



version mise à jour

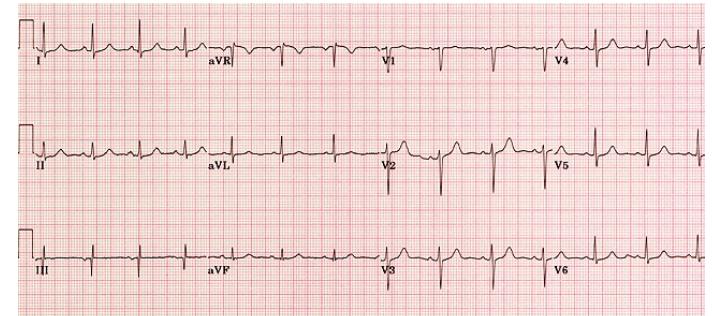


Apprenant : Lepicaut  
Objet : Technologies pour l'intelligence artificielle  
Période : Centre de formation  
Jury Académique : Philippe Besse . Beatrice Laurent-Bonneau . Brendan Guillouet  
Intervenant : Philippe BESSE  
Date : 25/03/19

**Mots clefs** : signal, ondelette, fourrier, classifieur, Accuracy, AUC

#### CADRAGE

date	20 février 2019
version	<b>25 mars 2019</b>
source des données	Signaux bruts, coefficients ondelettes et Transformées de fourrier
nombre de classes	k = 5 avec déséquilibre de classes
nombre d'enregistrements	87554 (training) + 6774 (test)
périmètre	Apprentissage statistique > machine learning + deep learning
problème	Classifieur Multi-classes (k>2) sur des données de signal ECG
type d'analyse	Méthode d'apprentissage machine supervisée
langage de programmation	Langage Python – Scikit-learn - TensorFlow - Keras
<b>Note destinée au lecteur</b>	résultats clefs



## **- SOMMAIRE -**

(1) Classification binaire sur signaux bruts	[Didier]	p. 3
(2) Classification multi-classes sur signaux bruts	[Omar]	p. 9
(3) Annexes		p. 15

Problème: classification supervisée multi-classes ( $k=5$ ) avec classes déséquilibrées (%) [N.82.7; Q.7.35; V.6.61 ; S.2.54; F.0.73 ]

signalétique: N = rythme Normal ; Q-V-S-F = arythmie (problème de rythme cardiaque)

Données: électrocardiogramme de battement de cœur [ECG], signaux bruts,

Nombre  $X_i = 188$

Nombre d'enregistrements = 87554 (training) + 6774 (test)

Définition: cœur bat trop vite ( $>100$  bat./min), on parle de tachycardie, trop lentement ( $<60$  bat./min), on parle de bradycardie

Nature du déséquilibre: fort pour les classes S et F

Type de classification: non en grande dimension (Nb  $X_i <$  Nb enreg.) et de moyenne taille (Pb de gestion mémoire : calcul in-memory)

\*\*\*\*

Audit du problème:

H0 : arythmie vs normal => axe de travail, critère prédictif => maximiser les métriques (Accuracy; AUC; autres) → Classif. Binaire k (5->2)  
versus

H1 : comprendre les k-1 arythmies vs normal => critère explicatif => trouver les techniques pour qualifier les k-1 classes arythmies [?]

Méthode de résolution: pas d'expertise métier sur les signaux de battements de cœur, je décide de travailler sur H0

. Transformer le problème multi-classes en binaire supervisé,

. Objectif = Développer les compétences techniques de réglages de classifieur en Machine et Deep Learning sur signaux bruts.

### 1. Preprocessing:

#### **Data brutes**

- . Stats univariées, bivariées,
- . Centrage-réduction,
- . Normalisation MinMaxScaler + test de shapiro-wilks (avant / après),
- . Shapiro-Wilks (train/test non gaussiens)
- . Matrice des corrélations Xi,
- . Traitements des Valeurs manquantes (0.00% train/test ),
- . Traitement des Outliers (~ 5.00% train/test - initial) ➔ lignes supprimées,

#### **Data rééquilibrées**

- . Transformation des k = 5 -> 2 (toutes les lignes avec 4 types d'arythmies = 1 seule classe arythmie),
- . Rééquilibrage des 2 classes (avec Algo. SMOTE) pour avoir 2 classes : [normal; arythmie] avec poids (%) [0.5;0.5], [34568;34568] lignes,

### 2. Méthode « Spot-Checking Pipe » pour sélectionner 1 classifieur de la gamme Machine Learning:

- . Sur nos 2 datasets Train et Test, tester « à la volée », différentes familles de classifieurs et modèles pour en retenir 1 seul ?
- . Famille : linéaire, non-linéaire et d'ensemble,
- . 54 modèles testés,
- . Sortie : Boxplot Accuracy (moyenne et ecart-type)
- . Critère de sélection du modèle : ranking des 10 meilleurs classifieurs sur le critère du Mean-Accuracy.

### 3. Gridsearch() et CrossValidation sur 2 classifieurs concurrents:

- . 1 classifieur machine learning ExtraTrees (Extremely Randomized Trees = RF avec faible variance) ➔ [+] 1 sélection de Xi (188 -> 45)
- . 1 classifieur deep learning Perceptron Multi-couches (2 layers)

### 4. Comparaison de la performance des 2 classifieurs: métriques [Accuracy ; AUC] + Courbe ROC

### 5. Grille des résultats: 2 itérations de calculs pour n = 4000\* et 34000 lignes [i.e. sous contrainte de calculs in-memory]

\* Échantillonnage aléatoire simple – data centrées-réduites.

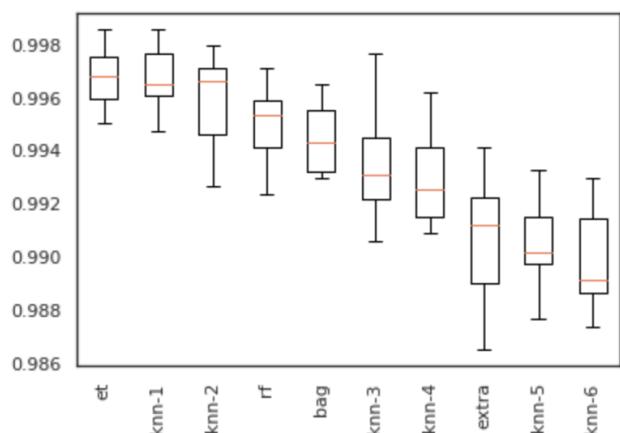
## 1. Classification binaire sur signaux bruts

## 1.3. Résultats: algos / Xi

### Importance Xi (ExtraTrees\*\*)

#### Spot-Checking ranking

Rank=1, Name=et, Score=0.997 (+/- 0.001)  
Rank=2, Name=knn-1, Score=0.997 (+/- 0.001)  
Rank=3, Name=knn-2, Score=0.996 (+/- 0.002)  
Rank=4, Name=rf, Score=0.995 (+/- 0.001)  
Rank=5, Name=bag, Score=0.995 (+/- 0.001)  
Rank=6, Name=knn-3, Score=0.993 (+/- 0.002)  
Rank=7, Name=knn-4, Score=0.993 (+/- 0.002)  
Rank=8, Name=extra, Score=0.991 (+/- 0.002)  
Rank=9, Name=knn-5, Score=0.990 (+/- 0.002)  
Rank=10, Name=knn-6, Score=0.990 (+/- 0.002)  
temps execution en mn = 0.1939408833333308



Dataset : Test rééquilibré, n = 4000

Sérialisation  
des Xi ?  
(effet guttman ?)

(ranking : 1 -> 13)  
pour la série Xi , pour i = [174 -> 186]

Feature ranking:

1. feature 182 (0.039993)
2. feature 184 (0.039029)
3. feature 183 (0.038100)
4. feature 181 (0.036626)
5. feature 185 (0.036579)
6. feature 186 (0.034489)
7. feature 180 (0.031835)
8. feature 176 (0.030716)
9. feature 178 (0.030077)
10. feature 179 (0.028706)
11. feature 175 (0.025555)
12. feature 177 (0.019693)
13. feature 174 (0.016356)

#### Principe d'importance des Variables avec les forêts d'Extra-arbres:

- . Approche Embedded = méthode de sélection de variable intégrée à la construction d'un modèle (i.e. Extra-Arbres),
- . Classer les Xi selon la valeur de leur variance inter Extra-Arbres (selon seuil fixé)
- . Par défaut avec Extra-Trees, valeur de seuil = 0.00

Analyse du classement, valeur du seuillage ? ➔ Expert = **Cardiologue**

Dataset : Train + Test rééquilibrés, n = 8000

## 1. Classification binaire sur signaux bruts

lignes = 4000  
train | test = 0.8 | 0.2  
Laptop

	MLP baseline	MLP advanced
	1 layer 500 epochs	2 layers 500 epochs
	Xi selection (51)	
	CV	
	k = 10	k = 10

Accuracy Value	97,31% (0.72%)	97,26% (+/- 0.59%)
Accuracy Curve		oui
Losses Curve		oui

AUC value	0,994
AUC Curve	oui

ROC Curve	oui
-----------	-----

temps execution calepin (mn)	348,25
	~ 6,00 Heures

lignes = 34000  
train | test = 0.8 | 0.2  
Machine de production

	MLP baseline	MLP advanced
	1 layer 500 epochs	2 layers 500 epochs
	Xi selection (41)	
	CV	
	k = 10	

Accuracy Value	97,76% (0.19%)	98,60% (+/- 0.17%)
Accuracy Curve		oui
Losses Curve		oui

AUC value	0,998
AUC Curve	oui

ROC Curve	oui
-----------	-----

temps execution calepin (mn)	1431
	~ 23,85 Heures

## 1.3. Résultats: Métriques AUC et Accuracy

ERT	
Xi (188)	Xi selection (51)

ERT	
Xi selection (51)	
CV	

ERT	
Xi selection (51)	Xi selection (51)
Gridsearch Parameter = MaxDepth [24]	GridsearchCV Parameter = all k = 10 best parameters

0,42

0,002

0,79

0,002

1,71  
0,004

Pour l'itération n = 34 000 lignes

**Q1 = sur les critères AUC et Accuracy, quel est le meilleur classifieur?**

- sur AUC, ExtraTrees est le meilleur classifieur avec un gain = 0,002 % par rapport a l'AUC de MLP advanced
- sur Accuracy, ExtraTrees est le meilleur classifieur avec un gain = 0,79 % par rapport a l'Accuracy de MLP advanced,

\*\*\*\*

Entre n = 4000 vs n = 34000

**Q2 = quels sont les phénomènes observés quand on augmente la taille des datasets train / test ?**

[entre les meilleures performances de chaque classifieur de même famille]

- sur l'Accuracy, ExtraTrees augmente de 1,71 %
- sur l'Accuracy, MLP advanced augmente 1,34 %
- sur l'AUC, ExtraTrees augmente de 0,004 %
- sur l'AUC, MLP advanced augmente 0,004 %

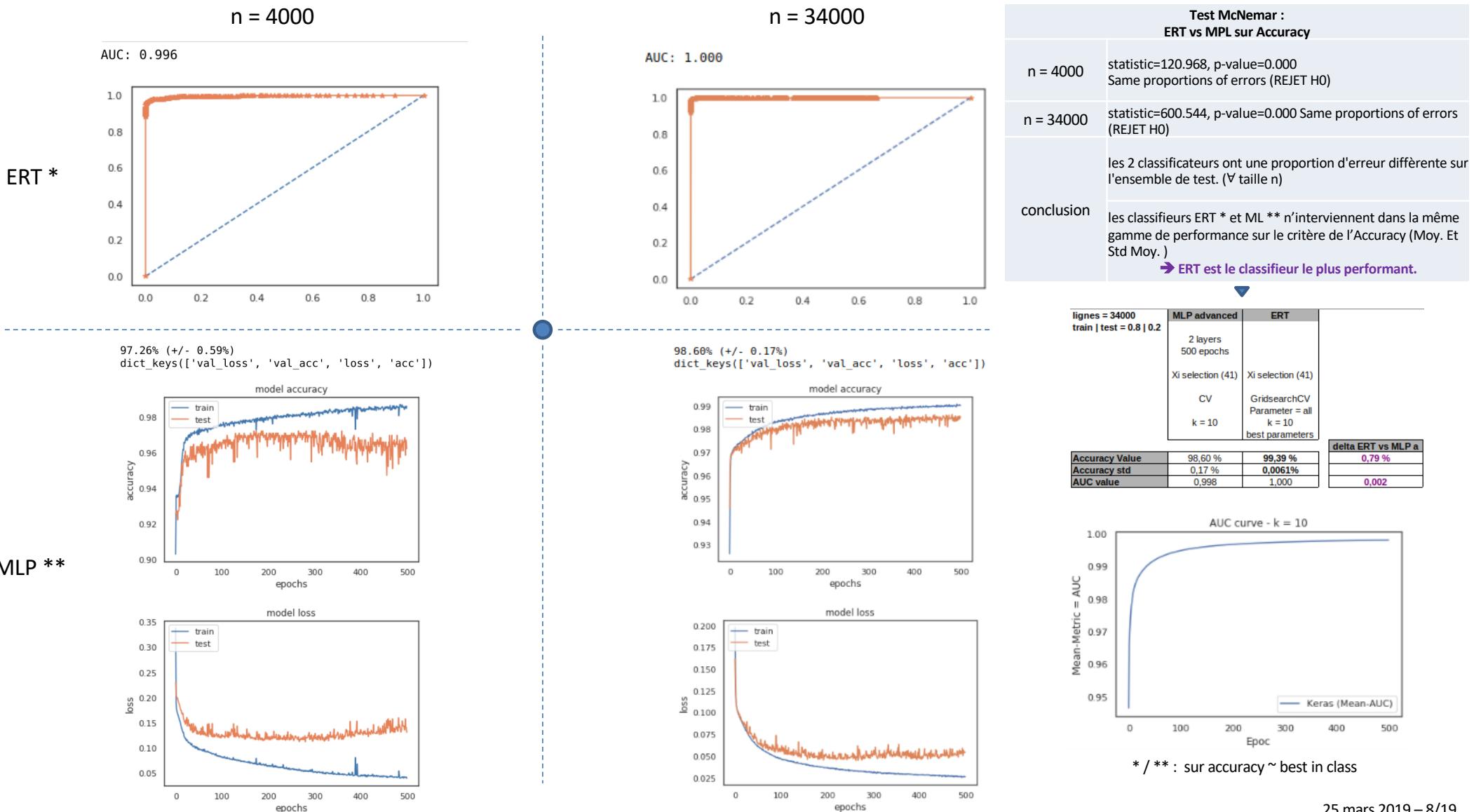
\*\*\*\*

**Q3 = pourquoi ERT est-il le meilleur classifieur sur ces signaux bruts ?**

- les Arbres de décision extrêmement aléatoires semblent bien fonctionner sur les données biomédicales : « *Leur polyvalence et leur interprétation intuitive les rendent particulièrement adaptés aux applications biomédicales* » [ argumentation à vérifier !]  
(<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3399093/>)

**Q4 = pourquoi ERT avec sélection de Xi est meilleur que ERT avec tous les Xi ?**

- ERT discrimine mieux avec les Xi les plus importants (résultat empirique partagé avec RF)

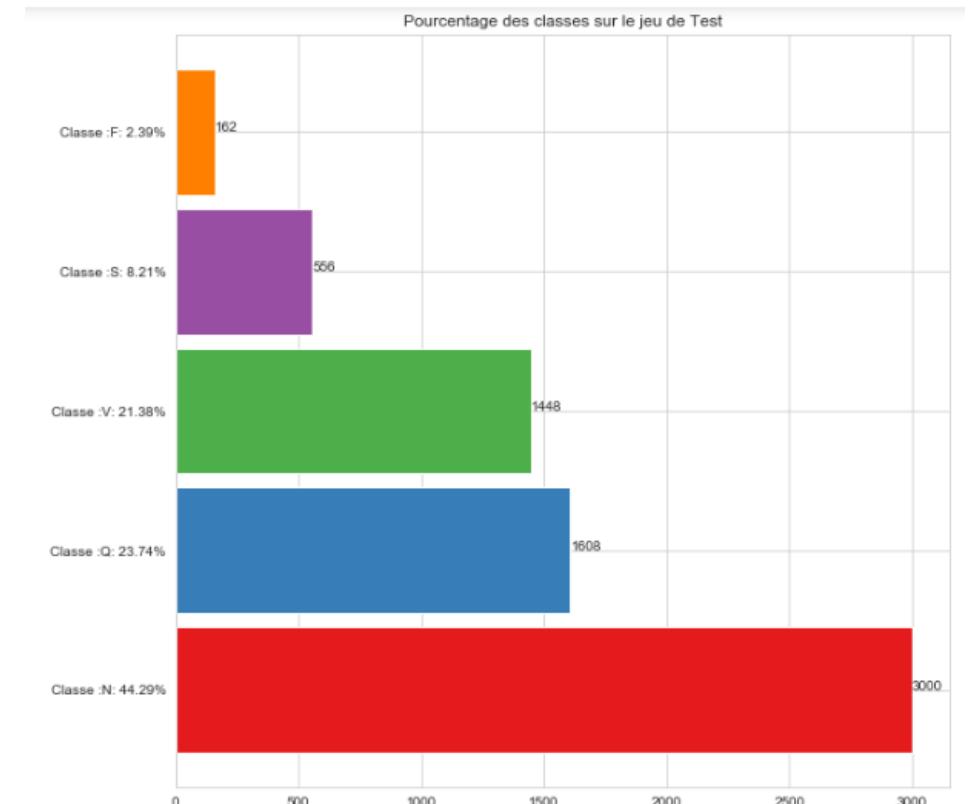
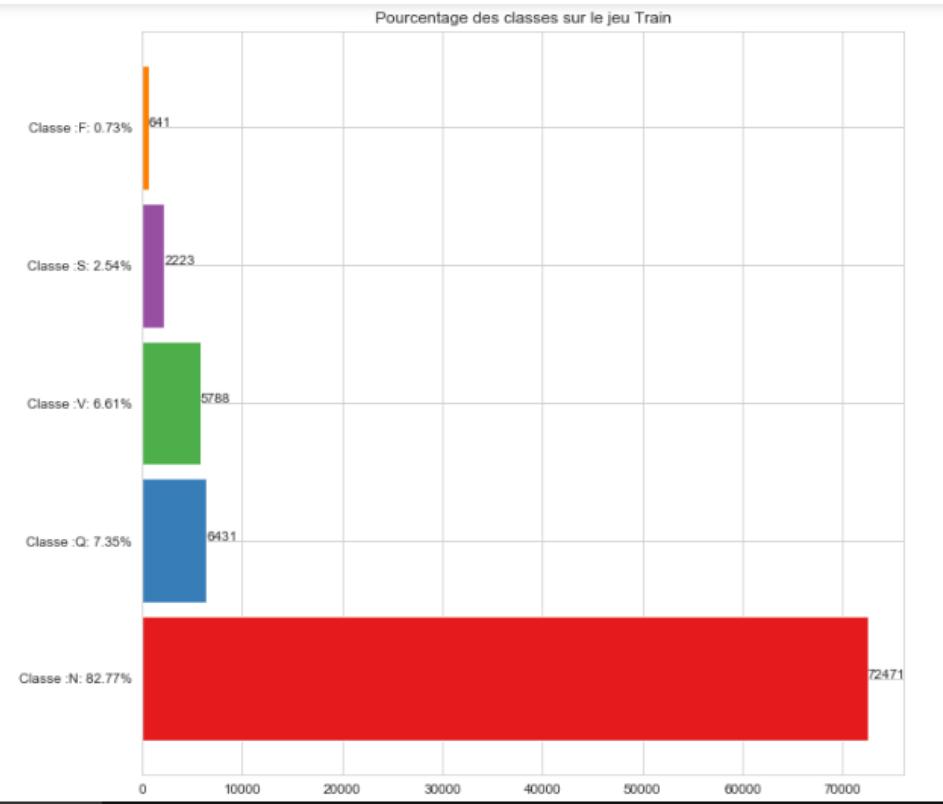


## 2. Classification multi-classes sur signaux bruts

### 2.1. Problème

Problème: classification supervisée multi-classes ( $k=5$ ) avec classes déséquilibrées (%) sur le Train [N.82.7; Q.7.35; V.6.61 ; S.2.54; F.0.73 ] et le Test [N.44.29; Q.23.74; V. 21.38 ; S.8.21; F.2.39]

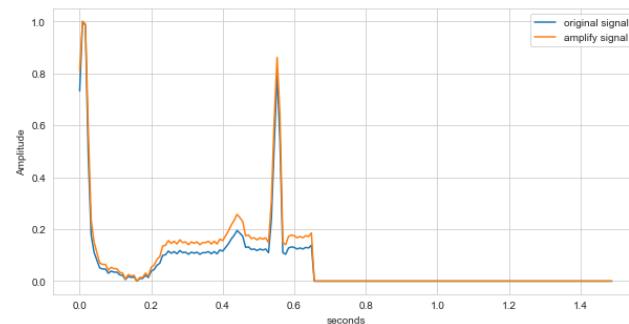
*signalétique: N = rythme Normal ; Q-V-S-F = arythmie (problème de rythme cardiaque)*



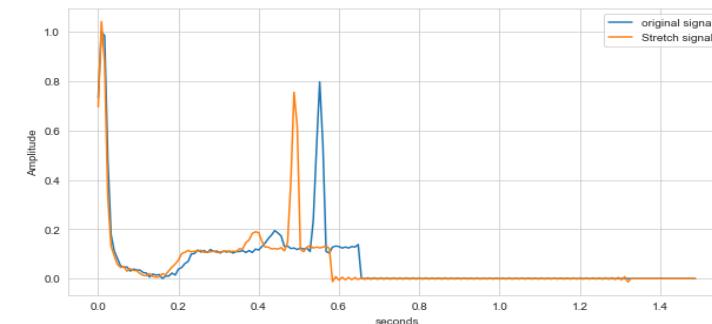
Pour corriger cela on peut procéder à:

- de l'oversampling, en dupliquant certaines séries.
- de l'augmentation de données, en créant de nouveaux signaux par déformation des signaux existants. (**retenue en ajoutant de l'aléatoire**)

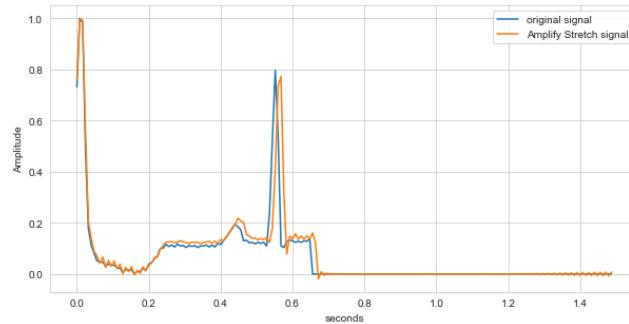
**Signal amplifié**



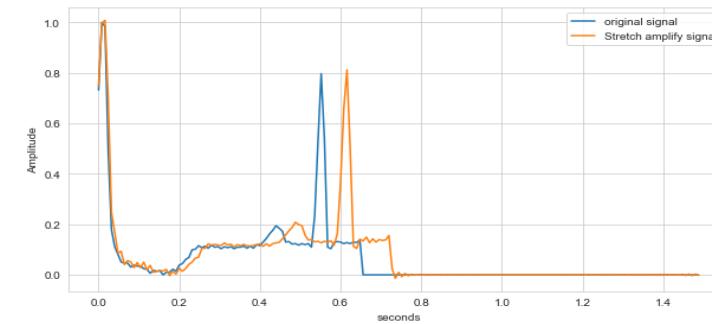
**Signal streché**



**Signal streché puis amplifié**



**Signal amplifié puis streché**

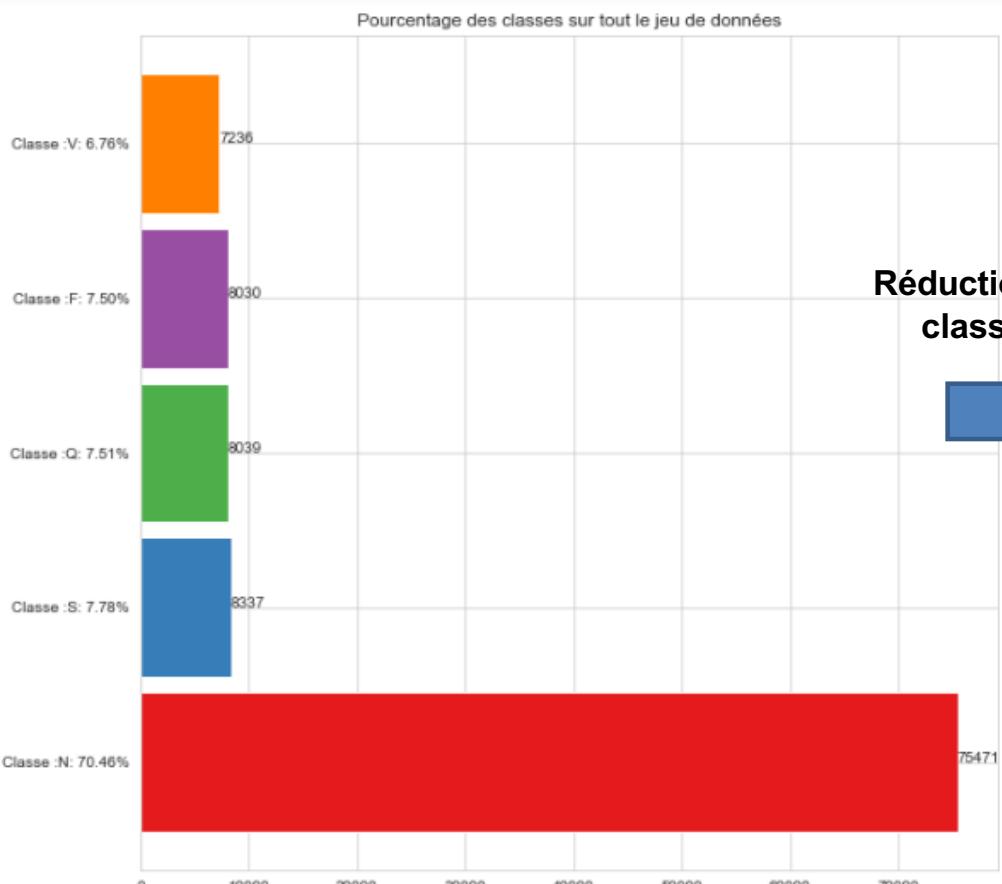


## 2. Classification multi-classes sur signaux bruts

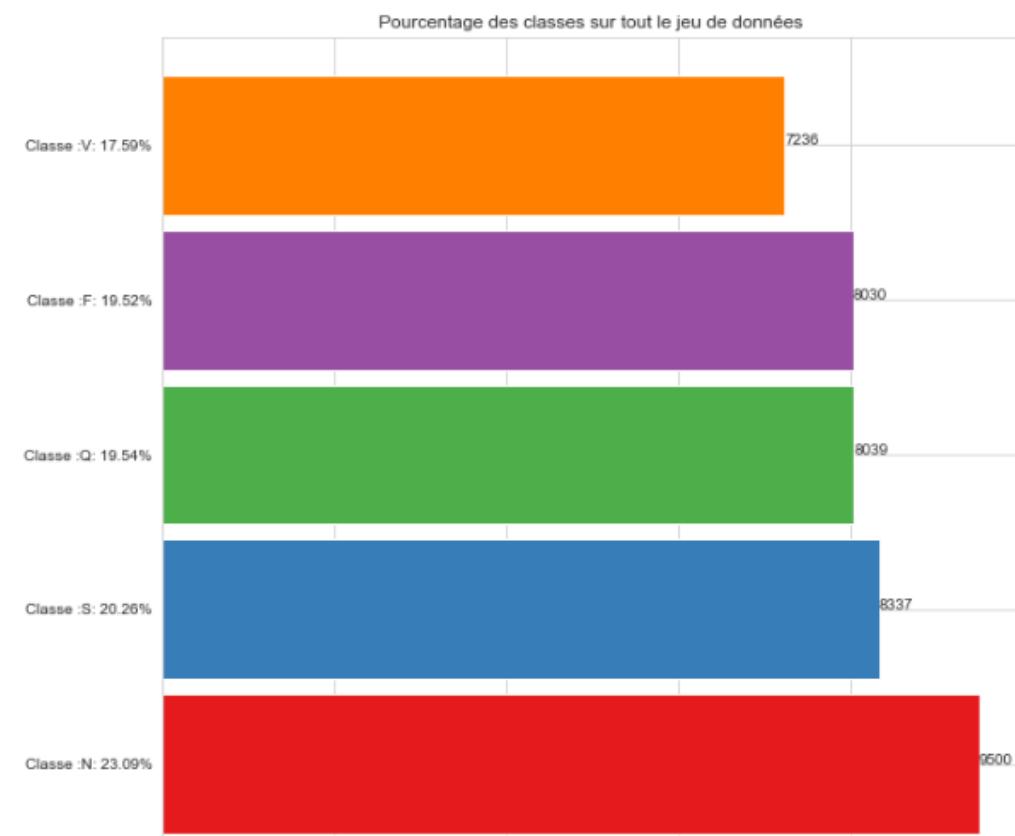
## 2.3. Résultat de la solution

Après application de la solution sur tout le jeu de données et principalement sur les **4** classes sous représentées :

La classe 1 a été strechée puis amplifiée.  
La classe 2 a été amplifiée.  
La classe 3 a été strechée.  
La classe 4 a été amplifiée.



Réduction de la  
classe 'N'  
→



En total, On a appliqué :

- Régression Logistique (78%)
- KNN
- SVM (Linéaire et Gaussien)
- Arbre de décision
- Forêt Aléatoire
- Bagging
- Gradient Boosting
- Xgboost
- MLP



En total, On a retenu :

- KNN (n\_neighbors)
- Bagging (n\_estimators)
- Forêt Aléatoire (n\_estimators et max\_depth)

GridSearch CV ( 10 folds)



Résultats Finaux:

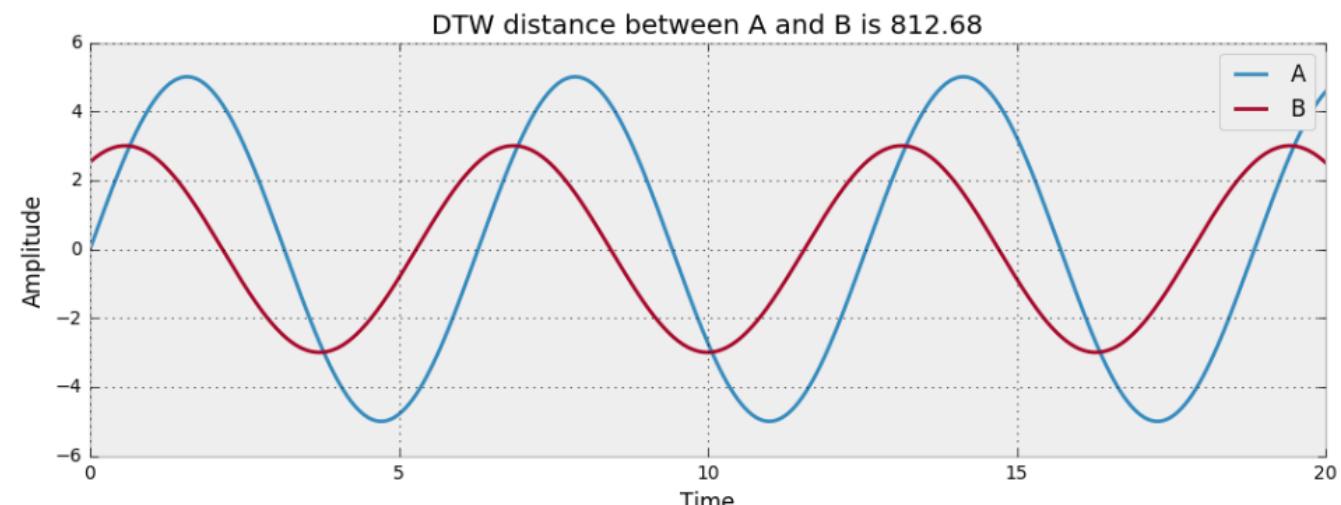
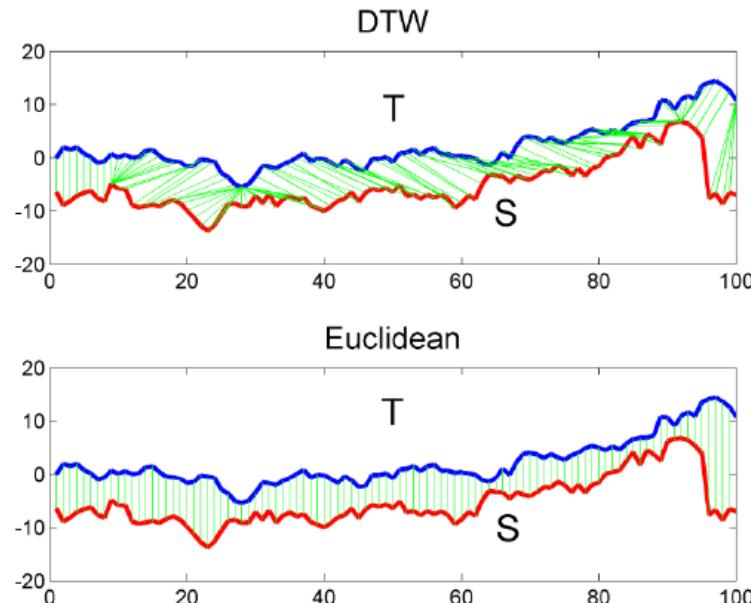
- KNN -> 94.57 % (86.5 sec)
- Bagging -> 94.91 % (**20min**)
- **Forêt Aléatoire -> 95.53 % (86 sec )**

## K Nearest Neighbors & Dynamic Time Warping

« Une méta-analyse réalisée par **Mitsa (2010)** suggère que, lorsqu'il s'agit de la classification temporelle, 1 Voisin le plus proche ( $K = 1$ ) et le Timewarping dynamique sont très difficiles à battre . »

**Question :** Comment peut-on utiliser KNN dans le contexte de données de séries chronologiques?

-> "Dynamic Time Warping" est une technique largement utilisée pour la reconnaissance vocale dans les années 80. L'algorithme DTW trouve l'alignement optimal entre deux séquences d'observations en déformant la dimension temporelle avec certaines contraintes.

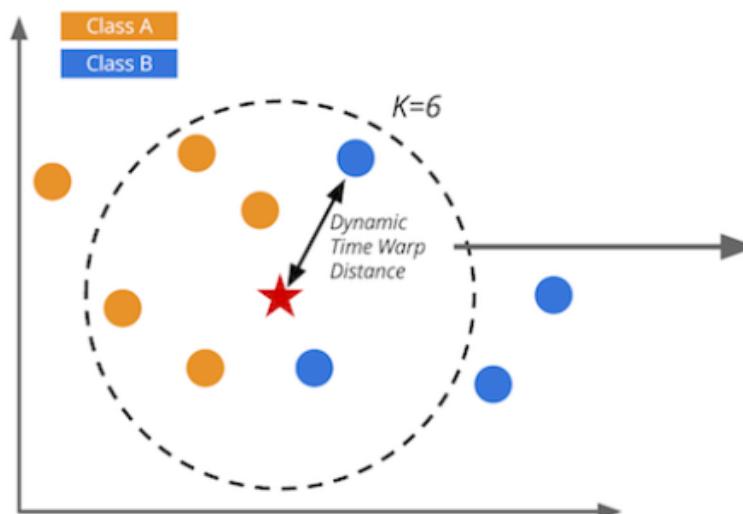


25 mars 2019 – 13/19

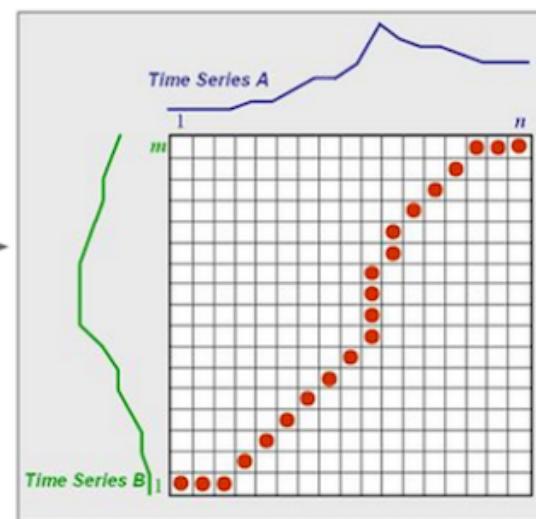
## K Nearest Neighbors & Dynamic Time Warping

« Une mété-analyse réalisée par **Mitsa (2010)** suggère que, lorsqu'il s'agit de la classification temporelle, 1 Voisin le plus proche ( $K = 1$ ) et le Timewarping dynamique sont très difficiles à battre . »

K Nearest Neighbors



Dynamic Time Warping



Réf : [https://nbviewer.jupyter.org/github/markdregan/K-Nearest-Neighbors-with-Dynamic-Time-Warping/blob/master/K\\_Nearest\\_Neighbor\\_Dynamic\\_Time\\_Warping.ipynb](https://nbviewer.jupyter.org/github/markdregan/K-Nearest-Neighbors-with-Dynamic-Time-Warping/blob/master/K_Nearest_Neighbor_Dynamic_Time_Warping.ipynb)

<https://github.com/markdregan/K-Nearest-Neighbors-with-Dynamic-Time-Warping>



Appliqu   H   dataset

## 3. Annexes 1.

## Technologies &gt; Stack de calcul [Didier]

<b>HARDWARE 1</b>		<b>HARDWARE 2</b>	
Machine pour monter le code (n = 4000)		Machine production du code (n = 34000)	
<b>Processeur</b>	Intel Core i7-8750 H@ 2.20GHz × 12	<b>Processeur</b>	Intel Core i7-7700K @ 4,2GHz × 4
<b>Carte graphique</b>	NVIDIA GeForce GTX 1060 Max-Q 6Go mémoire	<b>Carte graphique</b>	NVIDIA GeForce GTX 1080 - 8Go mémoire
<b>Mémoire</b>	2×8 Go, DDR4, DDR4-2400 (jusqu'à 32 Go)	<b>Mémoire</b>	2×8 Go, DDR4, DDR4-2400 (jusqu'à 32 Go)
<b>Stockage</b>	SSD 512Go M.2 + HDD 1 To @ 5400 RPM)	<b>Stockage</b>	SSD 512Go M.2 + HDD 2 To @ 7200 RPM)
<b>SOFTWARE</b>		<b>SOFTWARE</b>	
<b>Ubuntu</b>	Lts 18.04 Bionic Beaver	<b>Ubuntu</b>	Lts 18.04 Bionic Beaver
<b>Anaconda</b>	1.9.6	<b>Anaconda</b>	1.9.6
<b>Jupyter Notebook</b>	5.7.4	<b>Jupyter Notebook</b>	5.7.4
<b>Nvidia</b>	NVIDIA-SMI 396.54	<b>Nvidia</b>	NVIDIA-SMI 396.54
	cudatoolkit 9.2		cudatoolkit 9.2
	cudnn 7.2.1		cudnn 7.2.1
<b>Python</b>	version 3.6	<b>Python</b>	version 3.6
<b>Keras</b>	version 2.2.4	<b>Keras</b>	version 2.2.4
<b>Tensorflow-GPU</b>	version 1.12.0	<b>Tensorflow-GPU</b>	version 1.12.0

## Classifieur d'ensemble : famille d'arbres de décision extrêmement aléatoires

Dans ce travail, nous utilisons des arbres de décision extrêmement aléatoires (extra-arbres) [ 7 ]. Il s'agit d'une méthode de classificateur d'ensemble qui étend les arbres de décision classiques en introduisant un caractère aléatoire au cours du processus de construction. Pendant l'apprentissage, un ensemble d'arbres randomisés est construit de manière supervisée et utilisé ultérieurement pour prédire la sortie de vecteurs d'entrée jamais vus auparavant. Pendant l'entraînement, plusieurs arbres sont construits à partir de l'ensemble de données  $\{X, Y\}$  des  $s$  exemples d'entrée de dimension  $x_i \in \mathbb{R}^s$  et leur sortie correspondante  $y_i \in \{0, 1\}$ . La procédure de construction d'un extra-arbre fonctionne de manière récursive en scindant successivement les noeuds dont la sortie varie, comme décrit dans l'[algorithme 2](#). A chaque noeud successif, les seuils candidats  $\{\lambda_1, \dots, \lambda_s\}$  sont générés au hasard (dans la plage des données) pour chacune des  $s$  attributs d'entrée de  $X$ . Chaque seuil candidat  $\lambda$  est utilisé pour scinder les données  $\{X, Y\}$  en deux sous-ensembles  $\{X_l, Y_l\}, \{X_r, Y_r\}$ , puis il est associé à un score  $\psi_\lambda$  ([Eq.\(4\)](#)) sur la base de la réduction de la variance relative,

$$\psi_\lambda = \frac{\text{var}(Y) - \frac{|Y_l|}{|X|}\text{var}(Y_l) - \frac{|Y_r|}{|X|}\text{var}(Y_r)}{\text{var}(Y)}, \quad (4)$$

où  $|X|$  indique la longueur du vecteur  $X$  et  $\text{var}(Y)$  la variance de la sortie disponible au noeud actuel. Le seuil avec le score le plus élevé est utilisé pour annoter le noeud courant et les deux sous-ensembles résultants  $\{X_l, Y_l\}, \{X_r, Y_r\}$  sont utilisées comme données d'entrée pour la construction de la gauche  $t_l$  et le droit  $t_r$  subtrees, respectivement. Le processus de construction continue pour tous les noeuds ayant au moins  $n_{\min} = 5$  échantillons (comme suggéré dans [ 7 ]) et où le rendement varie selon les échantillons d'entraînement. Pendant la prédiction, le vecteur d'entrée  $x_i$  est traité par tous les arbres. Les sorties des arbres sont agrégées et la moyenne donne la prédiction finale  $\hat{y}_i$ .

<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3399093/>

`t = buildExtraTree (X, Y)`

---

```

si |X| < n_min OU isConstant (Y) puis
    Renvoie buildLeaf (moyenne (Y))
fin si
// Générer des seuils de candidats au hasard
{λ_1, ..., λ_n} ← GenerateSplits (min (X(j)), max (X(j))), j ∈ {1, ..., s}
// Sélection du seuil avec le meilleur score (Eq.\(4\))
λ_j ← arg max_{λ_j} ψ_{λ_j} (X)
// Split Data, Recurse et Build node
{X_l, Y_l, X_r, Y_r} ← λ_j (X, Y)
t_l ← buildExtraTree (X_l, Y_l)
t_r ← buildExtraTree (X_r, Y_r)
Renvoie le buildNode (λ_j, j, t_l, t_r)

```

---

. ExtraTrees = un méta-estimateur qui adapte un certain nombre d'arbres de décision aléatoires (ou extra-arbres) à divers sous-échantillons du jeu de données et utilise la moyenne pour améliorer la précision prédictive et contrôler le surajustement.

. ExtraTrees = Random Forest mais avec FAIBLE VARIANCE (mais biais ↗)

. Dans ExtraTrees, le caractère aléatoire ne provient pas de l'initialisation de données mais de la division aléatoire de toutes les observations.

## spot-check-machine-learning-algorithms-in-python ?

. Les algorithmes de vérification ponctuelle sont une technique d'apprentissage automatique appliquée conçue **pour fournir rapidement et objectivement un premier ensemble de résultats sur un nouveau problème de modélisation prédictive.**

. Contrairement à la recherche sur grille et à d'autres types de réglage d'algorithmes qui recherchent l'algorithme optimal ou la configuration optimale d'un algorithme, la **vérification ponctuelle est destinée à évaluer rapidement un ensemble varié d'algorithmes et à fournir un résultat approximatif de première coupe.** Ce premier résultat de coupe peut être utilisé pour se faire une idée si un problème ou une représentation du problème est effectivement prévisible et, le cas échéant, les types d'algorithmes qui mériteraient d'être étudiés plus en détail.

<https://machinelearningmastery.com/spot-check-machine-learning-algorithms-in-python/>

### algorithmes de classification mobilisés

```

1 # create a dict of standard models to evaluate {name:object}
2 def define_models(models=dict()):
3     # linear models
4     models['logistic'] = LogisticRegression()
5     alpha = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
6     for a in alpha:
7         models['ridge-'+str(a)] = RidgeClassifier(alpha=a)
8     models['sgd'] = SGDClassifier(max_iter=1000, tol=1e-3)
9     models['pa'] = PassiveAggressiveClassifier(max_iter=1000, tol=1e-3)
10    # non-linear models
11    n_neighbors = range(1, 21)
12    for k in n_neighbors:
13        models['knn-'+str(k)] = KNeighborsClassifier(n_neighbors=k)
14    models['cart'] = DecisionTreeClassifier()
15    models['extra'] = ExtraTreeClassifier()
16    models['svm'] = SVC(kernel='linear')
17    models['svmp'] = SVC(kernel='poly')
18    c_values = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
19    for c in c_values:
20        models['svmr'+str(c)] = SVC(C=c)
21    models['bayes'] = GaussianNB()
22    # ensemble models
23    n_trees = 100
24    models['ada'] = AdaBoostClassifier(n_estimators=n_trees)
25    models['bag'] = BaggingClassifier(n_estimators=n_trees)
26    models['rf'] = RandomForestClassifier(n_estimators=n_trees)
27    models['et'] = ExtraTreesClassifier(n_estimators=n_trees)
28    models['gbm'] = GradientBoostingClassifier(n_estimators=n_trees)
29    print('Defined %d models' % len(models))
30    return models

```

### Les étapes clefs du « pipe code Python » :

- Préparation : centrage-réduction, normalisation,
- Tirage de 1000 échantillons,
- Validation croisée, k = 10,
- Choix d'une Métrique accuracy (moy. + ecart-type),
- Visualisation avec classement de boxplot,

### 3. Annexes 4.

### GridsearchCV [Didier]

Recherche en grille  
Classifieur ERT  
Liste des paramètres

```
{'bootstrap': False,
'class_weight': None,
'criterion': 'gini',
'max_depth': None,
'max_features': 'auto',
'max_leaf_nodes': None,
'min_impurity_decrease': 0.0,
'min_impurity_split': None,
'min_samples_leaf': 1,
'min_samples_split': 2,
'min_weight_fraction_leaf': 0.0,
'n_estimators': 'warn',
'n_jobs': None,
'oob_score': False,
'random_state': 42,
'verbose': 0,
'warm_start': False}
```

#### (1) Recherche des meilleurs paramètres par GridSearch - sur 1 paramètre = max\_depth

```
param=[{"max_depth":list(range(1,25))}]
tree= GridSearchCV(ExtraTreesClassifier(n_estimators = 1000),param, cv=10, n_jobs=-1)
```

```
Spent Time: 2125.46
Learning Time: 19.21
Accuracy (%): Score = 98.000000 | Param. Opti = {'max_depth': 24}
```

#### (2) Recherche des meilleurs paramètres par GridSearchCV **LARGE** - sur paramètres actionnables

```
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 500, num = 5)]

# criterion = The function to measure the quality of a split
criterion = ['gini', 'entropy']

# Number of features to consider at every split
max_features = ['auto', 'sqrt']

# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(1, 50, num = 1)]
max_depth.append(None)

# Minimum number of samples required to split a node
min_samples_split = [2,3,4,5]

# Minimum number of samples required at each leaf node
min_samples_leaf = [1,2,3,4,5]

# Method of selecting samples for training each tree
bootstrap = [False, True]

# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
```

```
] : ERT_random.best_params_
{'n_estimators': 100,
'min_samples_split': 2,
'min_samples_leaf': 1,
'max_features': 'auto',
'max_depth': None,
'bootstrap': False}
```

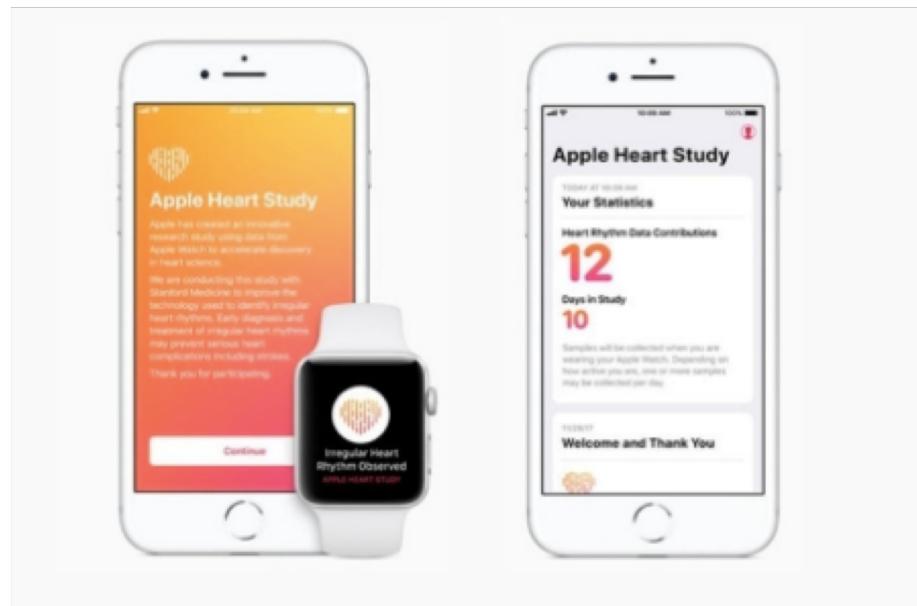
```
Model Performance
Average Error: 0.0063 degrees.
Accuracy = 99.37%.
```

#### (3) Recherche des meilleurs paramètres par GridSearchCV **RESSERRE** - sur paramètres actionnables

```
# Create the parameter grid based on the results of random search
param_grid = {
    'n_estimators': [300, 400, 500],
    'min_samples_split': [2, 3, 4, 5],
    'min_samples_leaf': [1, 2, 3],
    'max_features': ['auto'],
    'max_depth': [10, 20, 30, 40],
    'bootstrap': [False]}
```

```
Model Performance
Average Error: 0.0061 degrees.
Accuracy = 99.39%.
```

Les chercheurs de Stanford ont présenté les résultats préliminaires d'une étude virtuelle ayant recruté plus de 400 000 participants [16 mars 2019]



Les principales conclusions de l'étude comprennent:

- Dans l'ensemble, seuls **0,5%** des participants ont reçu des notifications de pouls irréguliers, une constatation importante compte tenu des préoccupations suscitées par une notification excessive.
- Les comparaisons entre la détection de pouls irrégulier sur Apple Watch et les enregistrements de patchs électrocardiographiques simultanés ont montré que l'algorithme de détection de pouls (indiquant une lecture de tachogramme positive) avait une valeur **prédictive positive de 71%**. Quatre-vingt-quatre pour cent du temps, les participants ayant reçu des notifications de pouls irréguliers se trouvaient en fibrillation auriculaire au moment de la notification.
- **Un tiers (34%)** des participants qui ont reçu des notifications de pouls irréguliers et qui ont suivi l'utilisation d'un timbre ECG plus d'une semaine plus tard ont présenté **une fibrillation auriculaire**. Étant donné que la fibrillation auriculaire est une affection intermittente, il n'est pas surprenant qu'elle ne soit pas détectée lors de la surveillance ultérieure du patch ECG.
- Cinquante-sept pour cent des personnes qui ont reçu des notifications de pouls irréguliers ont consulté un médecin.

<http://med.stanford.edu/news/all-news/2019/03/apple-heart-study-demonstrates-ability-of-wearable-technology.html>