/3574/docs-website/h2opy/docs/intro.html> (© Copyright 2015, H2O.ai.).

IMPLEMENTATION DE L'ALGORITHME DE LSC

Subdivision en échantillons d'apprentissage et de test

```
In [14]: #subdivision
cardioTrain,cardioTest = cardio.split_frame(ratios=[0.8],seed=1)

In [15]: #vérification train
cardioTrain.shape

Out[15]: (56007, 13)

In [16]: #vérification test
cardioTest.shape

Out[16]: (13993, 13)
```

IMPLEMENTATION DES ALGORITHMES

ALGORITHME DE LSC

```
In [25]: import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
from sklearn.metrics.pairwise import pairwise_distances
import scipy

def find_p(X, start=1, end=10):
    min_size=[]
    number_of_clusters=[]
    for i in range(start,end+1):
        min_size.append(i)
        number_of_clusters.append(KMeans(n_clusters=i).fit(X).inertia_)
    _, ax=plt.subplots()
    ax.set(ylabel='Inertia', xlabel='Number of clusters', title='The elbow method')
    plt.xticks(np.arange(start,end, 1))
    plt.plot(min_size,number_of_clusters)
    plt.show()

def get_Landmarks(X, p, method="random"):
    if method=="random":
        N = len(X)
        perm= np.random.permutation(np.arange(N))
        print(perm)
        landmarks = X[perm[:p],:]
        return landmarks
    else:
        kmeans_model=KMeans(n_clusters=p).fit(X)
        return kmeans_model.cluster_centers_

def gaussian_kernel(dist_mat, bandwidth):
    return np.exp(-dist_mat / (2*bandwidth**2))

def compose_Sparse_ZHat_Matrix(X, landmarks, bandwidth, r):
    dist_mat=pairwise_distances(X,landmarks)
    sim_mat=gaussian_kernel(dist_mat, bandwidth)

    Zhat = np.zeros(sim_mat.shape)

    for i in range(Zhat.shape[0]):
        #may need j.sort.selectperm
        top_Landmarks_indices = np.argsort(-sim_mat[i,:])[:r]
        top_Landmarks_coefs = sim_mat[i,top_Landmarks_indices]
        top_Landmarks_coefs /= np.sum(top_Landmarks_coefs)
        Zhat[i, top_Landmarks_indices] = top_Landmarks_coefs
    #May be wrong
    diagm=np.sum(Zhat, axis=0)**(-1/2)
    return diagm*Zhat

def LSC_Clustering(X, n_clusters, n_landmarks, method, non_zero_landmark_weights, bandwidth):
    landmarks = get_Landmarks(X, n_landmarks, method)
    Zhat = compose_Sparse_ZHat_Matrix(X, landmarks, bandwidth, non_zero_landmark_weights)
    svd_result = np.linalg.svd(Zhat, full_matrices=False)[0]
    clustering_result = KMeans(n_clusters=n_clusters).fit(svd_result)
    return clustering_result
```

```

test1=False
if test1:
    X,y= cardioTrain.as_np(),cardioTest.as_np()
    landmarks_rand=get_Landmarks(X,2)
    print(landmarks_rand)

    landmarks_kmeans=get_Landmarks(X,2,"KMeans")
    print(landmarks_kmeans)
    find_p(X)
    print(compose_Sparse_ZHat_Matrix(X, landmarks_kmeans, 1, 5))

test2=True
if test2:
    X,y=cardioTrain.as_np(),cardioTest.as_np()
    labels=LSC_Clustering(X, 2, 4, "Kmeans", 4, 0.5).labels_
    plt.scatter(X[:,0],X[:,1], c=labels)
    plt.show()

-----
AttributeError: Traceback (most recent call last)
<ipython-input-25-ab59f6767f29> in <module>
    73 test2=True
    74 if test2:
--> 75     X,y=cardioTrain.as_np(),cardioTest.as_np()
    76     labels=LSC_Clustering(X, 2, 4, "Kmeans", 4, 0.5).labels_
    77

AttributeError: 'H2OFrame' object has no attribute 'as_np'

```

Subdivision en échantillons d'apprentissage et de test

```

In [38]: print(cardio[:,0:12])
[[1.8393e+04 2.0000e+00 1.6800e+02 ... 0.0000e+00 1.0000e+00 0.0000e+00]
 [2.0228e+04 1.0000e+00 1.5600e+02 ... 0.0000e+00 1.0000e+00 1.0000e+00]
 [1.8857e+04 1.0000e+00 1.6500e+02 ... 0.0000e+00 0.0000e+00 1.0000e+00]
 ...
 [1.9066e+04 2.0000e+00 1.8300e+02 ... 1.0000e+00 0.0000e+00 1.0000e+00]
 [2.2431e+04 1.0000e+00 1.6300e+02 ... 0.0000e+00 0.0000e+00 1.0000e+00]
 [2.0540e+04 1.0000e+00 1.7000e+02 ... 0.0000e+00 1.0000e+00 0.0000e+00]

In [39]: #X matrice des var. explicatives
X = cardio[:,0:11]

In [40]: print(cardio[:,11])
[0. 1. 1. ... 1. 1. 0.]

In [41]: #y vecteur de la var. à prédire
y = cardio[:,11]

```

ALGORITHME DE LSC

```

In [52]: import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
from sklearn.metrics.pairwise import pairwise_distances
import scipy

def find_p(X, start=1, end=10):
    min_size=[]
    number_of_clusters=[]
    for i in range(start,end+1):
        min_size.append(i)
        number_of_clusters.append(KMeans(n_clusters=i).fit(X).inertia_)
    _, ax=plt.subplots()
    ax.set(ylabel='Inertia', xlabel='Number of clusters', title='The elbow method')
    plt.xticks(np.arange(start,end,1))
    plt.plot(min_size,number_of_clusters)
    plt.show()

def get_Landmarks(X, p, method="random"):
    if method=="random":
        N = len(X)
        perm= np.random.permutation(np.arange(N))
        print(perm)
        landmarks = X[perm[:p],:]
        return landmarks
    else:
        kmeans_model=KMeans(n_clusters=p).fit(X)
        return kmeans_model.cluster_centers_

def gaussian_kernel(dist_mat, bandwidth):
    return np.exp(-dist_mat / (2*bandwidth**2))

```

```

def gaussian_kernel(dist_mat, bandwidth):
    return np.exp(-dist_mat / (2*bandwidth**2))

def compose_Sparse_ZHat_Matrix(X, landmarks, bandwidth, r):
    dist_mat=pairwise_distances(X,landmarks)
    sim_mat=gaussian_kernel(dist_mat, bandwidth)

    Zhat = np.zeros(sim_mat.shape)

    for i in range(Zhat.shape[0]):
        #may need j.sort.selectperm
        top_Landmarks_indices = np.argsort(-sim_mat[i,:])[:r]
        top_Landmarks_coefs = sim_mat[i,top_Landmarks_indices]
        top_Landmarks_coefs /= np.sum(top_Landmarks_coefs)
        Zhat[i, top_Landmarks_indices] = top_Landmarks_coefs
    #May be wrong
    diagm=np.sum(Zhat, axis=0)**(-1/2)
    return diagm*Zhat

def LSC_Clustering(X, n_clusters, n_landmarks, method, non_zero_landmark_weights, bandwidth):
    landmarks = get_Landmarks(X, n_landmarks, method)
    Zhat = compose_Sparse_ZHat_Matrix(X, landmarks, bandwidth, non_zero_landmark_weights)
    svd_result = np.linalg.svd(Zhat, full_matrices=False)[0]
    clustering_result = KMeans(n_clusters=n_clusters).fit(svd_result)
    return clustering_result

#Test get_Landmarks
test1=False
if test1:
    X,y= X,y

    landmarks_rand=get_Landmarks(X,2)
    print(landmarks_rand)

    landmarks_kmeans=get_Landmarks(X,2,"KMeans")
    print(landmarks_kmeans)
    find_p(X)
    print(compose_Sparse_ZHat_Matrix(X, landmarks_kmeans, 1, 5))

test2=True
if test2:
    X,y=X,y
    labels=LSC_Clustering(X,2, 4, "Kmeans", 4, 0.5).labels_

    plt.scatter(X[:,0],X[:,1], c=labels)
    plt.show()

C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:48: RuntimeWarning: invalid value encountered in true_divide

```

```

C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:48: RuntimeWarning: invalid value encountered in true_divide

-----
LinAlgError                                Traceback (most recent call last)
<ipython-input-52-8589ebdaf578> in <module>
    74 if test2:
    75     X,y=X,y
--> 76     labels=LSC_Clustering(X,2, 4, "Kmeans", 4, 0.5).labels_
    77
    78     plt.scatter(X[:,0],X[:,1], c=labels)

<ipython-input-52-8589ebdaf578> in LSC_Clustering(X, n_clusters, n_landmarks, method, non_zero_landmark_weights, bandwidth)
    55     landmarks = get_Landmarks(X, n_landmarks, method)
    56     Zhat = compose_Sparse_ZHat_Matrix(X, landmarks, bandwidth, non_zero_landmark_weights)
--> 57     svd_result = np.linalg.svd(Zhat, full_matrices=False)[0]
    58     clustering_result = KMeans(n_clusters=n_clusters).fit(svd_result)
    59     return clustering_result

<__array_function__ internals> in svd(*args, **kwargs)

~\AppData\Roaming\Python\Python37\site-packages\numpy\linalg\linalg.py in svd(a, full_matrices, compute_uv, hermitian)
    159
    160     signature = 'D->DdD' if isComplexType(t) else 'd->dd'
--> 161     u, s, vh = gufunc(a, signature=signature, extobj=extobj)
    162     u = u.astype(result_t, copy=False)
    163     s = s.astype(_realtype(result_t), copy=False)

~\AppData\Roaming\Python\Python37\site-packages\numpy\linalg\linalg.py in _raise_linalgerror_svd_nonconvergence(err, flag)
    95
    96 def _raise_linalgerror_svd_nonconvergence(err, flag):
--> 97     raise LinAlgError("SVD did not converge")
    98
    99 def _raise_linalgerror_lstsq(err, flag):

LinAlgError: SVD did not converge

```

In [53]: np.__version__
Out[53]: '1.19.5'