Détection automatique de chutes à l'aide de montres connectées

Raphaël Kuhn
n° candidat: 23102

Sommaire

Un exemple: L'Apple Watch

CAPTEURS
Rythme Cardiaque
Oxygène dans le sang
Microphone
Accéléromètre

FONCTIONALITEES SANTE
Détection de la fréquence cardiaque, du bruit, de chute etc...
Prévention de maladies cardiaques
Suivi Sport

ET AUSSI...
Bluetooth, eSim -> Communication

Problématisation et objectifs du TIPE

Applications à la santé des montres connectées → Variées, Domaine très vaste

↓

Recentrer le sujet sur le la détection de chute

Comment détecter efficacement une chute à l'aide d'une montre connectée ?

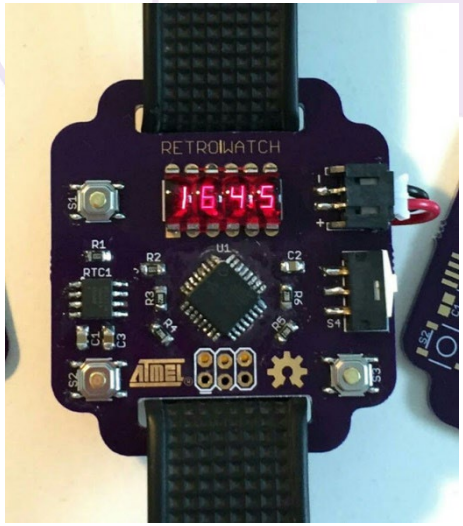
2 axes d'analyse:

- Axe hardware avec la création d'un prototype de montre connectée
- Axe software avec l'utilisation de l'intelligence artificielle pour optimiser la détection de chutes

1. Construction d'un prototype de montre connectée

Premières Pistes

Projets existants:



Retrowatch

By Samsonmarch

Mais:
- Souvent pas d'accéléromètre intégré pour la détection de chutes
- Cher
- Pas de personnalisation possible

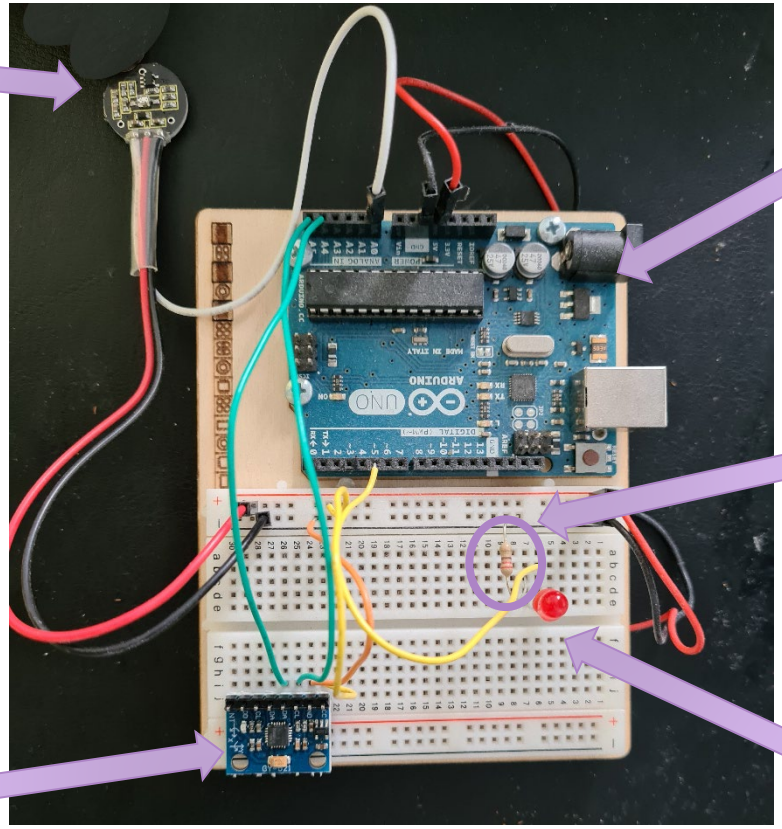
Donc:
- Utilisation Arduino (simple d'utilisation, versatile etc...)

Code source et projet: Retrowatch by Godstale https://github.com/godstale/retrowatch

'I made a smartwatch from scratch' by Samsonmarch https://imgur.com/a/FSBwD3g

Le prototype de montre basé sur l'Arduino UNO

Capteur de fréquence cardiaque



Arduino UNO

Résistance

Diode

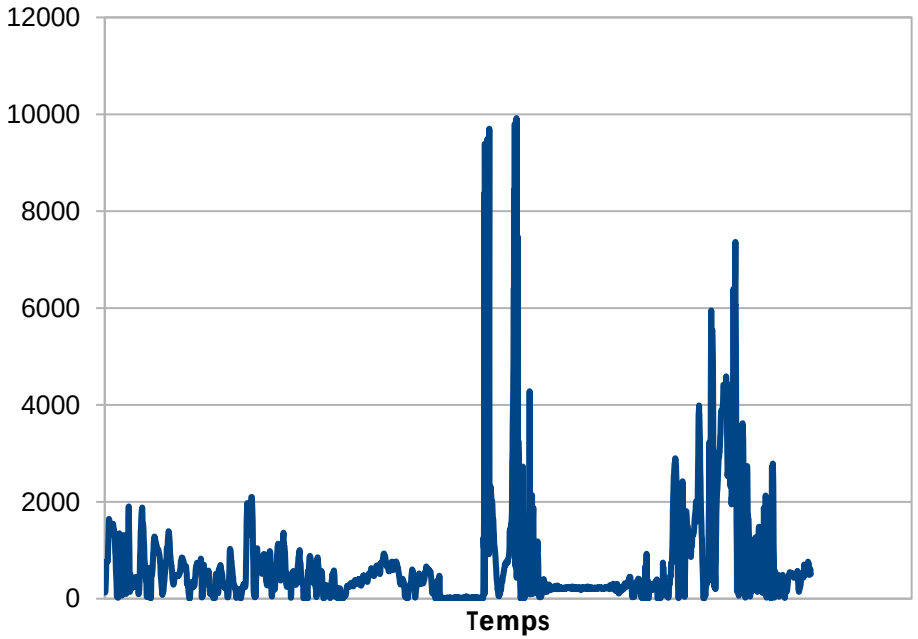
Accéléromètre

Enregistrement de 2 signaux à l'aide du prototype

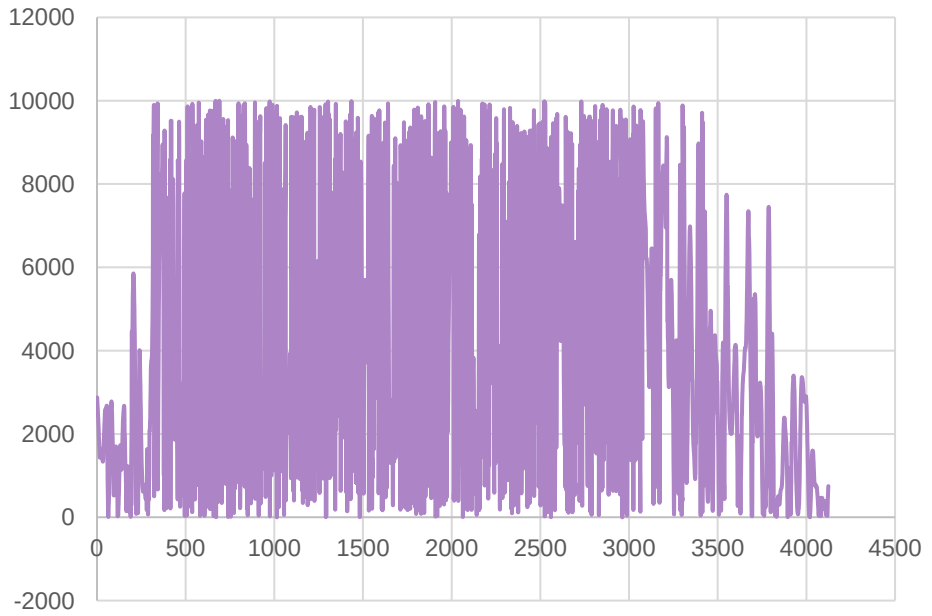
CHUTE

COURSE

Enregistrement de 2 signaux à l'aide du prototype



CHUTE



COURSE

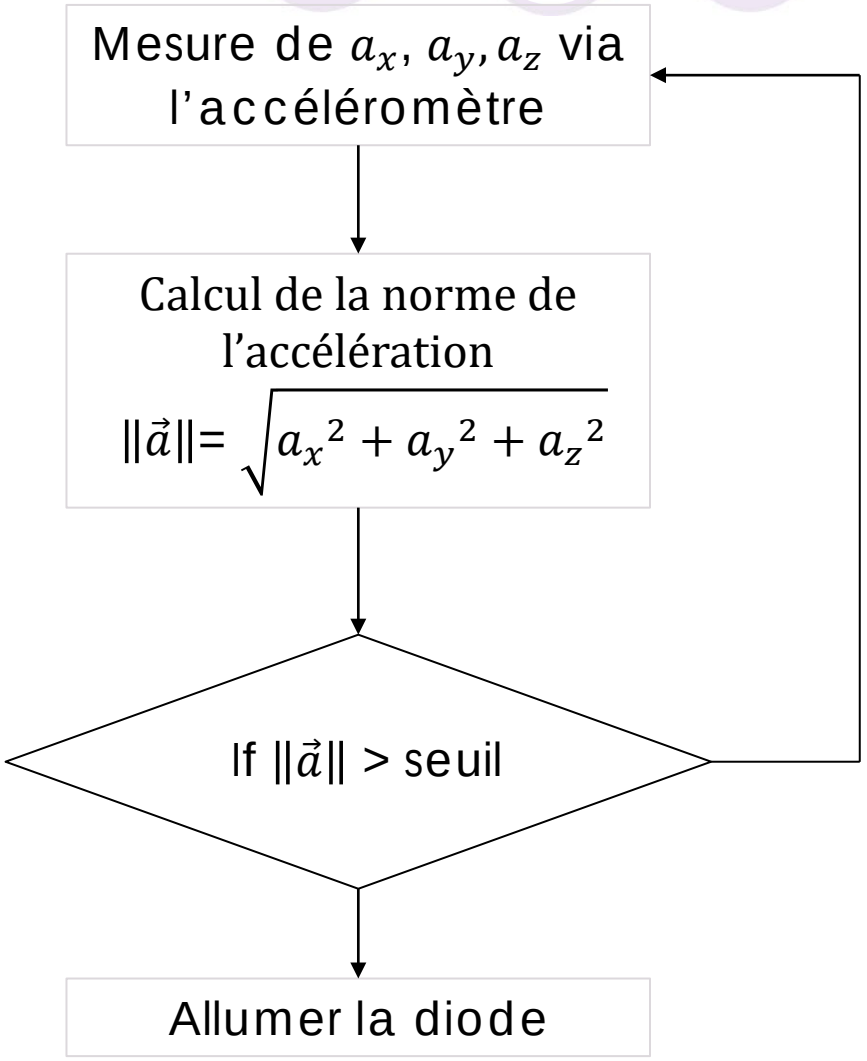
Différence nette à l'œil nu

Idéal pour l'utilisation de l'intelligence artificielle pour différencier les cas

Fonctionnement algorithmique du logiciel embarqué sur le prototype

(Seuil défini arbitrairement)

(On peut rajouter d'autres conditions comme une sur la fréquence cardiaque par exemple)



Les limitations de ce modèle simpliste

- La diode se déclenche dès que l'accélération dépasse un certain seuil, indépendamment du contexte

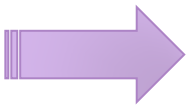
  -> Déclenchement lors d'une course

- Si le seuil de déclenchement est trop bas

  -> Déclenchement lorsque le sujet s'assoit

- Si celui-ci est trop haut

  -> Risque de non-déclenchement

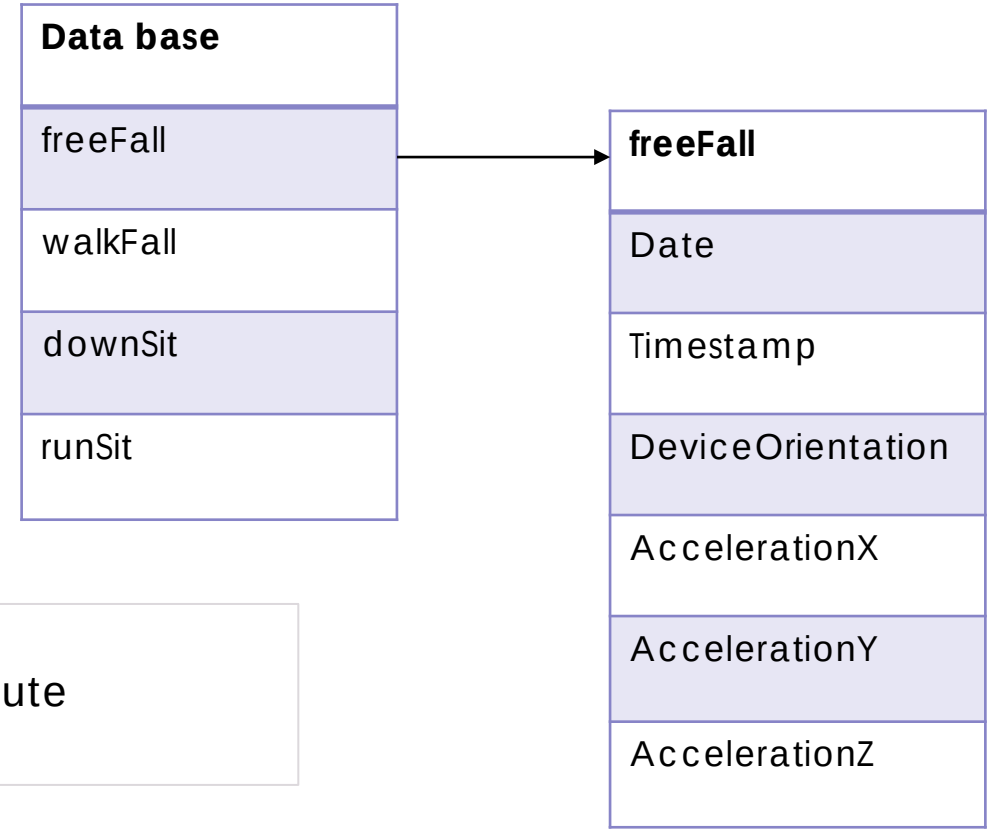


Nécessité d'effectuer un filtrage des signaux correspondants à priori à une chute

2. Affinage des résultats à l'aide de l'intelligence artificielle

Présentation de la base de données composée de différents types de chutes, de courses

Base de données sous licence libre « fall detection accelerometer data » provenant de Kaggle.com*

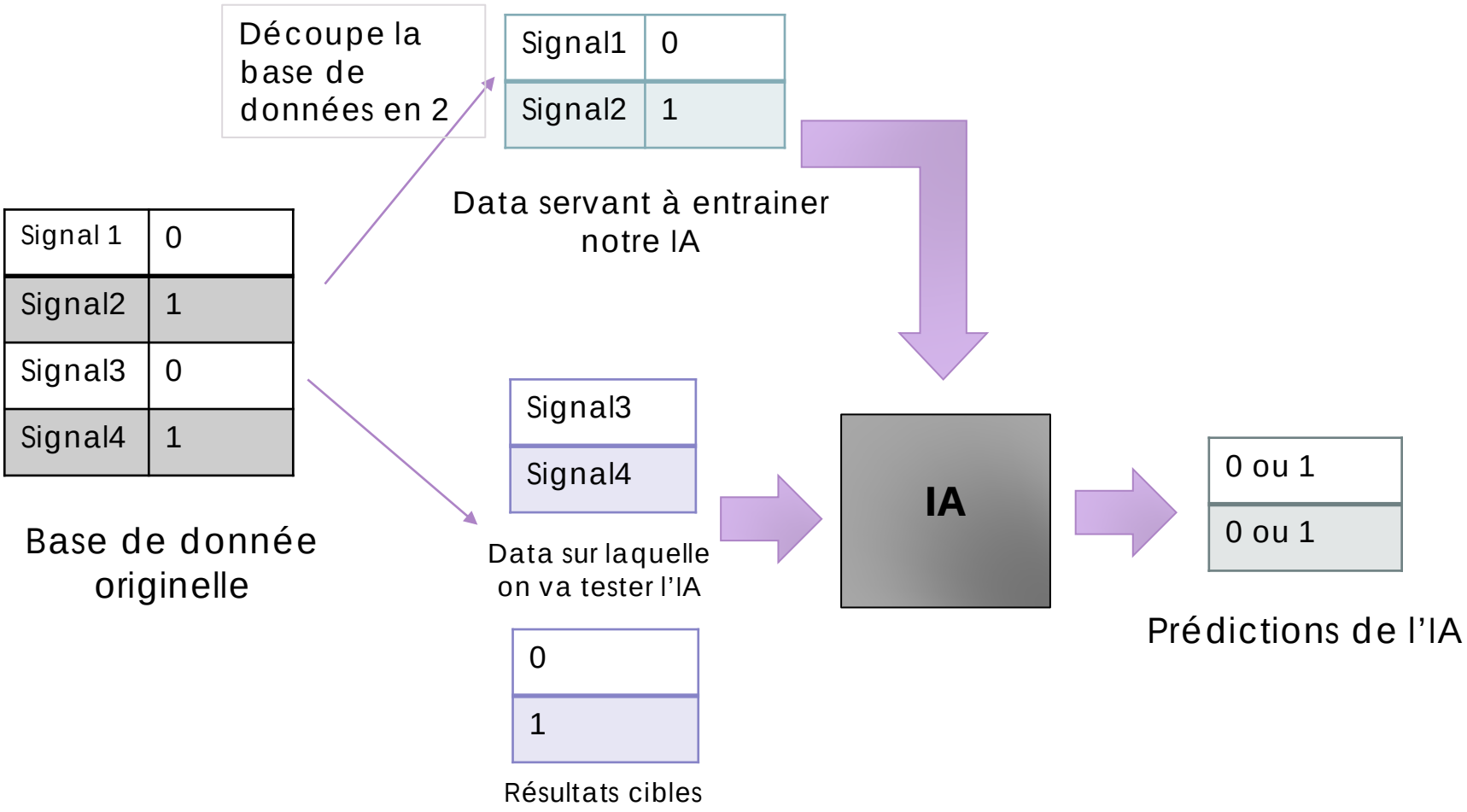


On associe à chaque signal:

- 1 si le signal correspond à une chute
- 0 si ce n'est pas le cas

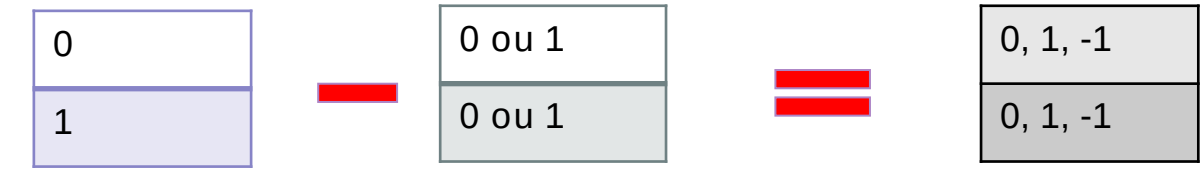
https://www.kaggle.com/datasets/harnoor343/fall-detection-accelerometer-data

Approche méthodologique pour l'apprentissage de l'IA



Approche méthodologique pour l'apprentissage de l'IA

Reste à comparer les prédictions de l'IA avec les résultats cibles

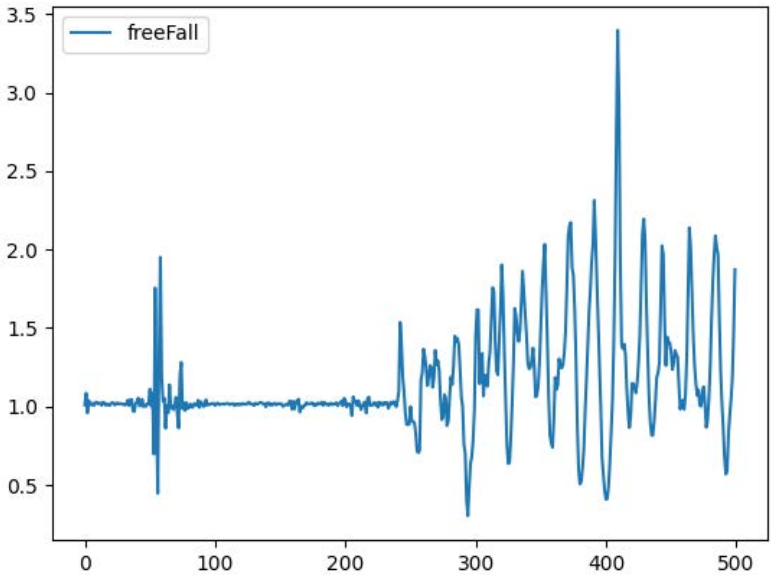


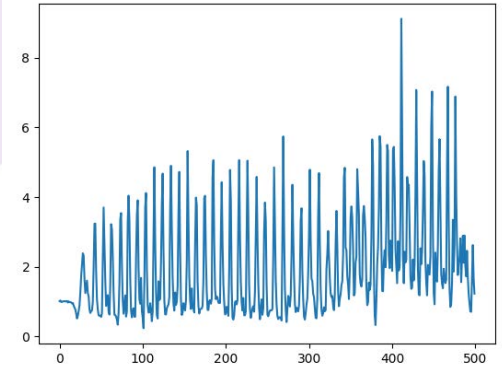
Résultats cibles

Prédictions de l'IA

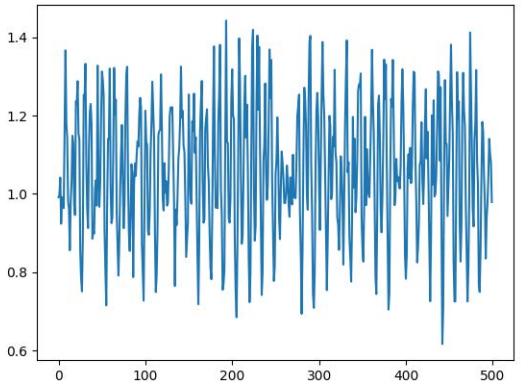
$$Pourcentage\ précison = \frac{nombre\ de\ 0}{taille\ du\ tableau} \times 100$$

Forme des signaux





Run



Walkfall

On réutilise la formule $\|\vec{a}\| = \sqrt{{a_x}^2 + {a_y}^2 + {a_z}^2}$

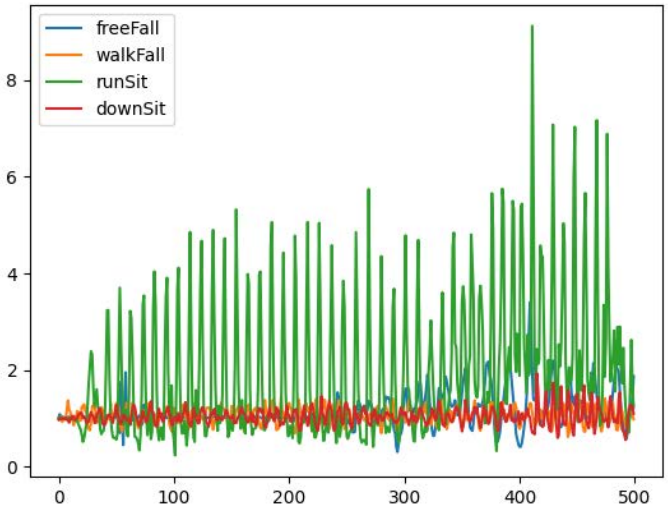
Signal complexe, nécessité de le simplifier

Simplification du signal – Méthode 1

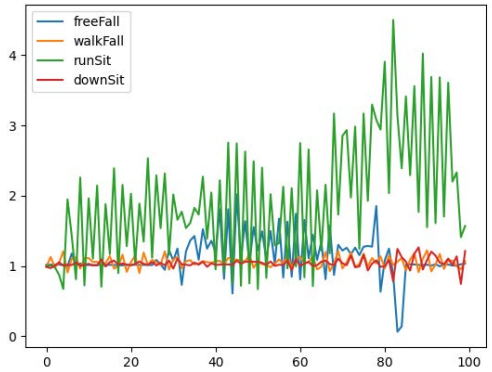
Décomposition du signal en intervalles successifs

$Soit\ n\ \in [1; 500]$

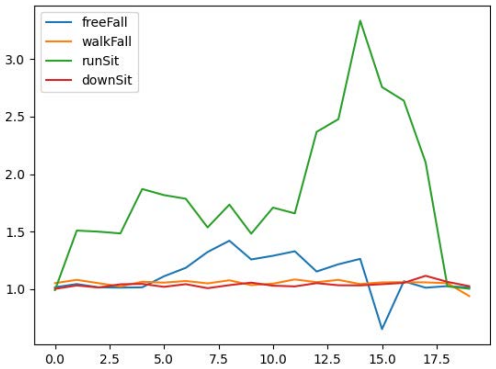
On décompose le signal en n intervalles. On effectue alors la moyenne de l'accélération sur chaque intervalle.
On réduit ainsi le nombre de valeurs à traiter par l'IA de 500 à n



$n = 100$



$n = 20$

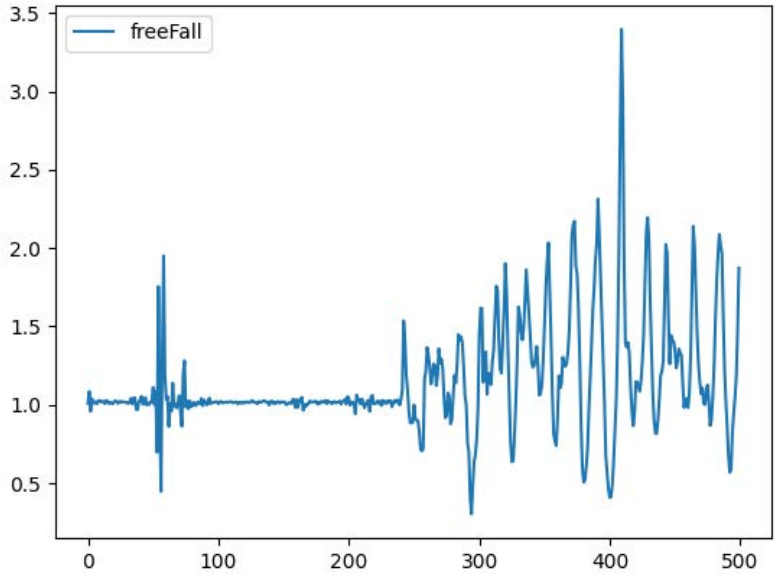


Simplification du signal – Méthode 2

Utilisation de l'algorithme Principal Component Analysis*

$Soit\ n \in [1; 10]$

L'algorithme PCA convertit un signal en un vecteur en n dimensions



$n = 10$

[-0.66071372 1.24201525
3.00892893 -0.18382451
0.77600243 -0.43045682
0.60959799 -0.19094508 -
1.94593178 5.92289126]

$n = 3$

[-0.66071372
1.24201525
3.00892893]

-> Idéal pour modéliser notre base de données

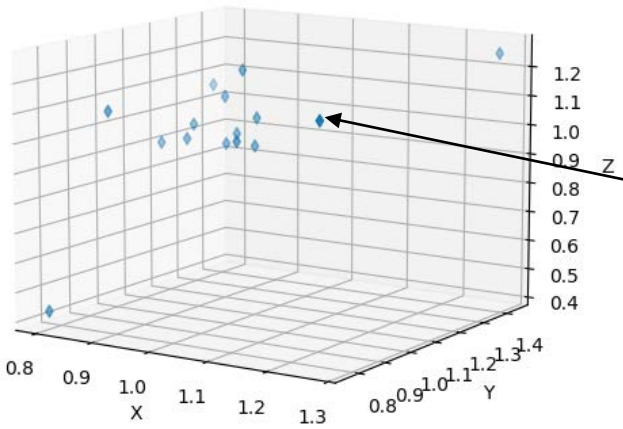
** Principal component analysis (PCA)*
Inclus dans la bibliothèque sklearn sur python

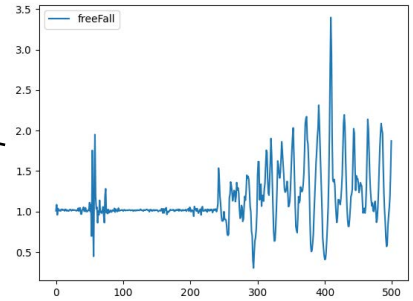
```
from sklearn.decomposition import PCA
```

Modélisation de la base de données en 3 dimensions grâce à l'algorithme PCA

freeFall

Chaque point correspond à un signal de la norme de l'accélération

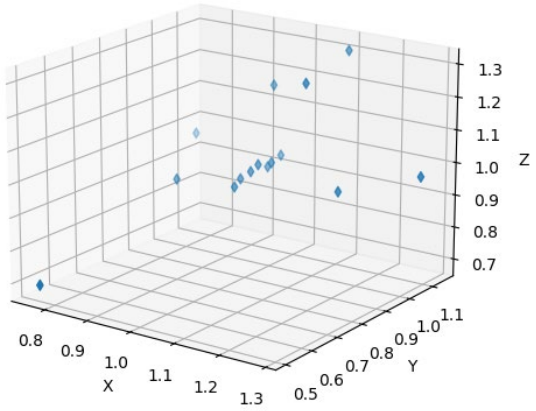


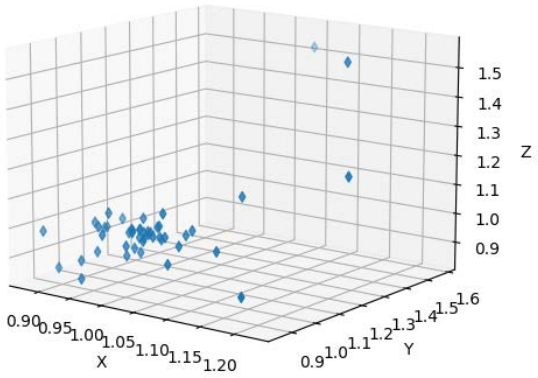


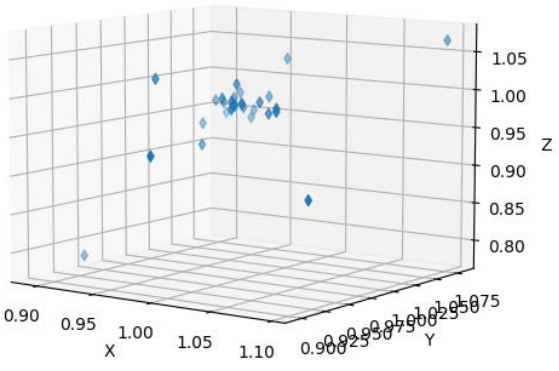
runSit

walkFall

downSit







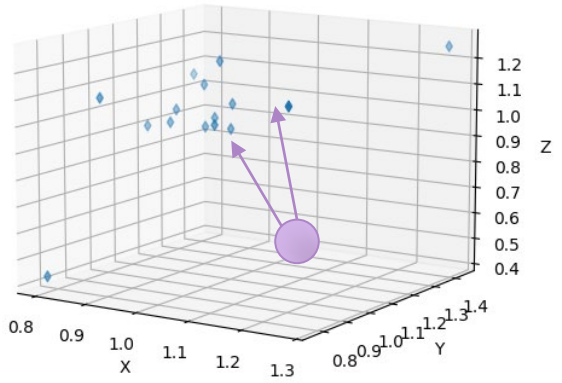
Algorithme basé sur la méthode des voisins

$$methodevoisins(datatrain, datatarget, targettrain, targettest, nbrvoisins)$$

$disteuclidienne(vecteur1, vecteur2)$

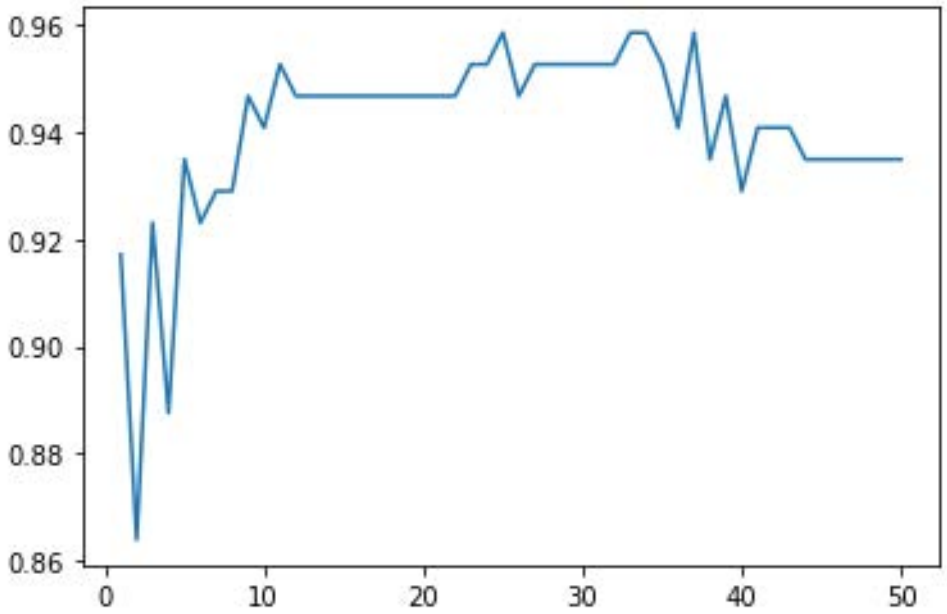
$nvoisins(liste, n)$

Détermine les n points les plus proches du point test, effectue la moyenne des 1 et 0 correspondant à une chute ou non pour chaque voisins. Le résultat est l'arrondi à l'entier cette moyenne.
On a donc déterminer si le point test correspondait à une chute ou non



Résultats

Précision



Nombre de voisins

Maximum pour n = 25, 33, 34, 37
Précision = 95,8%

Axes d'approfondissement

- Utiliser une autre méthode pour l'IA comme celle de Naive-Bayes qui donne de moins bon résultats (81% de précision) mais un temp de calcul plus bas (environ 0,5s)
- Associer le capteur de fréquence cardiaque au système (pour détecter en plus d'une chute potentielle, une possible chute de fréquence cardiaque)
- Intégrer le programme développé en phase 2 à l'Arduino en prenant en compte la puissance réduite de celui-ci
- Miniaturiser le prototype à l'aide de composants plus petits (à l'aide de l'Arduino Pro par exemple)

Annexe

L'algorithme Naive-Bayes

« Naive » -> On suppose que tout les paramètres sont inter-dépendants

Théorème de Bayes $P(H|E) = \frac{P(E|H) * P(H)}{P(E)}$

U : Ensemble des résultats possibles
H : hypothèse de résultat
E : Paramètres à tester
R : Résultat prédis

$$R = \max_{H \in U} P(H|E)$$

$$R = \max_{H \in U} \frac{P(E|H) * P(H)}{P(E)}$$

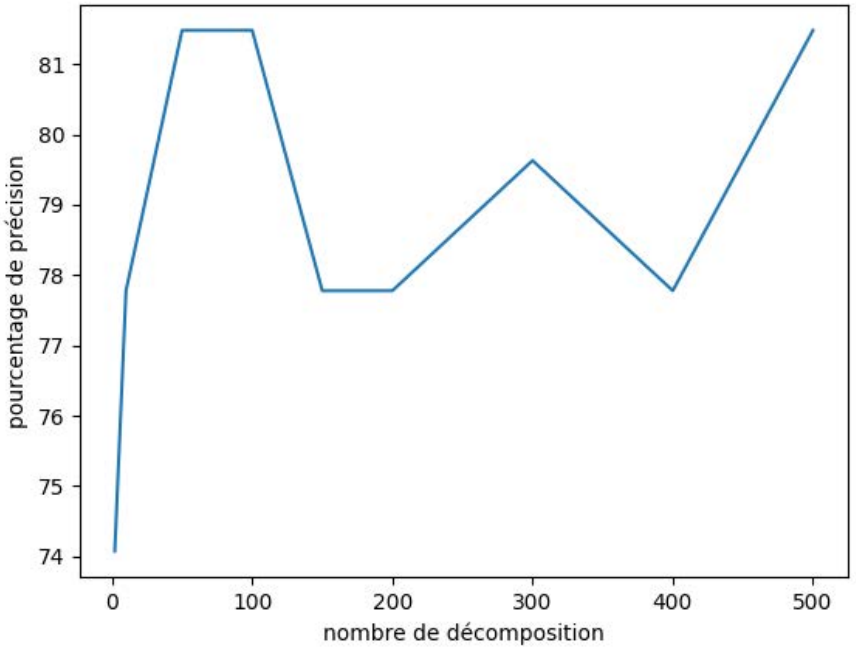
$$R = \max_{H \in U} P(E|H) * P(H)$$

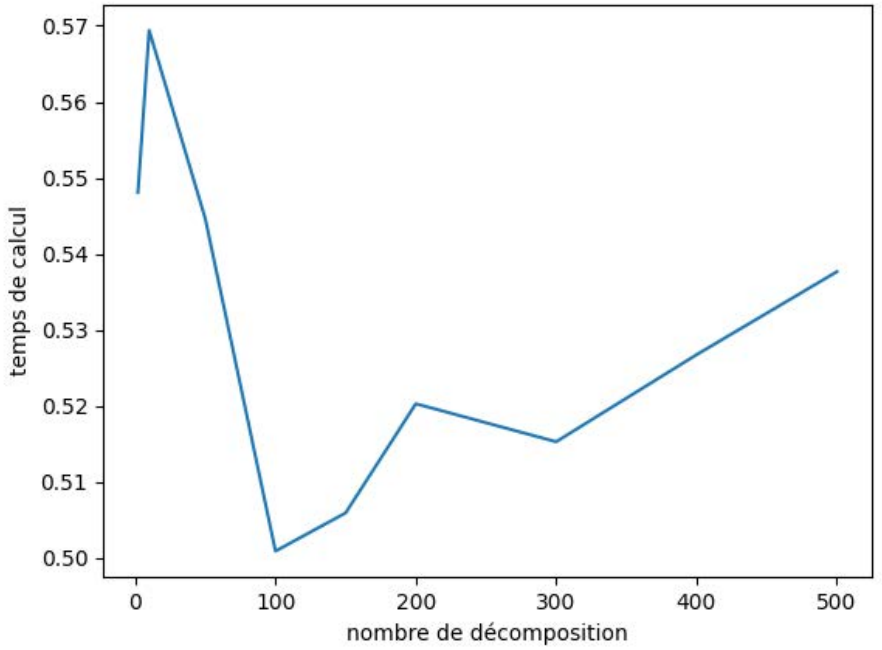
Paramètres à faire varier pour obtenir la meilleure précision de détection de chute

- Dans le cas où on simplifie le signal avec la méthode 1
  - Le nombre de décomposition n
- Avec la méthode 2
  - Le nombre de paramètres des vecteurs (la dimension)

De plus la recherche d'un faible temps de calcul est cruciale étant donné le contexte médical. Ce sera donc un autre résultat à suivre

Naive-Bayes après simplification utilisant la méthode des intervalles

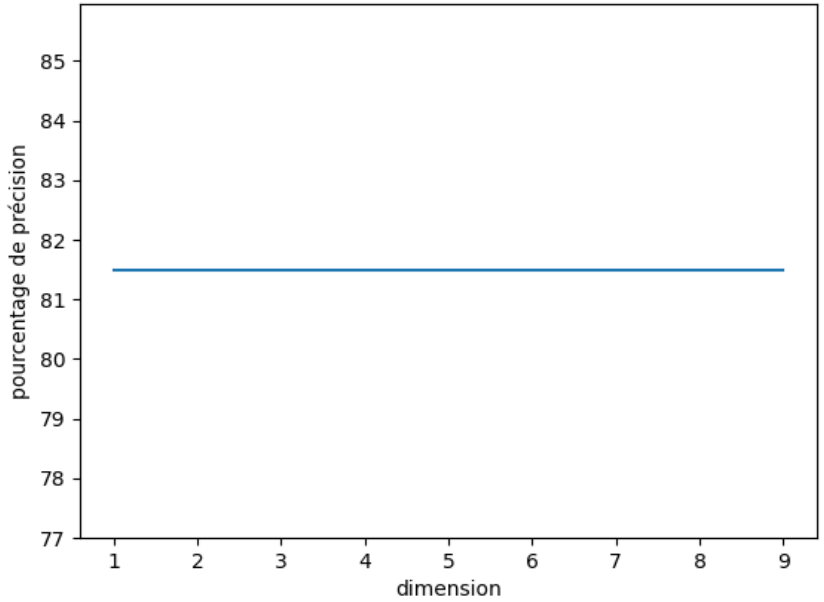


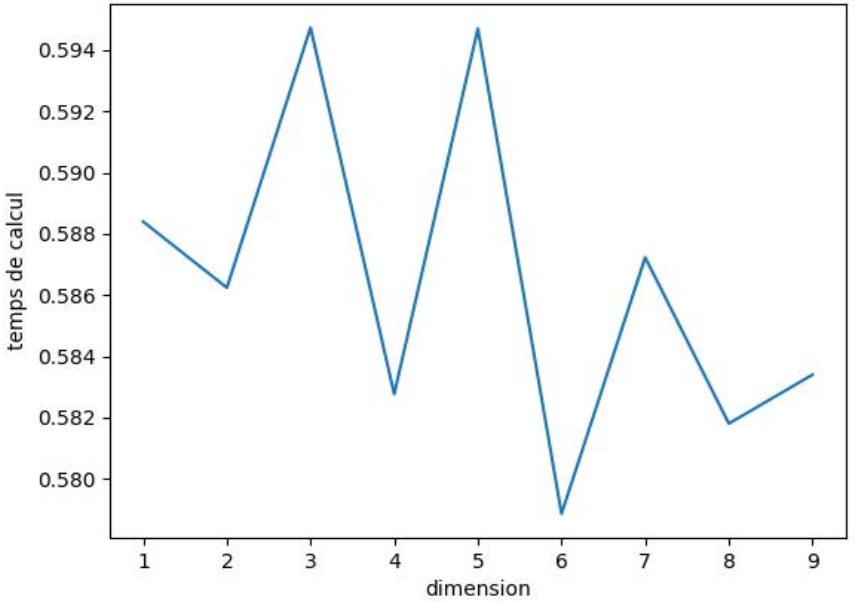


-> Le « nombre de décomposition » correspond aux nombres d'intervalles en lequel on divise le signal

Une décomposition du signal en 100 intervalles semble donc être le meilleur compromis entre rapidité d'exécution et précision

Naive-Bayes après simplification PCA





Le passage en vecteur à l'aide de PCA semble être couteux en temps de calcul et n'apporte pas de gain significatif en précision.
De plus le choix de la dimension n'affecte pas cette précision

Dans notre cas l'utilisation de l'algorithme PCA sera donc réservée à la modélisation de nos bases de données

Code Arduino

```
#include <MPU6050_tockn.h>
#include <Wire.h>

MPU6050 mpu6050(Wire);

long timer = 0;
float acceleration = 0;
float accX = 0;
float accY = 0;
float accZ = 0;
float seuil =0;

void setup() {
  Serial.begin(9600);
  pinMode(5, OUTPUT);
  Wire.begin();
  mpu6050.begin();
  mpu6050.calcGyroOffsets(true);
}

void loop() {

  mpu6050.update();

  accX = mpu6050.getAccX();
  accY = mpu6050.getAccY();
  accZ = mpu6050.getAccZ();
  acceleration = sqrt(accX*accX + accY*accY + accZ*accZ)

  if (acceleration > seuil) {
    digitalWrite(5, HIGH);
  }
}
```

Code Python

```
# Importation des bibliothèques


import csv
import math as m
import numpy as np
from sklearn import datasets, neighbors
from sklearn.decomposition import PCA
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import time
```

```
def initiali(lien, n=500):

    '''Prend en paramètre le lien d'un fichier csv et un entier n
        Décompose le signal en n intervalles
        Renvoi une liste de n normes correspondant à la moyenne du siganl sur chaque intervalle'''

    def moyenne(liste):
        somme = 0
        for i in liste:
            somme += i
        return somme/len(liste)

    fichier = open(lien)
    myReader = csv.reader(fichier)
    temp = []
    normes = []
    i = 0

    for row in myReader:
        if i == 0:
            i += 1
        else:
            new = row[0].split(';')
            temp.append(m.sqrt(float(new[3])**2 + float(new[4])**2 + float(new[5])**2))

    l = len(temp)//n
    for j in range(n):
        sous_liste = temp[j*l: (j+1)*l]
        normes.append(moyenne(sous_liste))

    return normes
```

```
def dist_euclidienne(vect1, vect2):
    somme = 0
    for i in range(len(vect1)):
        somme += (vect1[i] - vect2[i])**2
    return m.sqrt(somme)


def n_voisins(liste, n):

    def minimum(liste):
        indice_min = 0
        mini = liste[0]
        for i in range(len(liste)):
            x = liste[i]
            if mini > x:
                indice_min = i
                mini = x
        return indice_min

    def aux(liste, n, liste_minimum, comp=0):

        if n == 0:
            return liste_minimum
        else:
            i = minimum(liste)
            liste_minimum.append(i+comp)
            liste.pop(i)
            return n_voisins(liste, n-1, liste_minimum, comp+1)

    return aux(liste, n, [])
```

```
def methode_voisins(data_train, data_test, target_train,
target_test, nbr_voisins):

    resultats = []
    for dat in data_test:
        distances = []
        for x in data_train:
            distances.append(dist_euclidienne(dat, x))
        voisins = n_voisins(distances, nbr_voisins)
        res = 0
        for i in voisins:
            res += target_train[i]/nbr_voisins
        if res > 0.5:
            res = 1
        else:
            res =0
        resultats.append(res)
    result = np.array(resultats)

    error = sum(result - target_test)
    precision = 100 - abs((error / len(data_test) * 100))

    return precision
```

```
# Codage des méthodes, elles renvoient le pourcentage de réussite


def naiveBayes(data_train, data_test, target_train, target_test, *useless):

    # Traitement IA
    clf = GaussianNB()
    clf.fit(data_train, target_train)
    GaussianNB(priors=None)
    clf.get_params()
    result = clf.predict(data_test)

    # Analyse des résultats
    error = sum(result - target_test)
    precision = 100 - abs((error / len(data_test) * 100))

    return precision
```

```
def main(nbr_decompo, methode, *nombre_voisins, nombre_reduction=0):

    '''Prend en paramètres la méthode de traitement par intelligence artificielle
                            le nombre de valeurs
                            (le nombre de voisins pour KN)
                            (la dimension en laquelle on veut réduire notre data)
        Renvoie le pourcentage de réussite de la méthode'''

    a = [initiali(lien, nbr_decompo) for lien in liens_freeFall[::2]]
    b = [initiali(lien, nbr_decompo) for lien in liens_downSit[::2]]
    c = [initiali(lien, nbr_decompo) for lien in liens_freeFall[1::2]]
    d = [initiali(lien, nbr_decompo) for lien in liens_downSit[1::2]]
    e = [initiali(lien, nbr_decompo) for lien in liens_walkFall[::2]]
    f = [initiali(lien, nbr_decompo) for lien in liens_walkFall[1::2]]
    g = [initiali(lien, nbr_decompo) for lien in liens_runSit[::2]]
    h = [initiali(lien, nbr_decompo) for lien in liens_runSit[1::2]]

    data_train = np.array(a+e+b+g)
    target_train = np.array([0 for _ in range(len(a)+len(e))] + [1 for _ in range(len(b)+len(g))])
    data_test = np.array(c+f+d+h)
    target_test = np.array([0 for _ in range(len(c)+len(f))] + [1 for _ in range(len(d)+len(h))])

    if nombre_reduction != 0:
        data_train = simplificationPCA(data_train, nombre_reduction)
        data_test = simplificationPCA(data_test, nombre_reduction)

    return methode(data_train, data_test, target_train, target_test, nombre_voisins)
```

```
# Calcul de l'efficacité qualitative et temporelle de Naive-Bayes en fonction d’une liste de nombre d'intervalles de décomposition du signal


def decomposition_NB(liste):
    Y = []
    T = []
    for i in liste:
        start = time.time()
        pourcentage = main(500, naiveBayes, 0, i)
        end = time.time()
        Y.append(pourcentage)
        T.append(end-start)

    plt.plot(liste, Y)
    plt.xlabel('dimension')
    plt.ylabel('pourcentage de précision')
    plt.show()

    plt.plot(liste, T)
    plt.xlabel('dimension')
    plt.ylabel('temps de calcul')
    plt.show()
```

```
def trace_3D(list_data):

    '''Prend en paramètre une liste de data composé de vecteurs en diemnsions 3,
        Réalise un tracé en 3 dimensions de celle-ci'''

    for data in list_data:

        x = np.array([i[0] for i in data])
        y = np.array([i[1] for i in data])
        z = np.array([i[2] for i in data])

        # Tracé du résultat en 3D
        fig = plt.figure()
        ax = fig.gca(projection='3d')  # Affichage en 3D
        ax.scatter(x, y, z, label='Courbe', marker='d')  # Tracé des points 3D

    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    plt.tight_layout()
    plt.show()
```

```
def modelisation_data(nbr_decompo=500):

    data_freefall = [initiali(lien, nbr_decompo) for lien in liens_freeFall]
    data_downSit = [initiali(lien, nbr_decompo) for lien in liens_downSit]
    data_walkFall = [initiali(lien, nbr_decompo) for lien in liens_walkFall]
    data_runSit = [initiali(lien, nbr_decompo) for lien in liens_runSit]

    trace_3D([data_freefall, data_downSit, data_walkFall, data_runSit])



def graphe_data():

    X1 = initiali(liens_freeFall[0], 20)
    X2 = initiali(liens_walkFall[0], 20)
    X3 = initiali(liens_runSit[0], 20)
    X4 = initiali(liens_downSit[0], 20)
    Y = [i for i in range(len(X1))]

    plt.plot(Y, X1, label='freeFall')
    plt.plot(Y, X2, label='walkFall')
    plt.plot(Y, X3, label='runSit')
    plt.plot(Y, X4, label='downSit')
    plt.legend()
    plt.show()
```