PROJET DE MACHINE ET DEEP LEARNING

Système de détection en continu des épisodes de FA

Réalisé par :

EL-YAHYAOUI Abdenasser

ELKHATTARI Rabha

Filière: Ingénieur Cybersécurité et Confiance Numérique

Encadré par :

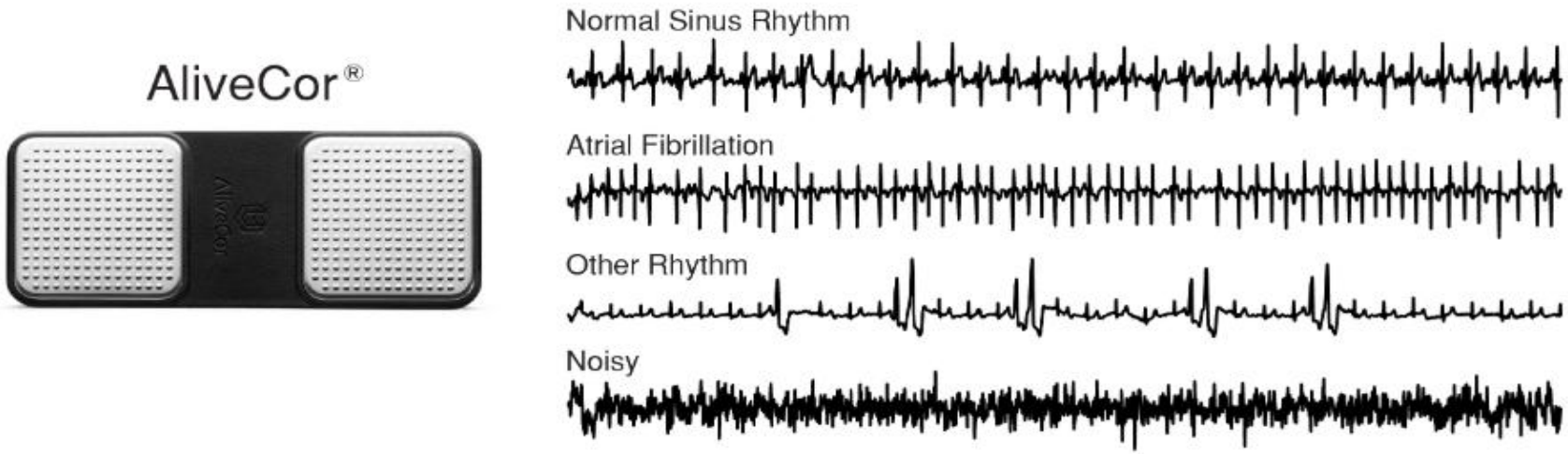
Pr. Abdelhak MAHMOUDI UM5 Rabat

Plan

I. *Introduction et problématique.*
II. *Algorithme d'apprentissage*
    - o*KNN*
    - o*DNN*
III. *Réalisation*
    - o*Implémentation*
    - o*Etude comparative*
IV. *Conclusion*

Introduction &Problématique :

>>Inefficacité des appareils de mesure de ECG pour la détection des FA

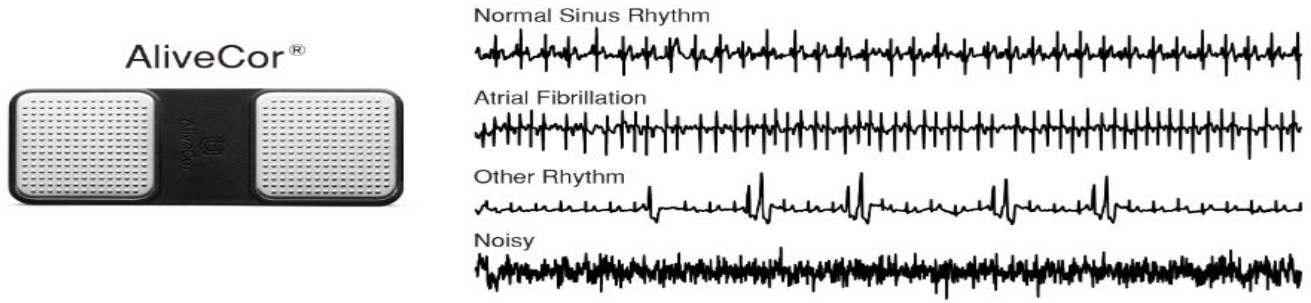


Gauche: dispositif d'acquisition ECG portatif AliveCor. Droite: Exemples d'enregistrement ECG pour chaque classe de rythme, Goodfellow et al. (2018).

==>Besoin d’un système qui détecte et avec précision ces FA

Base de Données

- Dans le Physionet Challenge 2017 , les concurrents ont été invités à construire un **modèle pour classer** une seule forme d'onde ECG de plomb comme rythme *sinusal normal*, *fibrillation auriculaire*, autre rythme ou bruyant



Gauche: dispositif d'acquisition ECG portatif AliveCor. Droite: Exemples d'enregistrement ECG pour chaque classe de rythme, Goodfellow et al. (2018).

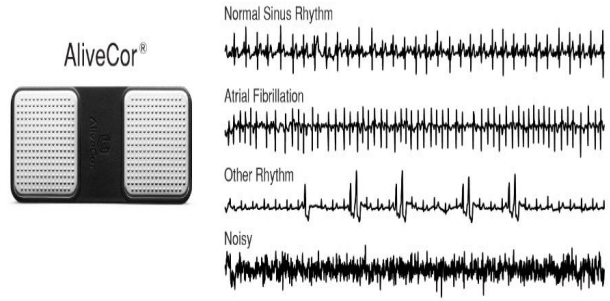
- L'ensemble de données comprenait 12 186 formes d'ondes ECG qui ont été données par AliveCor

Base de Données

- Pour télécharger le dataset (target non inclut) sur Colab :

  on utilise la commande : **!wget -r -N -c -np**

  https://physionet.org/files/challenge-2017/1.0.0/training2017.zip?download

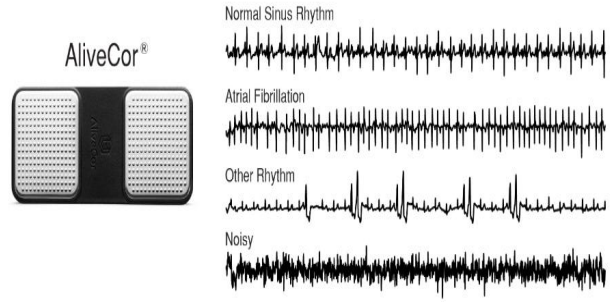


Gauche: dispositif d'acquisition ECG portatif AliveCor. Droite: Exemples d'enregistrement ECG pour chaque classe de rythme, Goodfellow et al. (2018).

les étiquettes (REFERENCE-v0.csv) de même lien https://physionet.org/files/challenge-2017/1.0.0/ REFERENCE-v0.csv

Base de Données

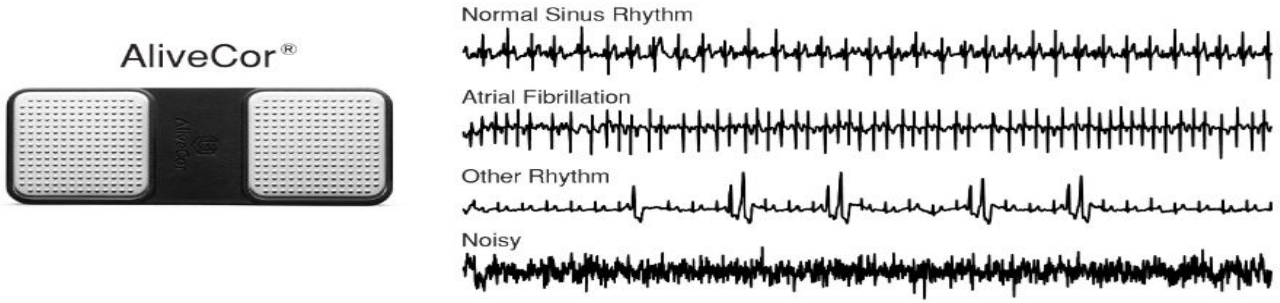
- avec N est le nombre de records signaux de toutes la base de donnée ,on store leurs valeurs échantillonné de tous les signaux.mat (des fichiers matlab A000i.mat) dans *une matrice de N ligne*
- sur la dernière colonne de la matrice on a inséré les labels ,qui se trouvent dans le fichier **REFERENCE-v0.csv** correspondant à chaque signal A000i.mat
  - Avec i varie entre 0 et N
  - Utilisation de la methode : ***ONE HOT ENCODING***
    - ü Car on dispose de 4 classes à classifier



Gauche: dispositif d'acquisition ECG portatif AliveCor. Droite: Exemples d'enregistrement ECG pour chaque classe de rythme, Goodfellow et al. (2018).

Base de Données

- par conséquent on a obtenu une seule matrice qui contient toute la base de donnée :



Gauche: dispositif d'acquisition ECG portatif AliveCor. Droite: Exemples d'enregistrement ECG pour chaque classe de rythme, Goodfellow et al. (2018).

  - (la colonne ‘trainset’) avec leur étiquettes (la colonne ‘traintarget’)
  - Elle est sauvegardé sur Google Drive dans le chemin suivant :
    - Ø **/drive/My Drive/notre base de données/trainingset.mat** (***2 GO***).

Le_classificateur_KNN:

- L’algorithme K-NN (K-nearest neighbors) est une méthode d’apprentissage supervisé. Il peut être utilisé aussi bien pour la régression que pour la classification.
- Pour effectuer une prédiction ,il suit le principe suivant :

  “dis moi qui sont tes voisins, je te dirais qui tu es…”.

- Il est dans la bibliothéque **sickitlearn** du module ***Neighbors***

Le_classificateur_KNN:

- Avec un nombre de voisin ***K=5*** et la distance euclidean on a obtenu le résutat:

```
[10] import numpy as np
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.model_selection import train_test_split
     from sklearn.model_selection import GridSearchCV
     from sklearn.model_selection import KFold
     from sklearn.model_selection import cross_val_score
```

```
[11] model=KNeighborsClassifier(n_neighbors=5,metric='euclidean')
```

```
[12] #XX,Xy,YY,Yx=train_test_split(X,y,test_size=0.5)
     X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2)
     print('Train set:',X_train.shape)
     print('Test set:',X_test.shape)
```

```
     Train set: (11141, 18000)
     Test set: (2786, 18000)
```

```
[13] model.fit(X_train,y_train)
```

```
     KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='euclidean',
                          metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                          weights='uniform')
```

```
[14] model.score(X_test,y_test)
```

```
     0.4382627422828428
```

Le_classificateur_KNN :

Puis avec un K=10 et la même distance euclidean:

```
[10] import numpy as np
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.model_selection import train_test_split
     from sklearn.model_selection import GridSearchCV
     from sklearn.model_selection import KFold
     from sklearn.model_selection import cross_val_score
```

```
[15] model=KNeighborsClassifier(n_neighbors=10,metric='euclidean')
```

```
[16] #XX,Xy,YY,Yx=train_test_split(X,y,test_size=0.5)
     X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2)
     print('Train set:',X_train.shape)
     print('Test set:',X_test.shape)
```

```
Train set: (11141, 18000)
Test set: (2786, 18000)
```

```
[17] model.fit(X_train,y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='euclidean',
                     metric_params=None, n_jobs=None, n_neighbors=10, p=2,
                     weights='uniform')
```

```
model.score(X_test,y_test)
```

```
0.416367552045944
```

Le_classificateur_KNN

- Maintenant on va utiliser la cross validation qui consiste à entrainer puis valider notre modèle sur plusieurs découpe possible du train:

```
Test set: (2780, 18000)
```

```
model.fit(X_train,y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='euclidean',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

```
[ ] X_train,X_val,y_train,y_val=train_test_split(X_train,y_train,test_size=0.1)
```

```
[ ] print(cross_val_score(model,X_train,y_train,cv=5,scoring='accuracy'))
```

```
[0.41425723 0.44139651 0.43441397 0.41446384 0.43341646]
```

```
[ ] cross_val_score(model,X_train,y_train,cv=5,scoring='accuracy').mean()
```

```
0.4275896002764773
```

```
[ ] model.score(X_test,y_test)
```

```
0.4339554917444365
```

Le_classificateur_KNN

--***Resultat :***

Les metrics précision et recall de notre modèle avec un K=5 et la distance euclidean:

```
[ ] y_pred=model.predict(X_test)
```

```
[ ] print("sur le jeu de test:{:.3f}".format(metrics.accuracy_score(y_test,y_pred)))
```

```
sur le jeu de test:0.450
```

```
[ ] metrics.recall_score(y_test,y_pred,average='micro')
```

```
0.07927927927927927
```

```
[ ] metrics.precision_score(y_test,y_pred,average='micro')
```

```
0.6502463054187192
```

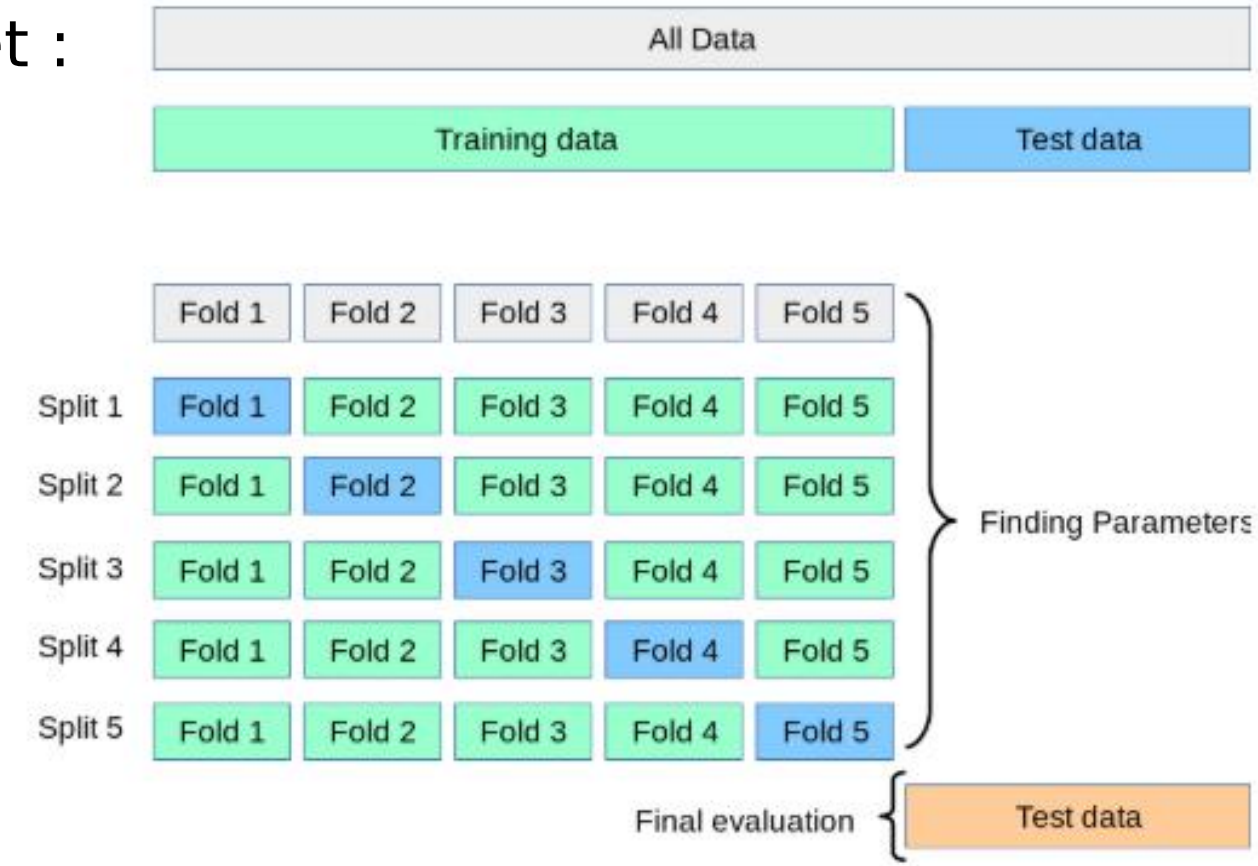
Le_classificateur_KNN

La cross validation avec un ***k=10*** et la distance euclidean:

```
[ ] X_train,X_val,y_train,y_val=train_test_split(X_train,y_train,test_size=0.1)

[ ] print(cross_val_score(model,X_train,y_train,cv=5,scoring='accuracy'))

    [0.3867036  0.39501385 0.40609418 0.40465632 0.40521064]

    cross_val_score(model,X_train,y_train,cv=5,scoring='accuracy').mean()

    0.3995357193310034
```

Le_classificateur_KNN

- En utilise la technique KFold du cross validation qui permet de découper notre train set :



Le_classificateur_KNN

Résultat :

```
[ ]  from sklearn.model_selection import KFold
     cv=KFold(n_splits=5)
```

```
[ ]  print(cross_val_score(model,X_train,y_train,cv=cv,scoring='accuracy'))
```

```
[0.41425723 0.44139651 0.43441397 0.41446384 0.43341646]
```

```
[ ]  cross_val_score(model,X_train,y_train,cv=cv,scoring='accuracy').mean()
```

```
0.422001029330959730.42200102933095973
```

```
[ ]  model.score(X_test,y_test)
```

```
0.4221105527638191
```

```
[ ]   y_pred=model.predict(X_test)
```

```
[ ]  print("sur le jeu de test:{:.3f}".format(metrics.accuracy_score(y_test,y_pred)))
```

```
sur le jeu de test:0.422
```

```
[ ]  metrics.recall_score(y_test,y_pred,average='micro')
```

```
0.05405405405405406
```

Le_classificateur_KNN

Résultat :

```
[ ] metrics.recall_score(y_test,y_pred,average='micro')
```

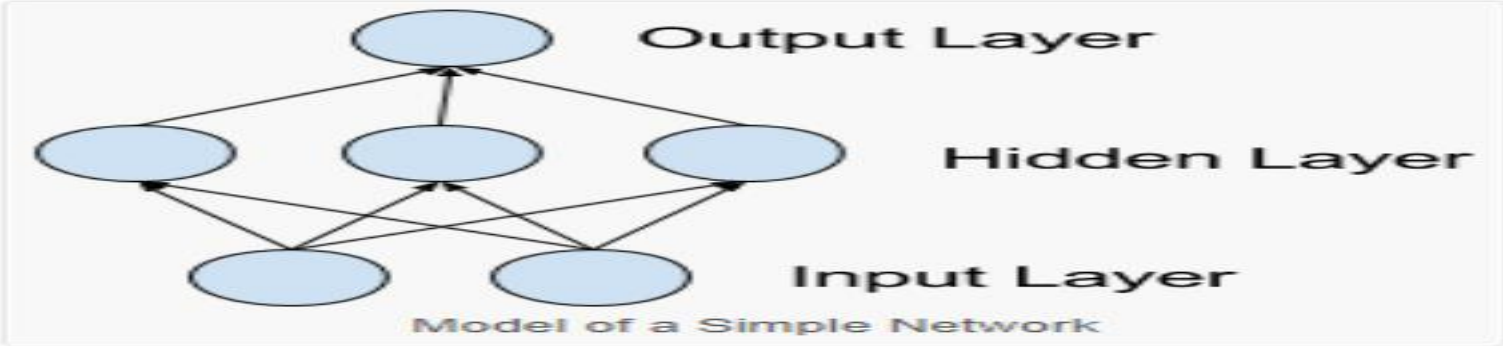
```
0.05405405405405406
```

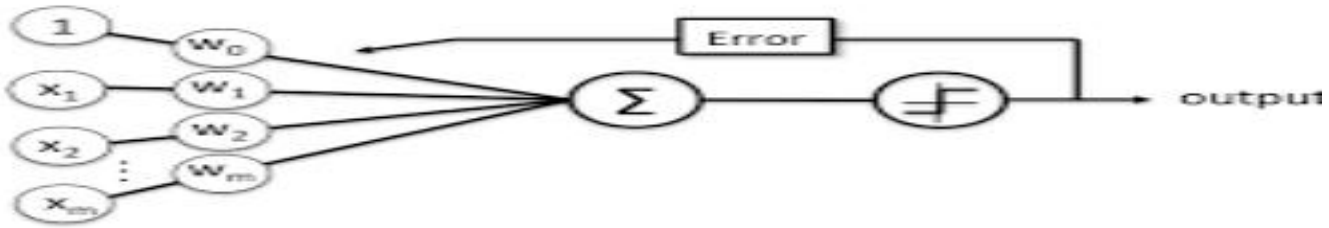
```
metrics.precision_score(y_test,y_pred,average='micro')
```

```
0.6133333333333333
```

Classificatoin avec DNN (réseau de norones profonds)

- Sont des perceptrons multicouche

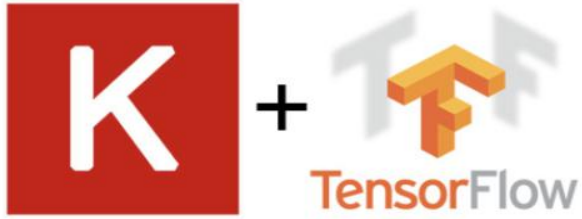




Schematic of a perceptron classifier.

Classificatoin avec DNN (réseau de norones profonds

- dans notre application nous avons travaillé avec TensorFlow ,en utilisant Keras API
- Il s'agit d'une API de haut niveau pour créer et former des modèles qui inclut les fonctionnalités spécifiques à TensorFlow, telles que l' **exécution rapide**



Classificatoin avec DNN (réseau de norones profonds

- ***Etape 1 :***
  - Pour construire notre model , on va utiliser Le type de modèle le plus courant est une pile de couches connu par **Sequential()**

Activation=*relu* pour tous les couches car elle n'accepte des valeurs négatives.

- ***Etape 2 :***
  - La configuration du modèle en utilisant la méthode **Compile()** qui va prendre en paramètre :
    - **Optimizer** =*SGD*
    - **Loss**= Binary _Crossentropy
    - **Metric** =Accuracy

Classificatoin avec DNN (réseau de norones profonds

- ***Etape 3 :***
  - Faire entrainer le modèle avec la méthode **fit()** en donnant de plus **les donnés** et leurs l**abel** :
    - **Epoch=60**
    - **Batch_size=100**
- *Résultat de cette étape :*

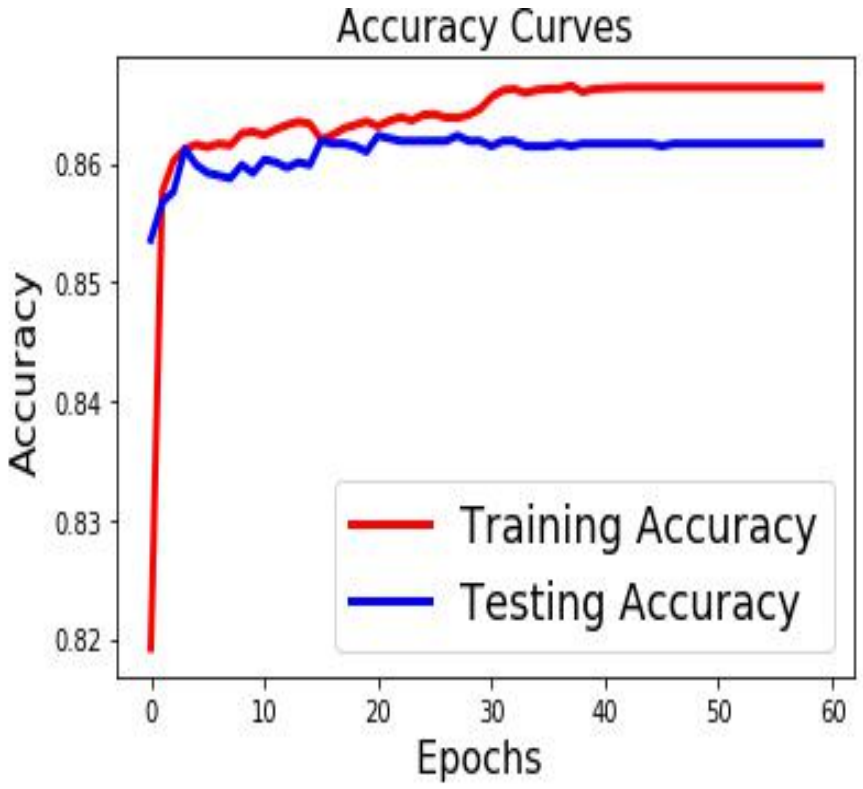
```
60
26 [==============================] - 3s 309us/sample - loss: 1.8567 - acc: 0.8664 - val_loss: 1.8946 - val_acc: 0.8617
```

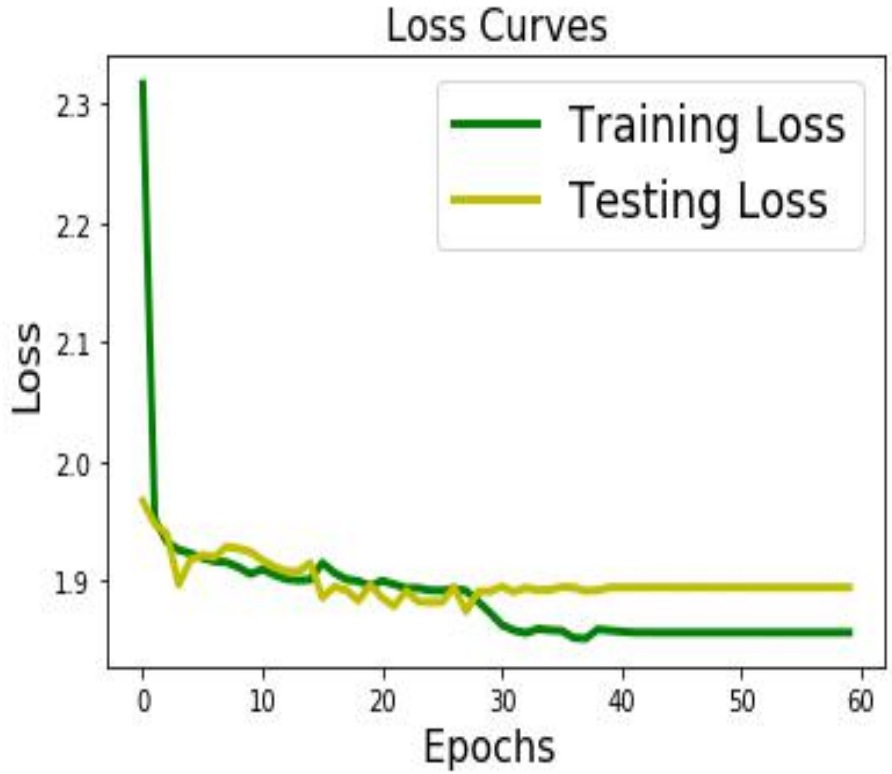
Classificatoin avec DNN (réseau de norones profonds

- **les résultats de l’évolution de notre modèle :** (sur les données de Test)

```
# Evaluate on test data
2786/2786 [==============================] - 0s 178us/sample - loss: 2.3724 - acc: 0.8462
test loss, test acc: [2.372430044374103, 0.8461953]
```

Classificatoin avec DNN (réseau de norones profonds





Classificatoin avec DNN (réseau de norones profonds

- Le Classificateur Perceptron multicouche (MLP) :
  - est un modèle qui se trouve dans la bibliothéque **Scikcit learn**
  - La classe MLPClassifier() prend comme paramatères principaux :
    - **cachet_layer_sizes *tuple, longueur = n_layers - 2, par défaut = (100,)***
    - **activation *{'identité', 'logistique', 'tanh', 'relu'}, défaut = 'relu'***
    - Fonction d'activation de la couche cachée.
    - **solveur *{ 'sgd', 'adam'}, default = 'adam'***
    - Le solveur pour l'optimisation du poids.

Etude comparative entre KNN et MLP (DNN)

vPrecision : à quelle fréquence notre modèle est correcte .

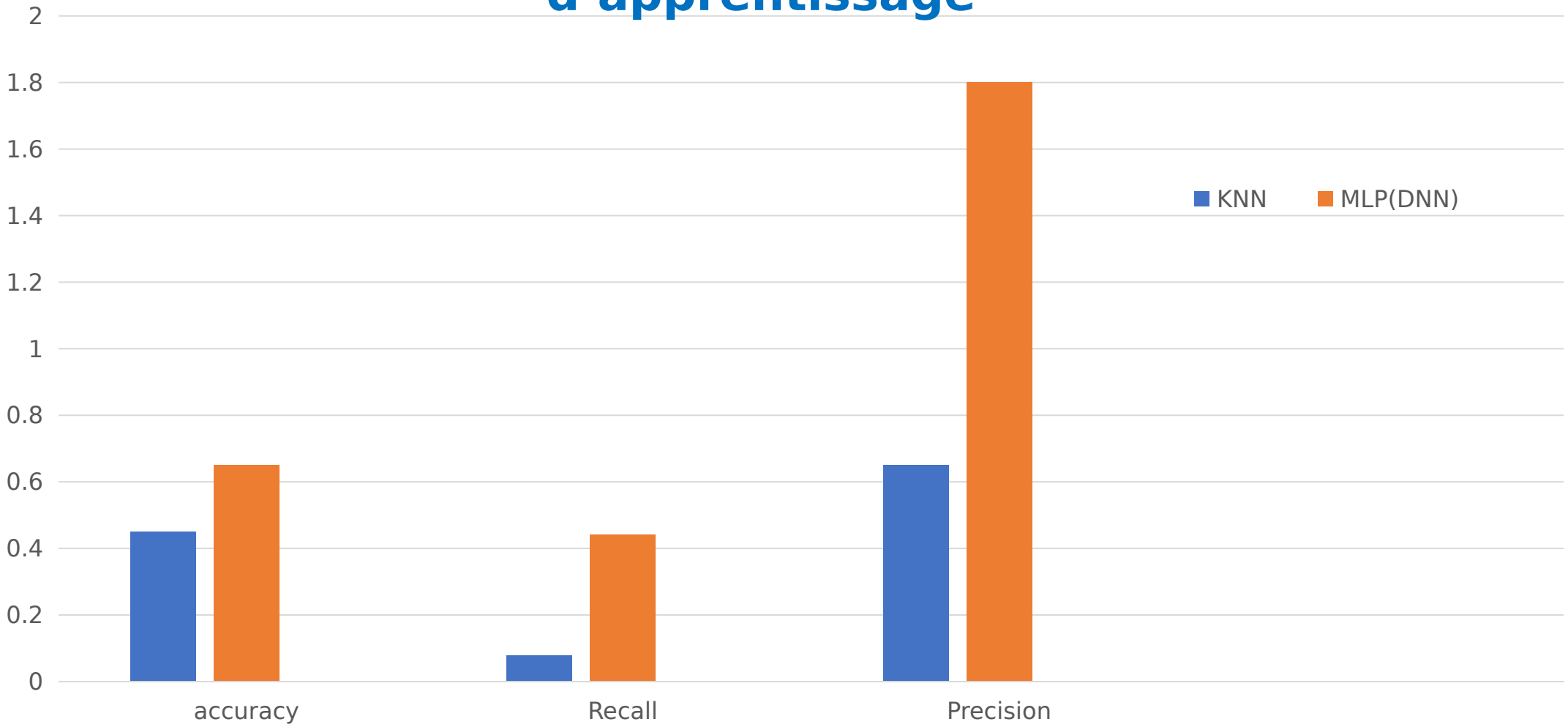
vAccuracy :

vRecall : donne une idée sur la capacité du modèle de prédire des résultat positive

Etude comparative entre KNN et MLP (DNN)

Metric	KNN	MLP(DNN)
accuracy_score	**0.45**	**0.654**
recall_score	**0.079**	**0.442**
precision_score	**0.65**	**0.549**

Etude Comparative du modèle pour deux algo d’apprentissage



Conclusion

- **Objectif :**
  - réaliser un modèle de détection des FA à base des signaux ECG.
  - Comparer entre deux algo d’apprentissage KNN /DNN .
- **Difficulté rencontrée:**
  - manipulation de la base de donnée
- **Résultat :**
  - Comme résultat nous avons pu réaliser un système de détection des FA avec une accuracy 84.61% à l’aide de l’algorithme DNN.
  - DNN est un algorithme ***plus performant*** que le KNN

Merci de votre visite

Travail fait par l'équipe d'ingenieurs de l'INPT:

EL-YAHYAOUI Abdenasser & ELKHATTARI Rabha

Deuxième année Cybersécurité et Confiance Numérique

Dans le cadre de la formation d'ingenieur à l'INPT sous le thème de “Machine et Deep Learning”