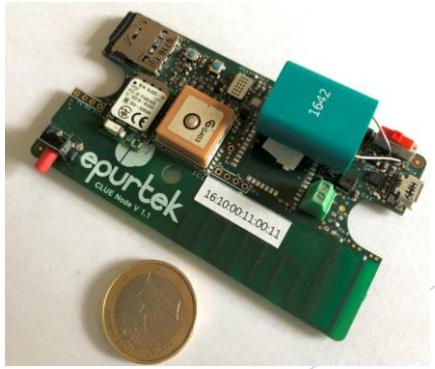
Conception et développement d'un outil de détection du mode de transport à partir d'une trace GPS et d'un accéléromètre

Encadrants :
Christophe **BERTERO**
Euriell **LE CORRONC**

Etudiants M1 ISTR-RODECO :

- Ayoub **BENTAFAT**
- Mohamed Bedir **KACIMI EL HASSANI**
- Smail **GOURMI**
- Yasser **TOUATI**



PLAN

I. Introduction
II. Acquisition des données
III. Pré-traitement (Preprocessing)
IV. Apprentissage automatique (machine learning)
V. Resultats
VI. Conclusion

Introduction

Objectifs: Detection mode de transport

Python

Intelligence artificielle

Apprentissage automatique (machine learning)Supervisé

Entrainement→modéle

Test→validation de l'outil

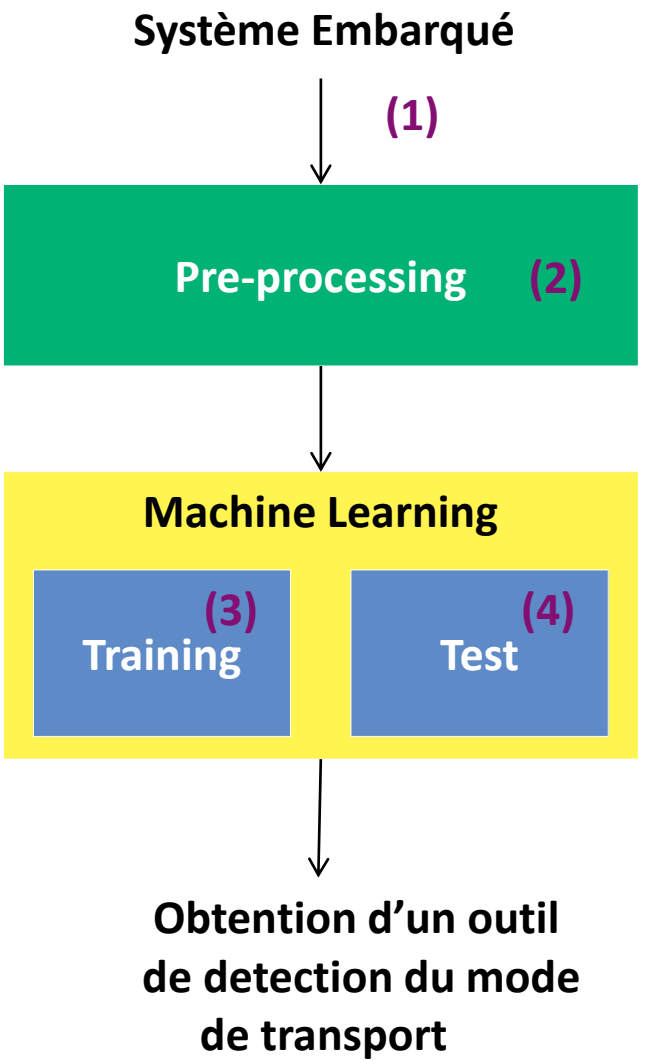
Étapes du projet

1)Acquisition des données

2)Pré-traitement (Preprocessing)

3)Machine Learning :

- Phase de training
- Phase de Test

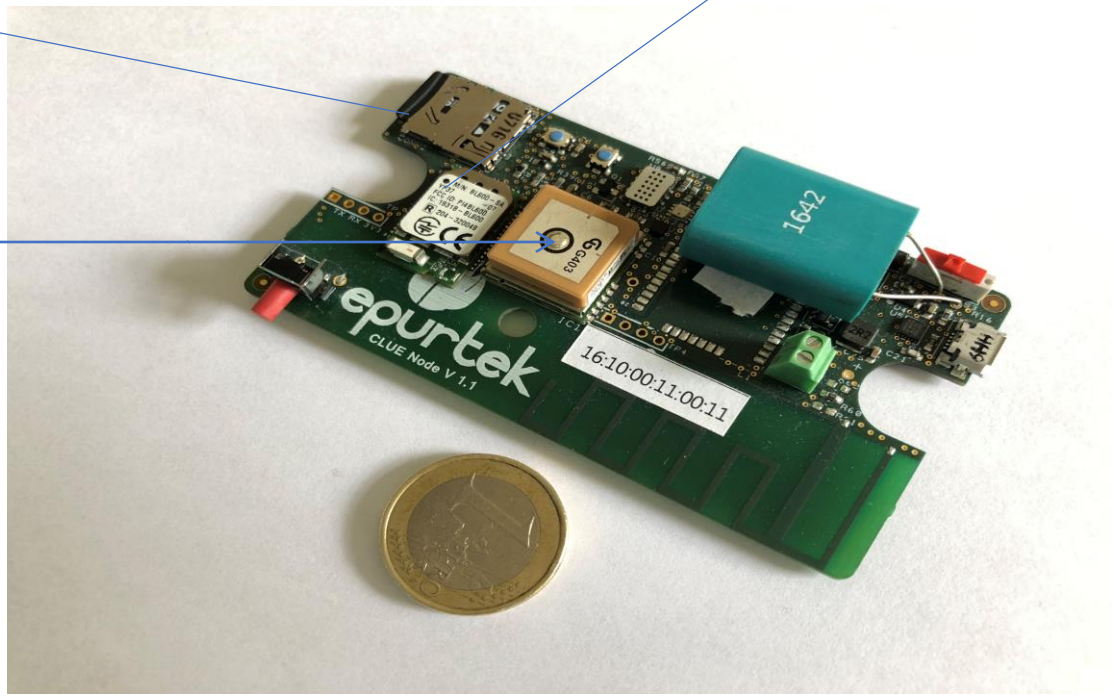


Système embarqué

Accéléromètre
(LIS2DH12)

Carte SD

Capteur GPS
(A2135_HA2235)



Acquisition de données

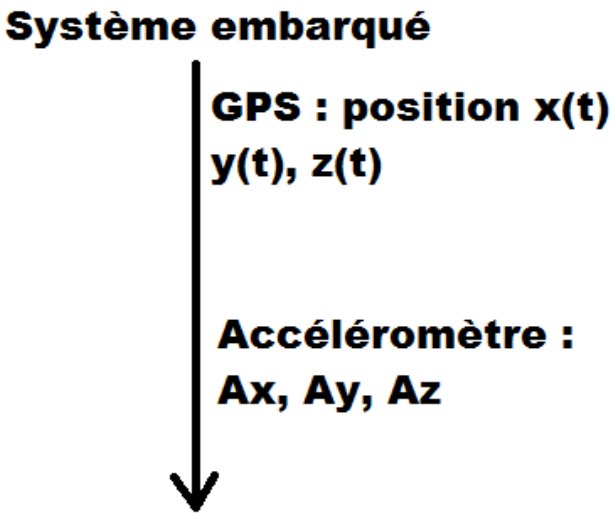
Capteurs & Fonctionnement

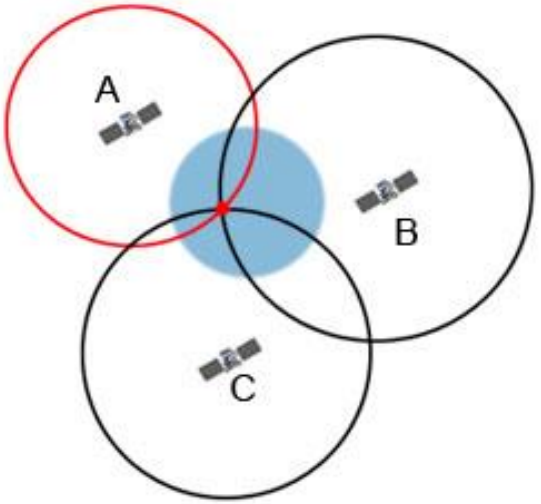
GPS

4 satellites

Données : Position x,y,z

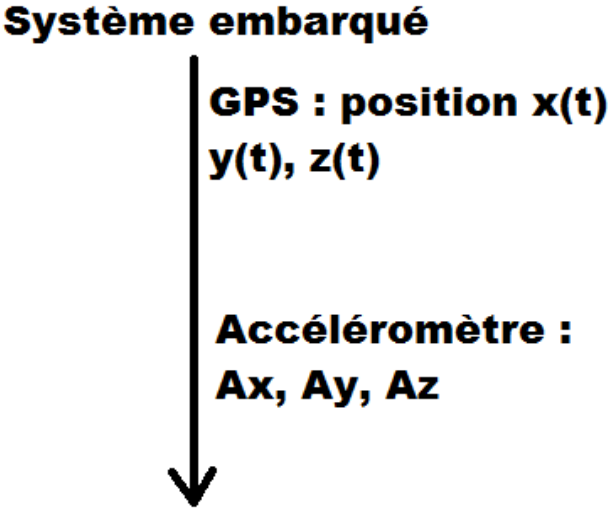
Trilateration





Acquisition de données

Capteurs & Fonctionnement

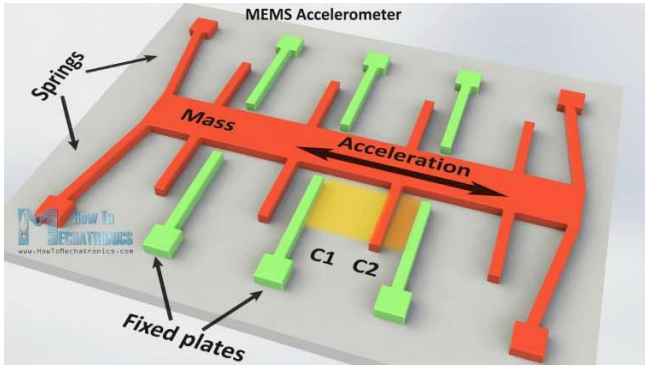


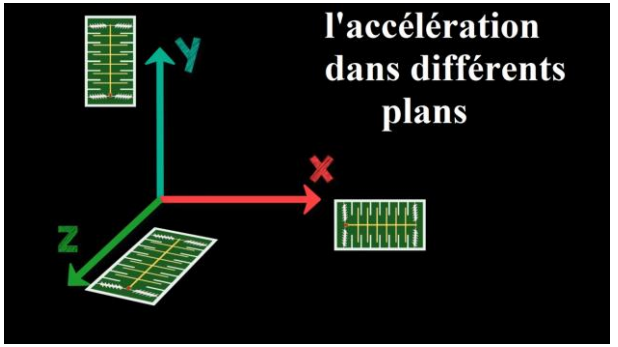
- Accéléromètre

- Capteur mesurant l'accélération

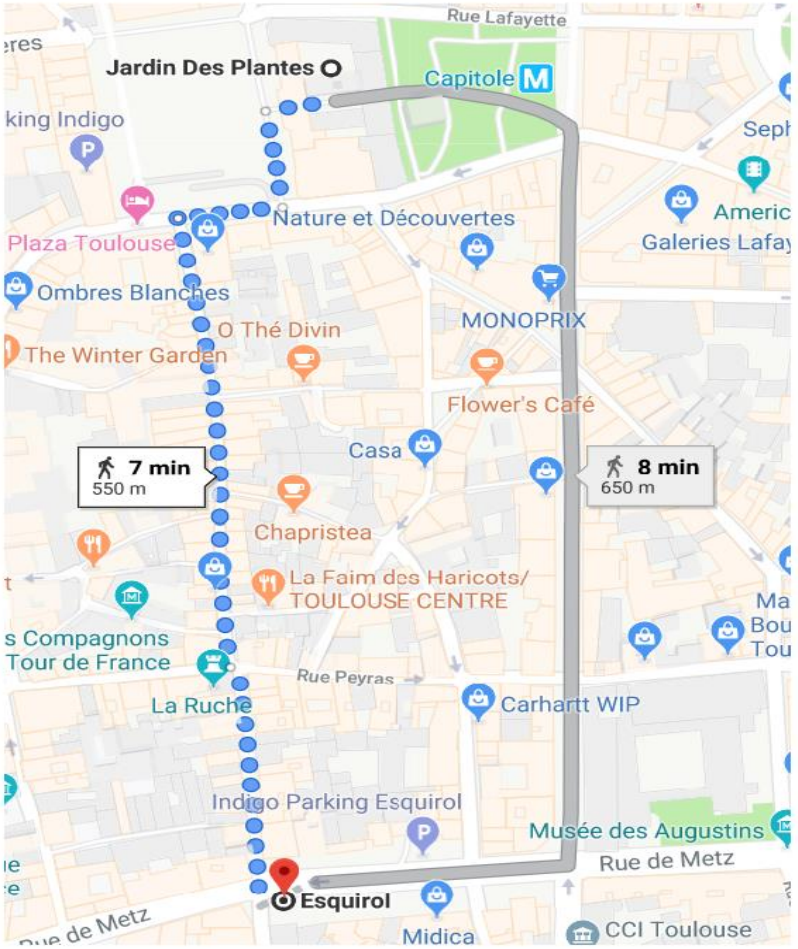
Données récupérés : Ax, Ay, Az.

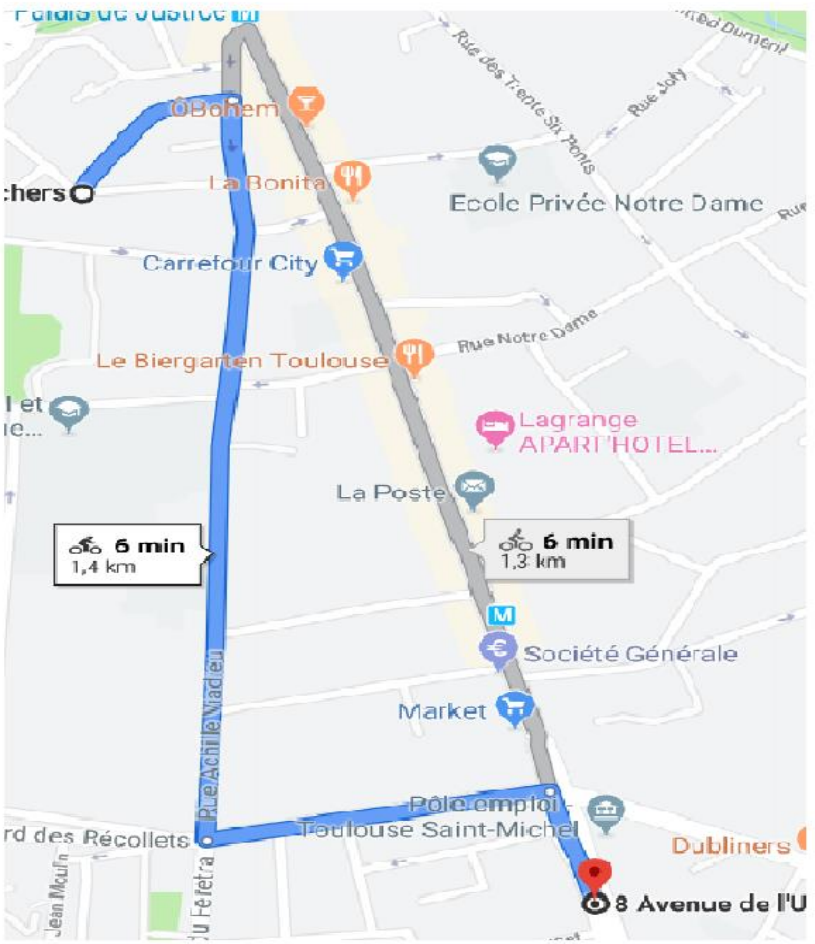
fonctionnement :





Protocole expérimentale :





⟶ Trajet: Jardin des plantes -> Esquirol

⟶ Durée : 12 min

Nb total d'échantillions : 18 642

Pré-traitement

Nettoyage des données

1-Filtrage

2290	19-03-31	16:32:58	GPS	435920164	14442563	-5039	3
2291	19-03-31	16:32:58	BME	99831	2675	34763	nan
2292	19-03-31	16:32:58	MOX	9406	130222121	65537	4259840
2293	19-03-31	16:32:59	GPS	435919117	14442244	-5039	3
2294	19-03-31	16:32:59	BME	99828	2674	34854	nan
2295	19-03-31	16:32:59	MOX	9284	142929825	65537	4259840
2296	19-03-31	16:33:00	GPS	435917632	14440935	-5039	3
2297	19-03-31	16:33:00	BME	99831	2677	34890	nan
2298	19-03-31	16:33:00	MOX	9324	142929825	65537	4259840
2299	19-03-31	16:33:01	ACC	0	0	1	65
2300	19-03-31	16:33:01	ACC	1	-1	2	66
2301	19-03-31	16:33:01	ACC	2	0	1	65
2302	19-03-31	16:33:01	ACC	3	1	1	65

Nb d'échantillions vélo : 4249

Nb d'échantillions non-vélo : 2101

Total : 6350

Mode	Temps(mn)	Nombre d'echantillion
Acc-Non-Velo	28	736
Acc-Velo	55	1568
Gps-Non-Velo	28	1365
Gps-Velo	55	2681

Pré-traitement

Normalisation

GPS

X	Y	Z
4.3595e+08	1.44928e+07	15043

Xn	Yn	Zn
0.0159692	0.820141	0.0400845

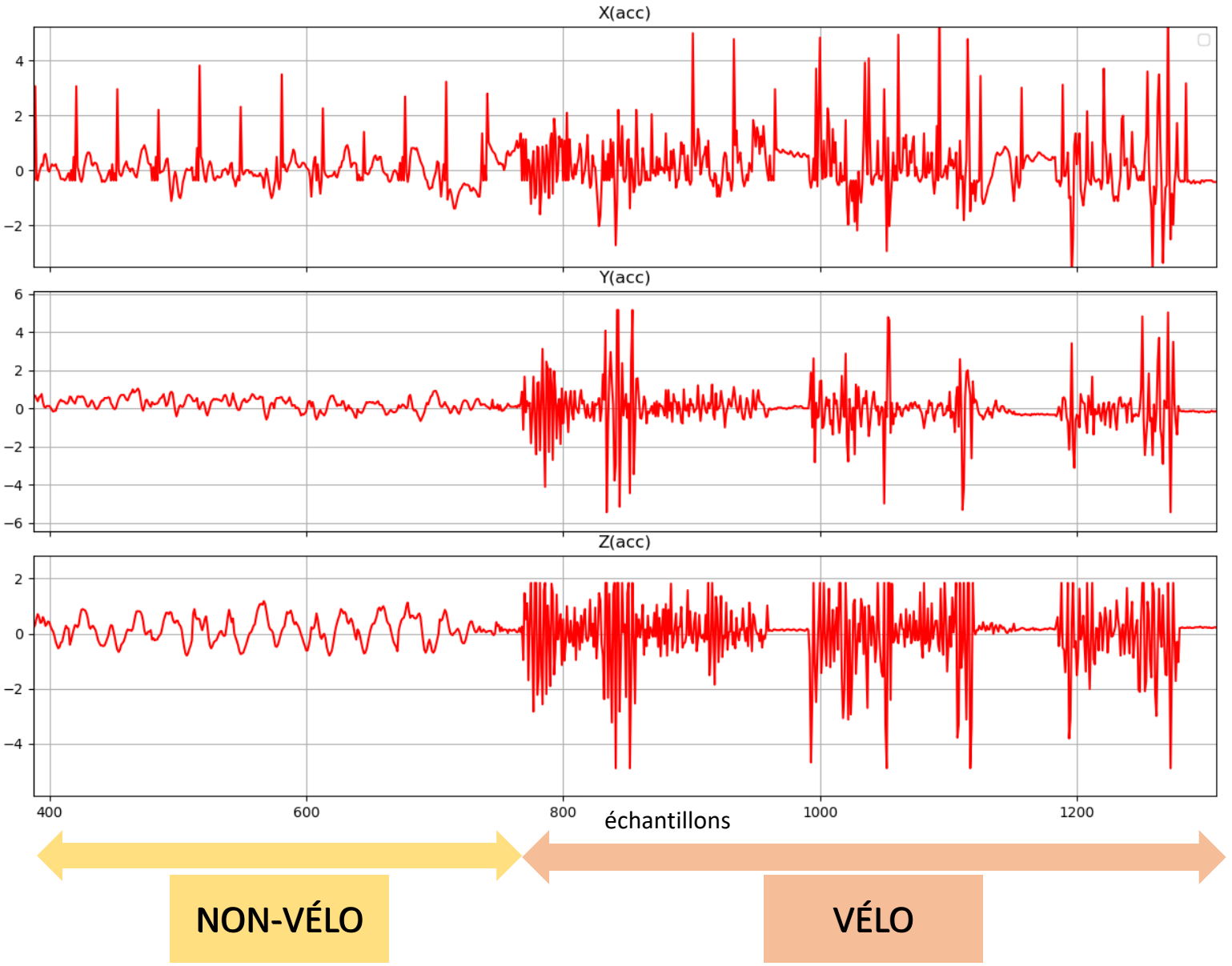
Acélérometre

X	Y	Z
61	24	61

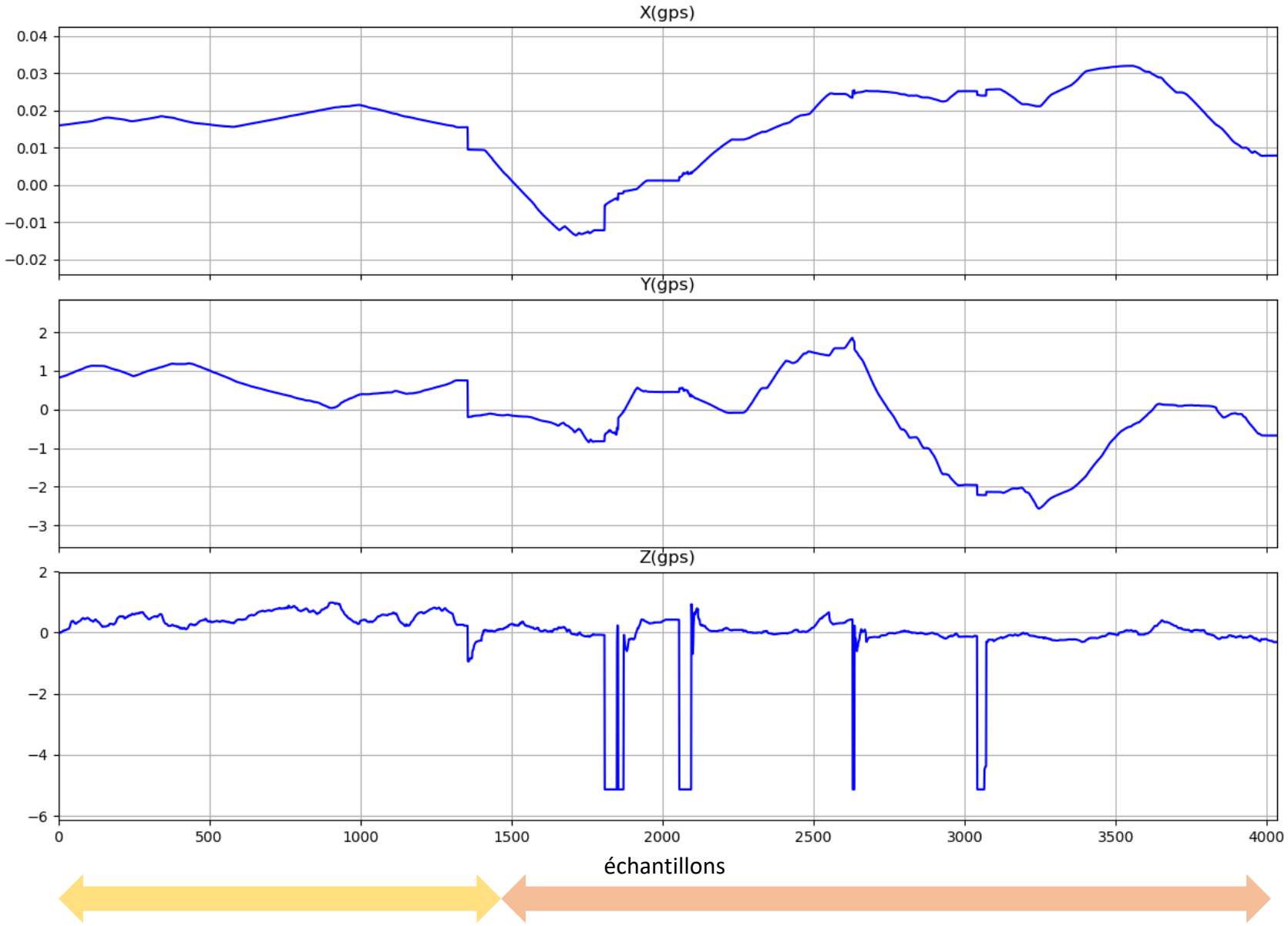
Xn	Yn	Zn
2.90183	0.888835	0.0965789

```
89 from sklearn.preprocessing import StandardScaler
90
91 std = StandardScaler()
92 X1 = std.fit_transform(data_test_acc)
93 X2 = std.fit_transform(data_test_gps)
94
95 data_norm_acc = pd.DataFrame(X1, columns=data_test.columns)
96 data_norm_gps = pd.DataFrame(X2, columns=data_test.columns)
```

Données : Acc


X(acc)
Y(acc)
Z(acc)
400
600
800
1000
1200
échantillons
NON-VÉLO
VÉLO

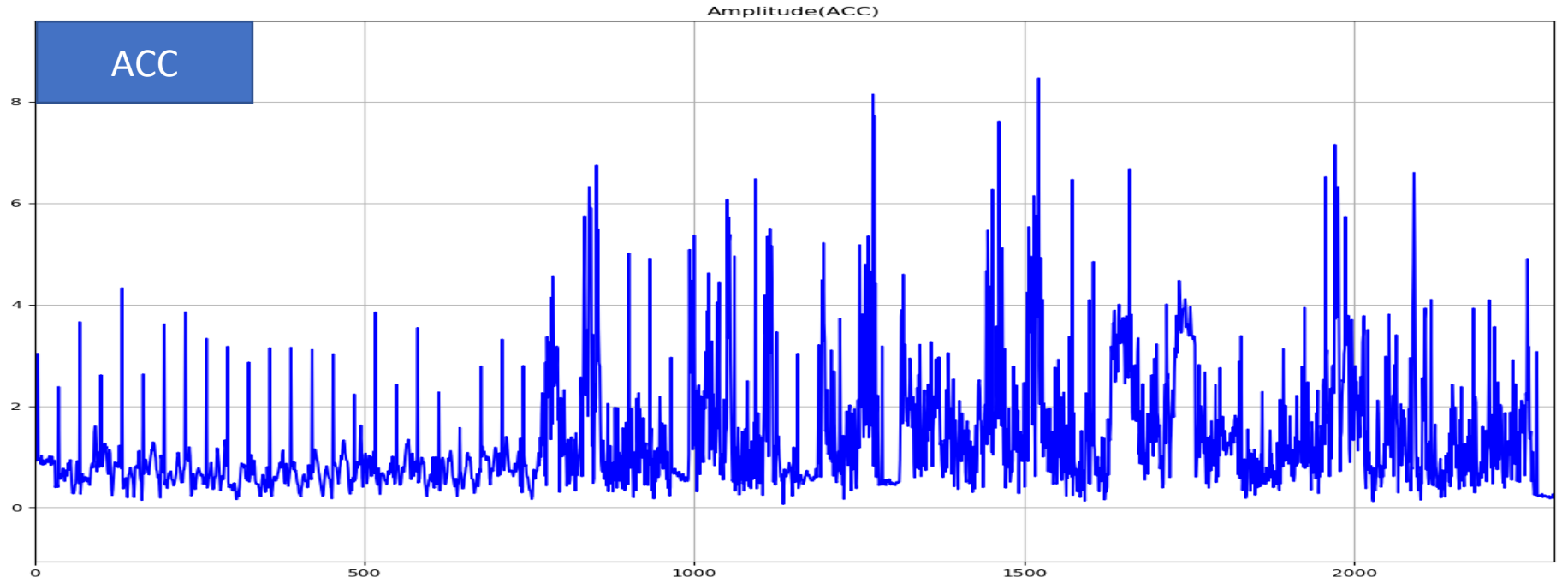
Données : GPS

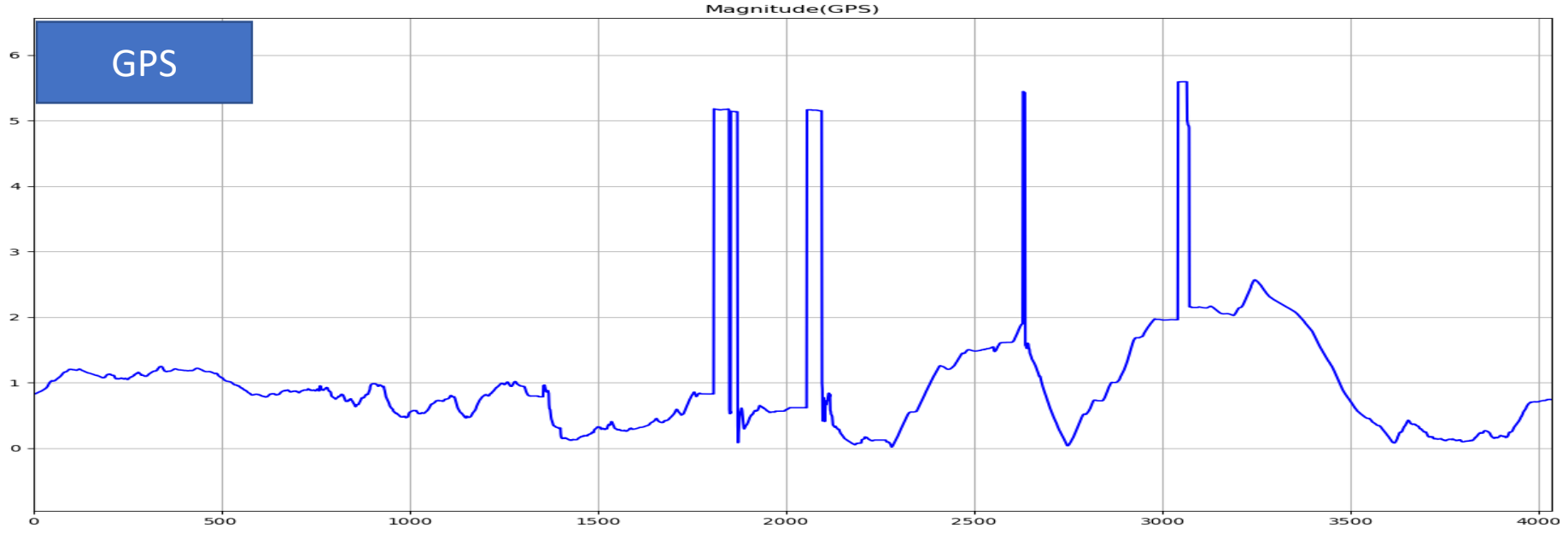


Calcul de l'Amplitude (carac.) :

1. Orientation du système embarqué inconnue
2. Données qui regroupe les informations concernant les 3 axes en même temps

```
173 import math
174
175 def magnitude(activity):
176     x2 = activity['Xn'] * activity['Xn']
177     y2 = activity['Yn'] * activity['Yn']
178     z2 = activity['Zn'] * activity['Zn']
179     m2 = x2 + y2 + z2
180     m = m2.apply(lambda x: math.sqrt(x))
181     return m
182 data_acc['magnitude']=magnitude(data_acc)
183 data_gps['magnitude']=magnitude(data_gps)
```

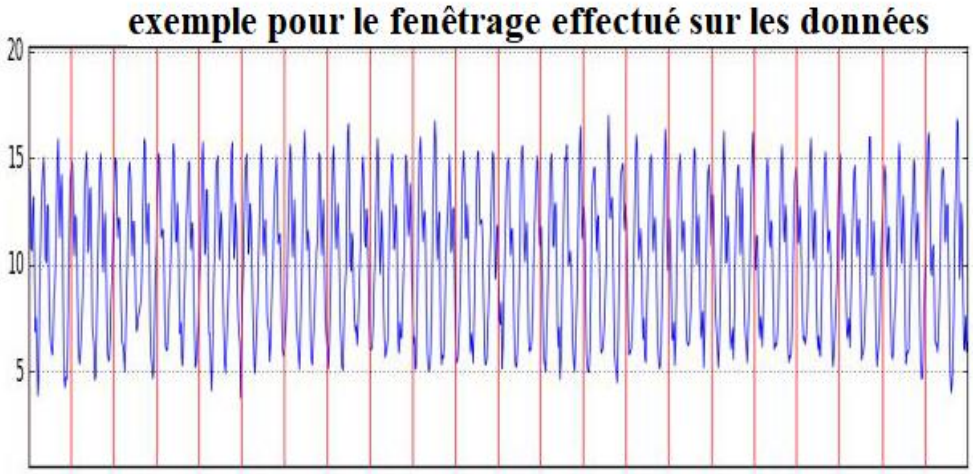

ACC
Amplitude(ACC)
0
2
4
6
8
0
500
1000
1500
2000


GPS
Magnitude(GPS)
0
1
2
3
4
5
6
0
500
1000
1500
2000
2500
3000
3500
4000

Fenêtrage :

Découpage des données en plusieurs intervalles ou fenêtres

Une fenêtre de 3 points est utilisée pour la période de classification pour éviter le bruit et éviter dede nuire à la précision




```python
#Function for defining the window on data
def window(axis,dx=2):
    start = 0;
    end=0;
    size = axis.count();
    print("start: ",start,"size : ",size)
    while (start <= size and end <= size ):
        end = start + dx
        if(end <= size):
            yield start,end
        start = start+int (dx/2)
```

On calcule les caractéristiques pour chaque fenêtre

Extraction des caractéristiques a partir des fenêtres

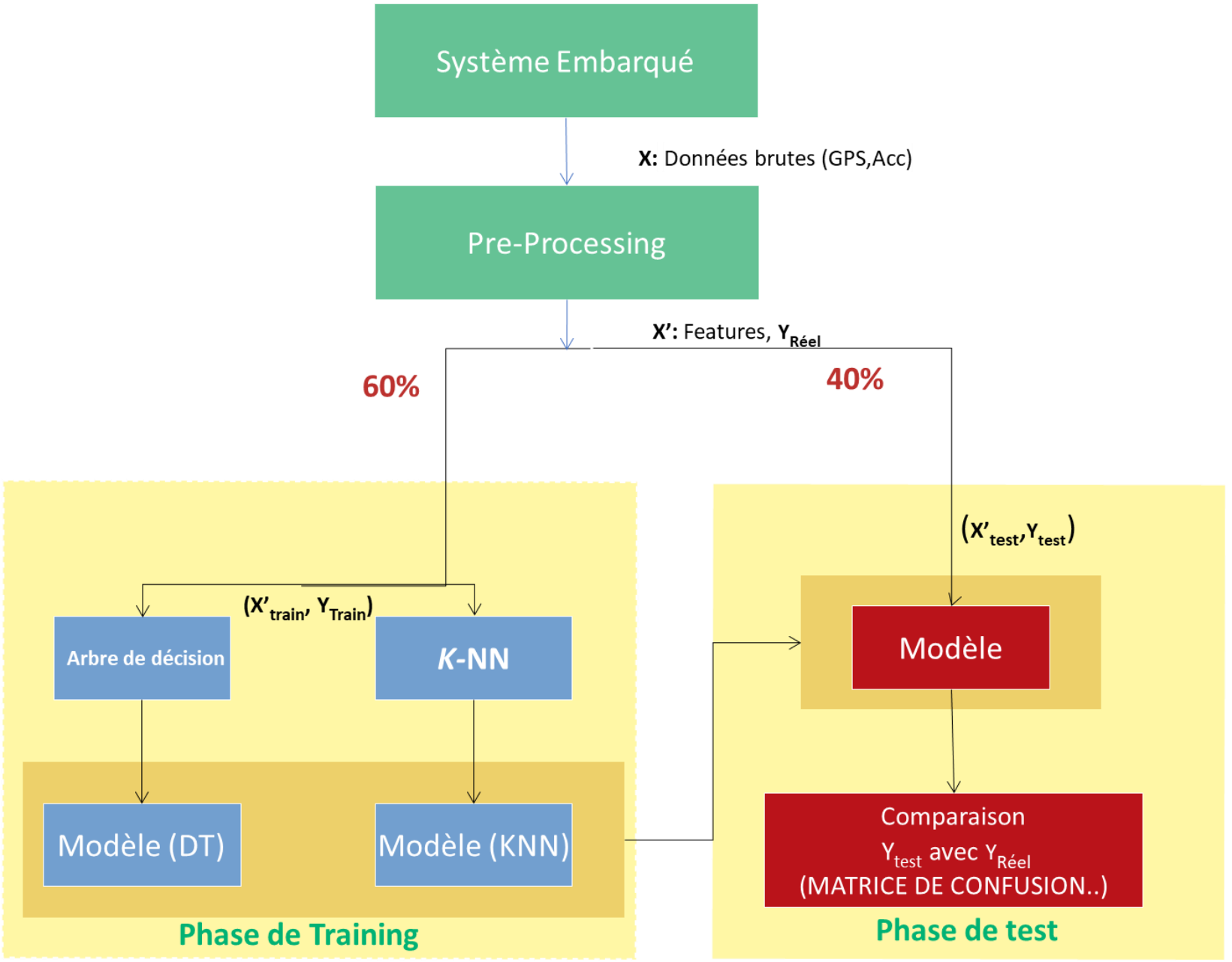
```python
#Features which are extracted from Raw sensor data
def window_summary(axis, start, end):
#    print("start: ",start,"size : ",end)
    acf = stattools.acf(axis[start:end])
#    print("acf: ",acf)
    acv = stattools.acovf(axis[start:end])
    sqd_error = (axis[start:end] - axis[start:end].mean()) ** 2
    return [
        axis[start:end].mean(),
        axis[start:end].std(),
        axis[start:end].var(),
        axis[start:end].min(),
        axis[start:end].max(),
        acf.mean(), # mean auto correlation
        acf.std(), # standard deviation auto correlation
        acv.mean(), # mean auto covariance
        acv.std(), # standard deviation auto covariance
        skew(axis[start:end]),
        kurtosis(axis[start:end]),
        math.sqrt(sqd_error.mean())
    ]
```

```python
#Add an additional axis of magnitude of the sensor data
from scipy.stats import skew, kurtosis
from statsmodels.tsa import stattools
```

Index	('', 'MODE')	('', 'ACC-GPS')	('X', 'MEAN')	('X', 'STD')	('X', 'VAR')	('X', 'MIN')	('X', 'MAX')
2300	1	0	0.00356413	0.0759031	0.00576128	-0.0501074	0.0572357
2301	1	0	-0.0232717	0.113855	0.0129629	-0.103779	0.0572357
2302	1	0	-0.0232717	0.113855	0.0129629	-0.103779	0.0572357
2303	0	1	0.0159693	1.14518e-07	1.31143e-14	0.0159692	0.0159694
2304	0	1	0.015975	7.90173e-06	6.24373e-11	0.0159694	0.0159806
2305	0	1	0.0159862	7.90173e-06	6.24373e-11	0.0159806	0.0159918

Mode : (0 : Non-Vélo),(1 : Vélo)

Apprentissage automatique (machine learning)



Code de DT & k-nn :

1- Apprentissage (training)

```
325 #Create KNN Classifier
326 knn = KNeighborsClassifier(n_neighbors=2)
327
328 #Create Decision Tree Classifier
329 dt = DecisionTreeClassifier(criterion="entropy", max_depth=10) # generates predi
330
331 dt_result = []
332 knn_result=[]
333
334 for i in range(0, 10):
335     X_train, X_test, y_train, y_test =train_test_split(X, Yreel, test_size=.4)
336     dt.fit(X_train, y_train.values.ravel())
337     knn.fit(X_train, y_train.values.ravel())
```

2-test

```
339     decitree =dt.score(X_test, y_test)
340     kn = knn.score(X_test, y_test)
```

3-cross validation

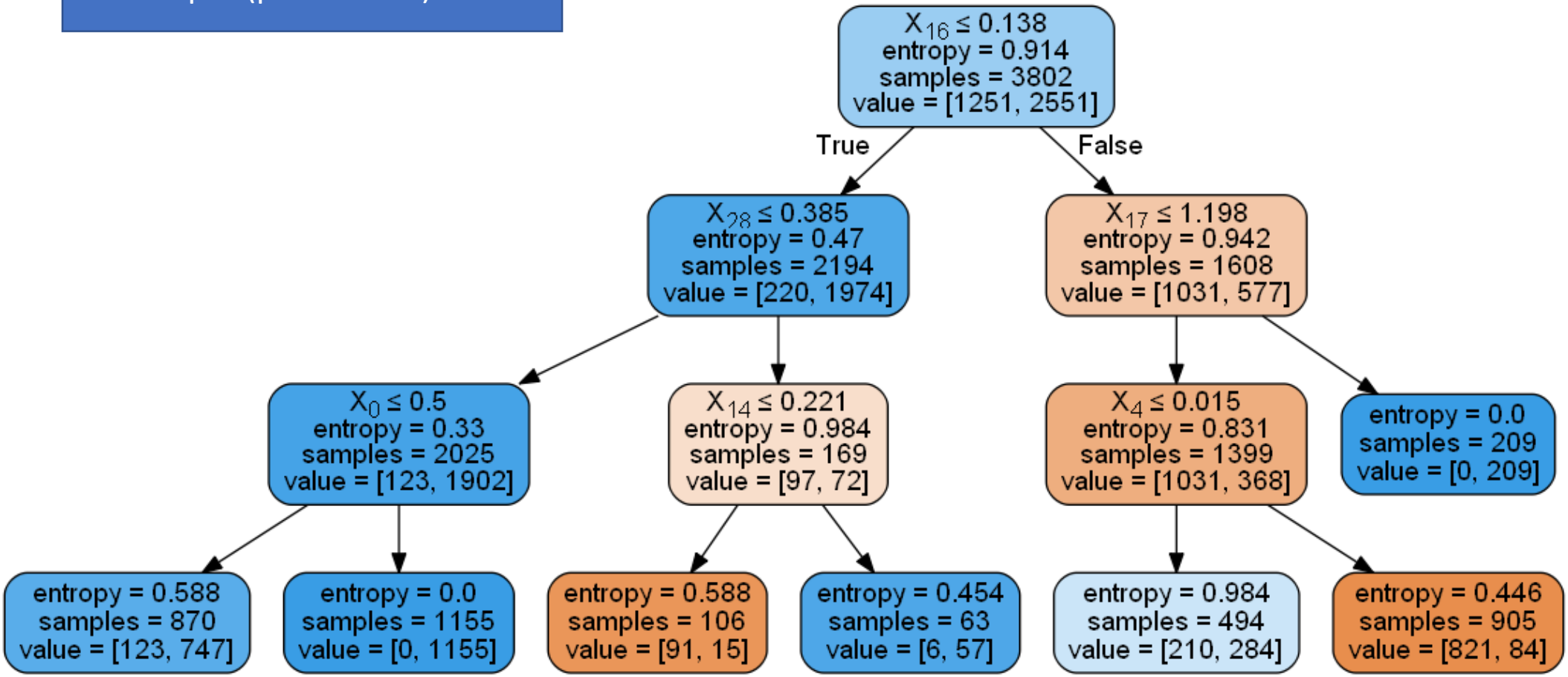
```
399 from sklearn.model_selection import cross_val_score
400 scores = cross_val_score(knn, X, Yreel.values.ravel(), cv=5)
401 print("cross validationn : ",scores,"\n")
```

4-matrice de confusion

```
378 y_pred_knn = knn.predict(X_test)
379 print("Confusion Matrix Knn:\n",confusion_matrix(y_test,y_pred_knn))
380
381 y_pred_dt = dt.predict(X_test)
382 print("Confusion Matrix decisionTree:\n",confusion_matrix(y_test,y_pred_dt))
```

L’arbre de décision

Depth (profondeur) =3



3-validation croisée

1. La validation croisée peut être effectuée k fois, pour éviter le phénomène de overfitting.
2. L'ensemble de données est partitionné de manière aléatoire en k sous-ensembles mutuellement exclusifs

```
Loop  0  :  DecisionTree :  0.8749506903353057  KNN :  0.9479289940828403
Loop  1  :  DecisionTree :  0.8848126232741618  KNN :  0.9518737672583827
Loop  2  :  DecisionTree :  0.8741617357001973  KNN :  0.947534516765286
Loop  3  :  DecisionTree :  0.8717948717948718  KNN :  0.9483234714003945
Loop  4  :  DecisionTree :  0.8785009861932939  KNN :  0.9495069033530572
Loop  5  :  DecisionTree :  0.8749506903353057  KNN :  0.9463510848126233
Loop  6  :  DecisionTree :  0.8792899408284024  KNN :  0.9408284023668639
Loop  7  :  DecisionTree :  0.8800788954635108  KNN :  0.9487179487179487
Loop  8  :  DecisionTree :  0.850887573964497  KNN :  0.9439842209072978
Loop  9  :  DecisionTree :  0.8733727810650888  KNN :  0.9487179487179487
```

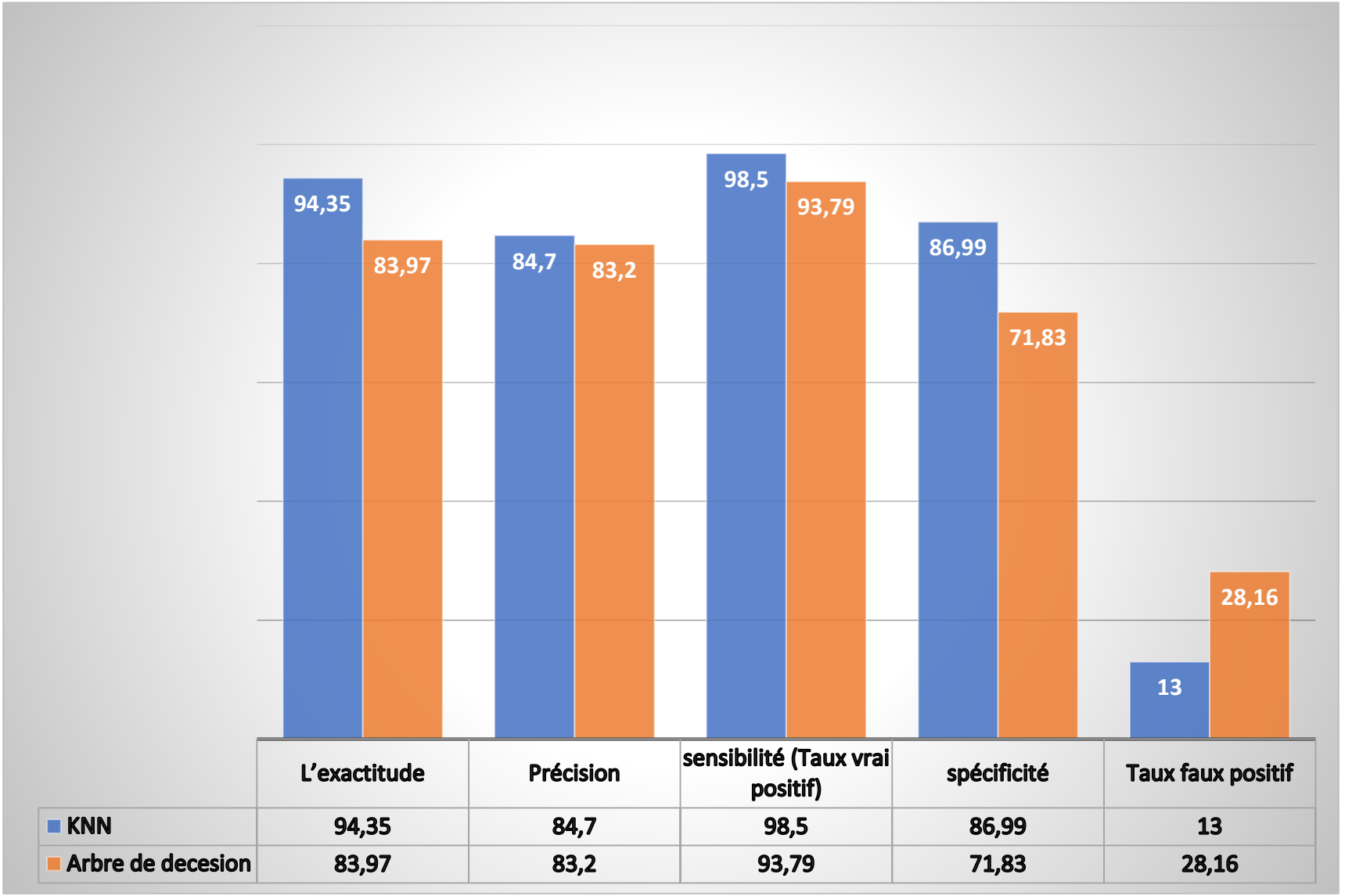
	Moyenne de l'exactitude	Ecart type
KNN	0.947	0.002
Arbre de décision	0.874	0.008

4-matrice de confusion

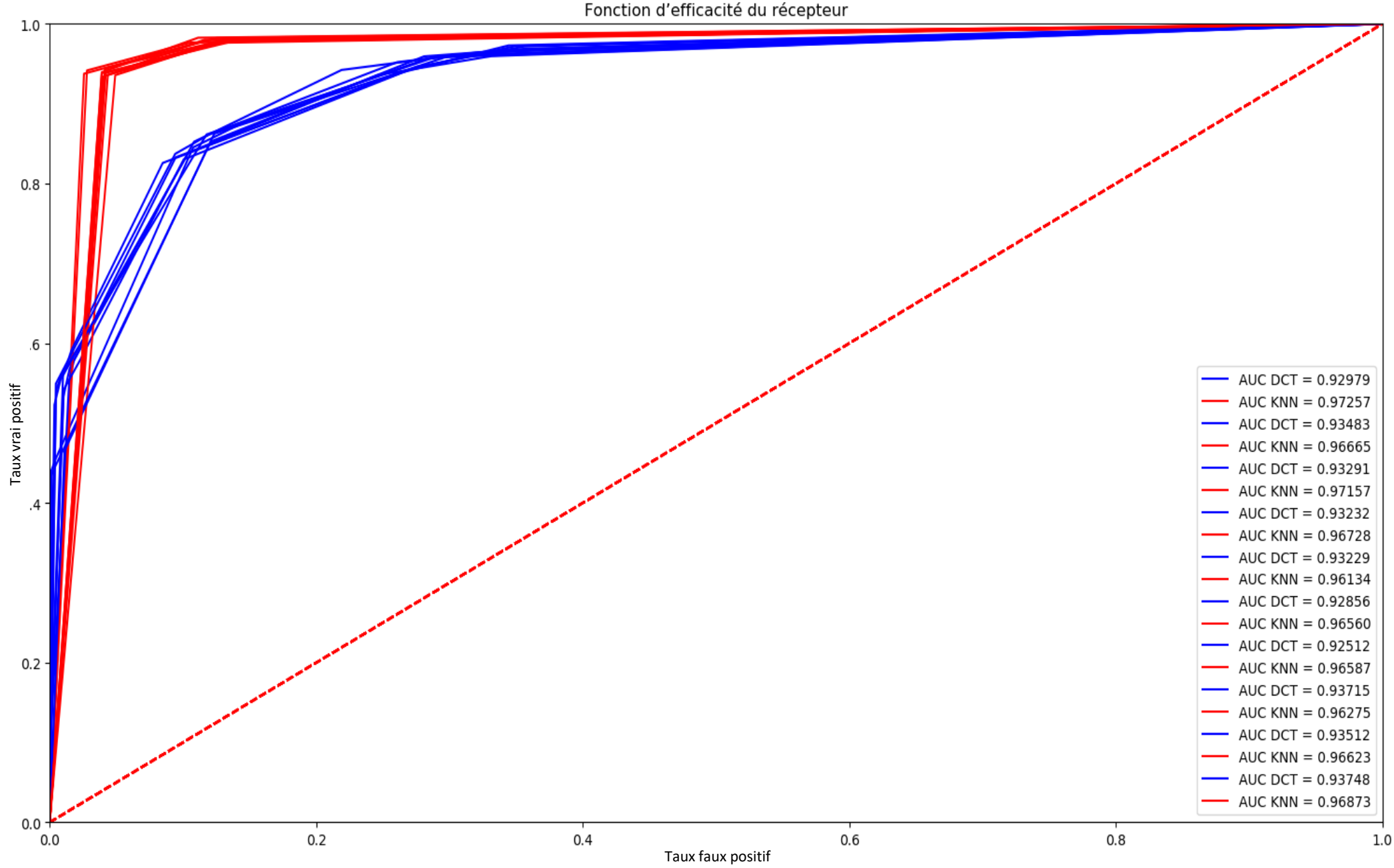
KNN (2535)		Classe Réelle	
		Non-Vélo	Vélo
Classe prédite	Non-vélo	Vrai négative(812)	Faux négative(23)
	Vélo	Faux positive(102)	Vrai positive(1598)

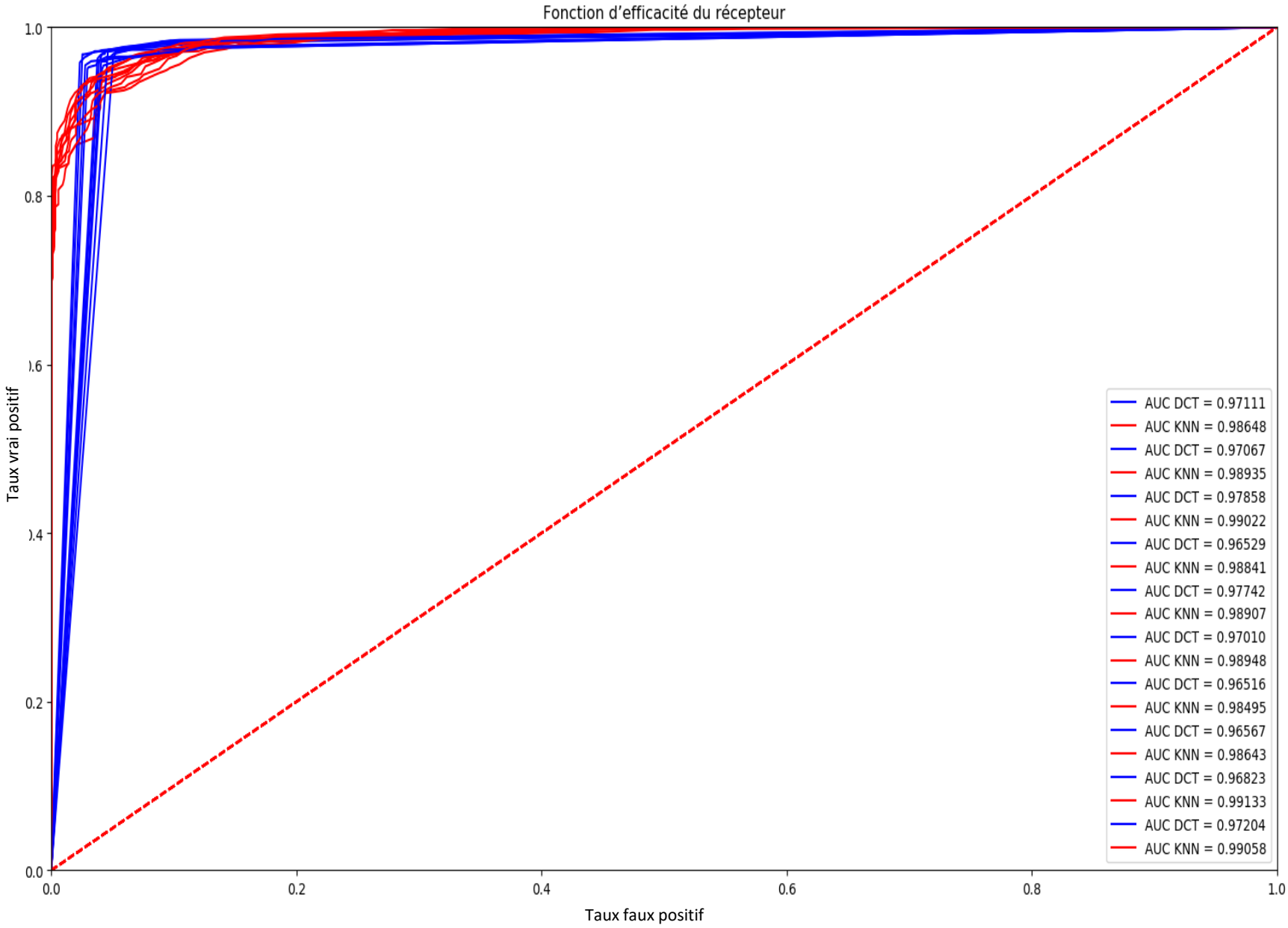
L'arbre de décision (2535)		Classe Réelle	
		Non-Vélo	Vélo
Classe prédite	Non-vélo	Vrai négative(798)	Faux négative(37)
	Vélo	Faux positive(58)	Vrai positive(1642)

Validation de model



	L'exactitude	Précision	sensibilité (Taux vrai positif)	spécificité	Taux faux positif
KNN	94,35	84,7	98,5	86,99	13
Arbre de decesion	83,97	83,2	93,79	71,83	28,16


Fonction d'efficacité du récepteur
Taux vrai positif
Taux faux positif
0.0
0.2
0.4
.6
0.8
1.0
AUC DCT = 0.92979
AUC KNN = 0.97257
AUC DCT = 0.93483
AUC KNN = 0.96665
AUC DCT = 0.93291
AUC KNN = 0.97157
AUC DCT = 0.93232
AUC KNN = 0.96728
AUC DCT = 0.93229
AUC KNN = 0.96134
AUC DCT = 0.92856
AUC KNN = 0.96560
AUC DCT = 0.92512
AUC KNN = 0.96587
AUC DCT = 0.93715
AUC KNN = 0.96275
AUC DCT = 0.93512
AUC KNN = 0.96623
AUC DCT = 0.93748
AUC KNN = 0.96873


Fonction d'efficacité du récepteur
Taux vrai positif
Taux faux positif
0.0
0.2
0.4
0.6
0.8
1.0
AUC DCT = 0.97111
AUC KNN = 0.98648
AUC DCT = 0.97067
AUC KNN = 0.98935
AUC DCT = 0.97858
AUC KNN = 0.99022
AUC DCT = 0.96529
AUC KNN = 0.98841
AUC DCT = 0.97742
AUC KNN = 0.98907
AUC DCT = 0.97010
AUC KNN = 0.98948
AUC DCT = 0.96516
AUC KNN = 0.98495
AUC DCT = 0.96567
AUC KNN = 0.98643
AUC DCT = 0.96823
AUC KNN = 0.99133
AUC DCT = 0.97204
AUC KNN = 0.99058

Conclusion

1) Amélioration des résultats
2) Apprentissage automatique (avantage/désavantage)
3) Autre méthode : contraintes physiques (avantage/dés.)
   => Coder des seuills directement dans le code Python
4) Emergence des systèmes informatiques embarqués :
   Nvidia Jtson

MERCI POUR VOTRE ATTENTION.