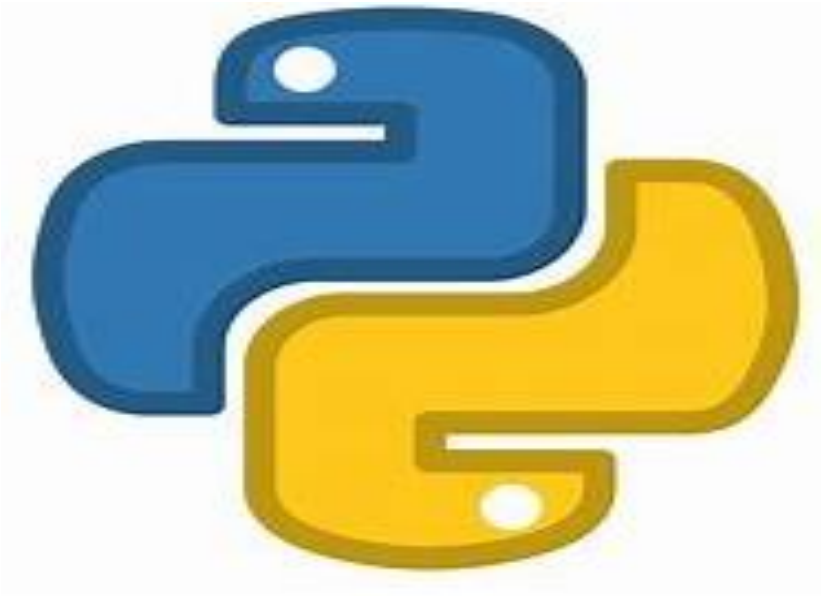
# Rapport de projet



**réaliser par** **HADDAD FATEH**

**rapport détailler sur mon travaille dans le projet de python :**

Jai commencer par importer les bibliothèque que on a vu en cour il s'agit de pandas , numpy et matplotlib pour travailler avec des données tabulaires, généralement stockées sous forme de fichiers CSV , tracer des graphiques, générer des nombres aléatoires, effectuer des opérations mathématiques et manipuler des tableaux multidimensionnels.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

## Etape 1 :

j'ai charger les 2 fichier csv mobile_train.csv,mobile_test_data.csv stocké dans data_train et X_test

```
# Chargement des données pour train_data et x_test
data_train = pd.read_csv('mobile_train.csv')
X_test = pd.read_csv('mobile_test_data.csv')
```

## Etape 2 : séparation des données

ou je divise les données de data_train en deux parties : df_train et df_valid. La première partie, df_train, contient 80% des données de data_train, choisies de manière aléatoire à l'aide de la fonction sample() de pandas. La seconde partie, df_valid, contient les données restantes qui n'ont pas été incluses dans df_train, et est obtenue en supprimant les indices de df_train de data_train à l'aide de la fonction drop(), Cette séparation est utile pour l'entraînement et la validation du modèle. ainsi jai separer les données en enlevant la dernière colonne de chaque DataFrame pour obtenir les valeurs d'entrée (X) et la dernière colonne pour obtenir les valeurs cibles (Y). X_train contient les valeurs d'entrée de l'ensemble d'entraînement ; Y_train contient les valeurs (de classe) de l'ensemble d'entraînement ; X_valid contient les valeurs d'entrée de l'ensemble de validation ; Y_valid contient les valeurs (de classe) de l'ensemble de validation

```
# Séparation des données

df_train = data_train.sample(frac=0.8, random_state=64)
df_valid = data_train.drop(df_train.index)


X_train = df_train.iloc[:, :-1].values
Y_train = df_train.iloc[:, -1].values
X_valid = df_valid.iloc[:, :-1].values
Y_valid = df_valid.iloc[:, -1].values
```

puis je transforme le dataframe X_test en un tableau numpy.

```
X_test = X_test.values
```

Etape 3 :**implementation du k plus proche voisin :**

j'ai travailler avec 2 méthode une que j'ai exécuter et l'autre je lai mis en commentaire les 2 méthode se ressemble ou je me suis inspirer des travaux d'ériger que on as fait on tp et td

**3.1** j'ai commencer par définir un fonction euclidean_distancequi calcule la distance euclidien entre 2 vecteur du dataframe

```
def euclidean_distance(v1, v2):
    return np.sqrt(np.sum(np.square(v1 - v2)))
```

**3.2** ensuite j'ai défini un fonction neighbors :

qui calcule les k voisins les plus proches du point de test x_test parmi les points d'entraînement X_train. Elle calcule d'abord la distance euclidienne entre le point de test x_test et chaque point d'entraînement dans X_train à l'aide de la fonction euclidean_distance. Ensuite, elle stocke la distance calculée dans une liste. Elle crée ensuite un DataFrame avec deux colonnes: la première colonne contient les étiquettes de classe y_label correspondantes à chaque point dans X_train, et la deuxième colonne contient les distances correspondantes stockées dans la liste. Elle trie ensuite le DataFrame en fonction des distances et retourne les

k premiers voisins les plus proches, triés par ordre croissant de distance.

```
def neighbors(X_train, y_label, x_test, k):
    list_distances = []
    for i in range(X_train.shape[0]):
        la_distence = euclidean_distance(X_train[i], x_test)
        list_distances.append(la_distence)
    df = pd.DataFrame()
    df["label"] = y_label
    df["distance"] = list_distances
    df = df.sort_values(by="distance")
    return df.iloc[:k, :]
```

**3.3** j'ai définis une fonction  prediction :

qui prend en entrée un dataframe contenant les k voisins les plus proches d'un exemple de test et retourne la classe prédite Pour ce faire, elle compte le nombre d'occurrences de chaque classe dans le dataframe neighbors en utilisant la méthode value_counts() de Pandas, puis elle retourne la classe qui a le plus grand nombre d'occurrences en utilisant la méthode index[0].

```
def prediction(neighbors):
    counts = neighbors['label'].value_counts()
    return counts.index[0]
```

**3.4** cest  la function devalution : Cette fonction évalue les performances du modèle kNN en utilisant l'ensemble de validation (X_valid, Y_valid) et les données d'apprentissage (X_train, Y_train). La fonction prend également en compte le nombre de voisins k utilisé pour effectuer la classification. si l'argument "verbose" est vrai, la fonction affiche des informations

supplémentaires, telles que les prédictions fausses et la précision finale.

```
def evaluation(X_train, Y_train, X_valid, Y_valid, k, verbose=True):
    vrai = 0  # resultat correct (juste)
    faux = 0  # resultat       faux
    liste_preds = []
    compteur = 0
    for i in range(X_valid.shape[0]):
        nearest_neighbors = neighbors(X_train, Y_train, X_valid[i], k)
        mon_predict = prediction(nearest_neighbors)
        liste_preds.append(mon_predict)
        if mon_predict == Y_valid[i]:
            vrai += 1
        else:
            faux += 1
            if verbose:
                print("prediction fausse: ")
                print("->> la Prediction : ->>", prediction(nearest_neighbors))
                print("le resultat que normalement on doit trouver cest  : ", Y_valid[i])
        compteur += 1
    accuracy = vrai / compteur
    if verbose:
        print("-->accuracy:-->", accuracy)
    return accuracy
```

**3.5** fonction test : cette function prend en entrée des données d'apprentissage "X_train" et des étiquettes d'apprentissage "Y_train", des données de test "X_test" et un paramètre "k". Elle renvoie une liste des prédictions pour chaque exemple de test. Pour chaque exemple de test, la fonction utilise la fonction "neighbors" pour trouver les k voisins les plus proches dans les données d'apprentissage, puis utilise la fonction "prediction" pour prédire la classe de l'exemple de test en fonction des classes des k voisins les plus proches. La fonction affiche également chaque prédiction faite pour les exemples de test.

```
def test(X_train, Y_train, X_test, k):
    liste_predictions = []
    for i in range(X_test.shape[0]):
        nearest_neighbors = neighbors(X_train, Y_train, X_test[i], k)
        pred = prediction(nearest_neighbors)
        liste_predictions.append(pred)
        print("-->Prediction -->: ", pred, "<--")
    return liste_predictions
```

**Etape** de création d'un fichier résultât pour tester sur le site de chalenge

prend les données de test X_test et utilise la fonction test pour faire des prédictions pour chaque observation dans X_test. Les prédictions sont stockées dans une liste appelée liste_predictions. Ensuite, cette liste est convertie en un tableau numpy et sauvegardée dans un fichier CSV appelé mobile_test_predictions.csv. En fin de compte, ce fichier CSV contiendra

les prédictions pour les données de test.(avec 400 elements)

```
# Entraînement du modèle
liste_predictions = test(X_train, Y_train, X_test, 10)
# Sauvegarde des prédictions dans un fichier CSV
A = np.array(liste_predictions)
np.savetxt('mobile_test_predictions.csv', A)
```

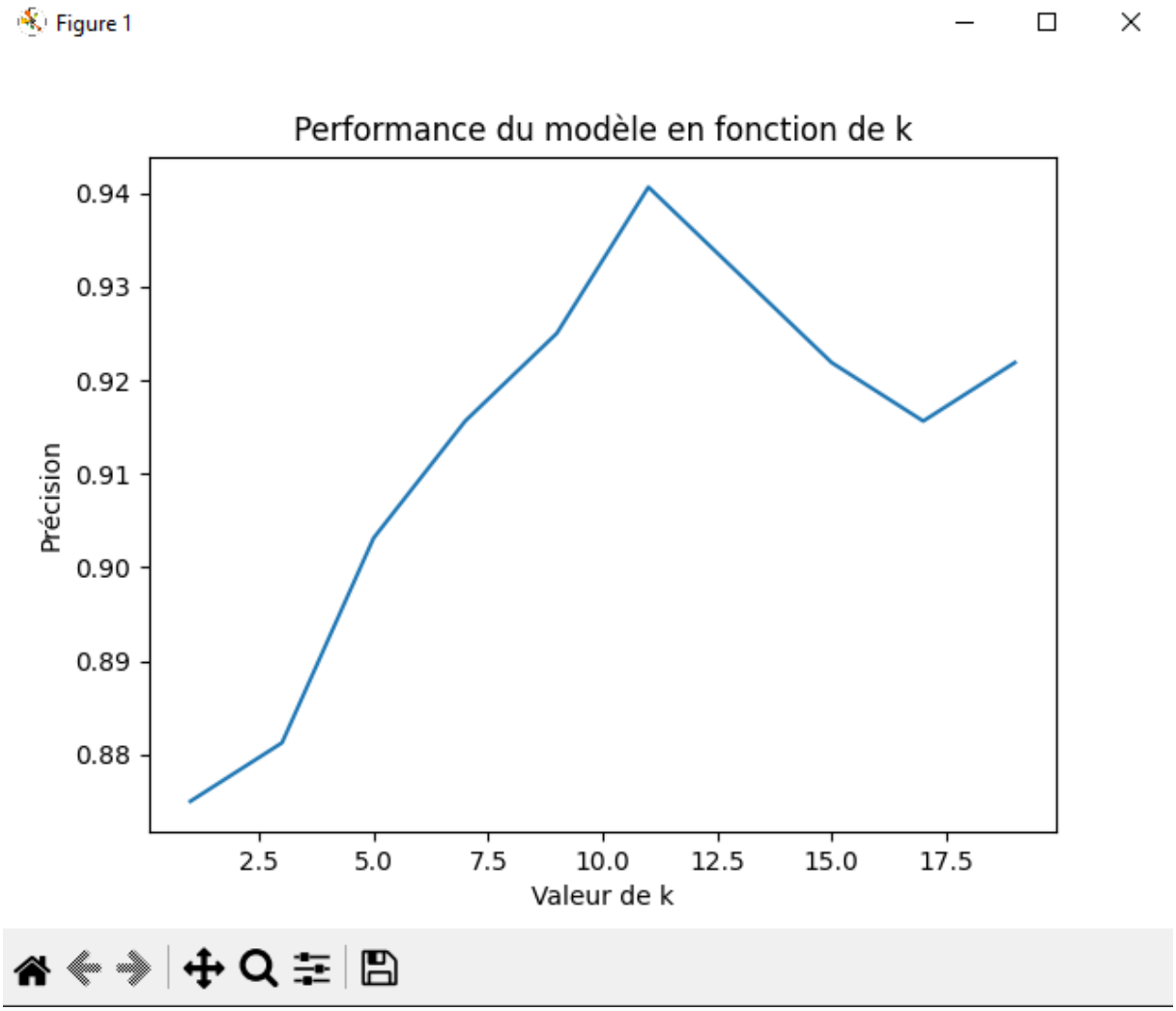
au final j'ai fini avec une étape d'analyse de performance du modèle de classification KNN en fonction de la valeur de K ,k_values est une liste de nombres impairs allant de 1 à 19 qui représentent les valeurs de k pour lesquelles on veut évaluer la performance du modèle. accuracy_values est une liste vide qui sera remplie avec les précisions calculées pour chaque valeur de k.La boucle for parcourt chaque valeur de k, appelle la fonction evaluation avec X_train, Y_train, X_valid, Y_valid et k, et enregistre la précision dans accuracy_values Une fois toutes les précisions calculées, le code affiche un graphique avec l'axe des abscisses représentant les valeurs de k et l'axe des ordonnées représentant la précision du modèle pour chaque valeur de k.

```
# Recherche du meilleur k par validation croisée
k_values = range(1, 20, 2)
accuracy_values = []

for k in k_values:
    accuracy = evaluation(X_train, Y_train, X_valid, Y_valid, k, verbose=False)
    accuracy_values.append(accuracy)

plt.plot(k_values, accuracy_values)
plt.xlabel('Valeur de k')
plt.ylabel('Précision')
plt.title('Performance du modèle en fonction de k')
plt.show()
```

Le résultat




**K=10 semble le meilleure valeur que k peut prendre pour avoir le meilleur Scor possible avec un score de 0.95 sur le site de chalenge**

**Remarque**

**Dans mon approche je nais pas utiliser la normalisation vue que dans ce cas la méthode de normalisation ne m'aide pas d'voir le score max et le but du projet c'est d'avoir le meilleur score possible .**

## Partie 2 réseau de neurones

Etape 1 Importation des bibliothèques : Les bibliothèques nécessaires sont importées, notamment pandas pour la manipulation des données, numpy pour la manipulation des tableaux numériques, matplotlib pour la visualisation des graphiques et différentes classes de modèles d'apprentissage automatique de scikit-learn pour l'entraînement et l'évaluation des modèles.

## Etape 2 : chargement des données comme dans la première partie exacte

2.1 Chargement des données : Les données d'entraînement sont chargées à partir d'un fichier CSV ('mobile_train.csv') dans un objet DataFrame appelé "data_train". Les données de test sont également chargées à partir d'un fichier CSV ('mobile_test_data.csv') dans un objet DataFrame appelé "X_test".

2.2 Séparation des données : Les données d'entraînement sont divisées en un ensemble d'entraînement (80% des données) et un ensemble de validation (20% des données) à l'aide de la méthode "sample" de pandas pour effectuer un échantillonnage aléatoire avec une graine aléatoire fixe (random_state=64). Les fonctionnalités (X) et les étiquettes (Y) pour les ensembles d'entraînement et de validation sont extraits sous forme de tableaux NumPy ("X_train", "Y_train", "X_valid", "Y_valid").

```
# # Chargement des données pour train_data et X_test
data_train = pd.read_csv('mobile_train.csv')
X_test = pd.read_csv('mobile_test_data.csv')
#
# Séparation des données
df_train = data_train.sample(frac=0.8, random_state=64)
df_valid = data_train.drop(df_train.index)

X_train = df_train.iloc[:, :-1].values
Y_train = df_train.iloc[:, -1].values
X_valid = df_valid.iloc[:, :-1].values
Y_valid = df_valid.iloc[:, -1].values
```

**Etape 3 :**Définition de la fonction d'évaluation : Une fonction "evaluate_knn_model" est définie pour évaluer les performances d'un modèle K plus proches voisins en fonction du nombre de voisins (k) en utilisant la validation croisée. La fonction prend en entrée les données d'entraînement et de validation, ainsi qu'un tableau de valeurs de k à tester. Elle entraîne le modèle K plus proches voisins pour chaque valeur de k, prédit les étiquettes pour l'ensemble de validation, calcule la précision et stocke les précisions dans une liste ("accuracies").

```
# Fonction d'évaluation pour le modèle K plus proches voisins
1 usage
def evaluate_knn_model(X_train, Y_train, X_valid, Y_valid, n_neighbors):
    accuracies = []  # Liste pour stocker les précisions pour chaque nombre de voisins

    for n in n_neighbors:
        clf = KNeighborsClassifier(n_neighbors=n)
        clf.fit(X_train, Y_train)
        Y_pred = clf.predict(X_valid)
        accuracy = accuracy_score(Y_valid, Y_pred)
        accuracies.append(accuracy)

    return accuracies
```

## Etape 4 :

**4.1** Évaluation du modèle K plus proches voisins : La fonction "evaluate_knn_model" est appelée avec les données d'entraînement et de validation, ainsi qu'un tableau de valeurs de k de 1 à 20. Les précisions sont calculées pour chaque valeur de k et stockées dans la liste "accuracies".

**4.2** Tracé du graphique de performance : Un graphique de performance est tracé en utilisant les valeurs de k sur l'axe des abscisses et les précisions sur l'axe des ordonnées à l'aide de la bibliothèque matplotlib. Le graphique montre comment la performance du modèle K plus proches voisins évolue en fonction du nombre de voisins.

```
# Paramètres pour le modèle K plus proches voisins
n_neighbors = np.arange(1, 21)  # Nombre de voisins à tester

# Appeler la fonction d'évaluation pour obtenir les précisions pour chaque nombre de voisins
accuracies = evaluate_knn_model(X_train, Y_train, X_valid, Y_valid, n_neighbors)


# Tracer le graphe de la performance en fonction du nombre de voisins
plt.plot(n_neighbors, accuracies)
plt.xlabel('Nombre de voisins')
plt.ylabel('Précision')
plt.title('Performance du modèle K plus proches voisins')
plt.grid(True)
plt.show()
```

## Etape 5 : c'est pas une étape c'est juste que Jai essayer d'appliquer la fonction dévaluation  donner dans l'énoncer du  tp 3 pour voir la fonction qui donne le meilleur résultat possible

**5.1** La classe MLPClassifier est importée de la bibliothèque scikit-learn avec "from sklearn.neural_network import MLPClassifier".

**5.2** Un objet "clf" est créé en instanciant la classe MLPClassifier avec des paramètres spécifiques, tels que le solveur "adam", la taille de la couche cachée "(350,)", l'alpha "1e-04" pour la régularisation L2, et la méthode d'apprentissage "adaptive" pour la mise à jour du taux d'apprentissage.

**5.3** Le modèle est entraîné sur les données d'entraînement à l'aide de la méthode "fit" de l'objet "clf" en utilisant les données d'entraînement "X_train" et les étiquettes d'entraînement "Y_train".

**5.4** La fonction "evaluation" est appelée avec les données d'entraînement (X_train, Y_train), les données de validation (X_valid, Y_valid), le nombre de voisins "K", l'objet du modèle MLPClassifier "clf", et le paramètre "verbose" défini sur "True" pour afficher les résultats.

**5.5** À l'intérieur de la fonction "evaluation", les variables TP (True Positive) et FP (False Positive) sont initialisées à 0 pour compter les prédictions correctes et incorrectes.

**5.6** Si le paramètre "nn" est défini sur "True", les prédictions du modèle MLPClassifier sont obtenues à l'aide de la méthode "predict" de l'objet "clf" sur les données de validation "X_valid", et les prédictions sont comparées avec les étiquettes de validation "Y_valid". Si les prédictions sont incorrectes, les résultats attendus et les prédictions sont affichés à l'aide de la fonction "print".

**5.7** Si le paramètre "nn" est défini sur "False", la fonction "predict" est appelée sur l'objet du modèle MLPClassifier "nn" pour obtenir les prédictions sur les données de validation "X_valid[i].reshape(1, -1)" pour chaque échantillon de validation.

**5.8** L'exactitude (accuracy) est calculée comme le rapport entre le nombre de prédictions correctes (TP) et le total des prédictions (total).

**5.9** Si le paramètre "verbose" est défini sur "True", l'exactitude est affichée à l'aide de la fonction "print".

**5.10** Enfin, l'exactitude est retournée comme résultat de la fonction "evaluation".

**5.11** Note : Il est important de noter que la fonction "evaluation" utilise l'objet du modèle MLPClassifier "clf" pour obtenir les prédictions si le paramètre "nn" est défini sur "True", et utilise l'objet du modèle MLPClassifier "nn" si le paramètre "nn" est défini sur "False". Assurez-vous que les objets du modèle sont correctement initialisés et entraînés avant d'appeler la fonction "evaluation".

```python
from sklearn.neural_network import MLPClassifier

# Initialize the MLPClassifier with appropriate parameters
clf = MLPClassifier(solver='adam', hidden_layer_sizes=(350,), alpha=1e-04, learning_rate='adaptive')

# Fit the classifier to the training data
clf.fit(X_train, Y_train)
```

```python
def evaluation(X_train, Y_train, X_valid, Y_valid, k, nn, verbose=True):
    TP = 0 # True Positive
    FP = 0 # False Positive
    total = 0

    if nn == True:
        pred = clf.predict(X_valid)
        for i in range(X_valid.shape[0]):
            if pred[i] == Y_valid[i]:
                TP += 1
            else:
                print("Prediction : ", pred[i])
                print("Resultat attendu : ", Y_valid[i])
            total += 1
    else:
        for i in range(X_valid.shape[0]):
            # Define the prediction method for nn
            prediction = nn.predict(X_valid[i].reshape(1, -1))
            if prediction == Y_valid[i]:
                TP += 1
            else:
                FP += 1
                print("Prediction : ", prediction)
```

```python
            total += 1
    accuracy = (TP) / total
    if verbose:
        print("Accuracy:" + str(accuracy))
    return accuracy
K=10
# Call the evaluation function
evaluation(X_train, Y_train, X_valid, Y_valid, K, clf, verbose=True)
```

## etape 6 entrainement du model

**6.1** Les données de test sont chargées à partir d'un fichier CSV appelé "mobile_test_data.csv" à l'aide de la fonction "pd.read_csv" de la bibliothèque

pandas. Les données de test sont généralement utilisées pour évaluer la performance du modèle sur des données inconnues et non utilisées pendant l'entraînement.

**6.2** Les prédictions sont faites sur les données de test en utilisant la méthode "predict" de l'objet du modèle MLPClassifier "clf". Les prédictions sont stockées dans la variable "pred".

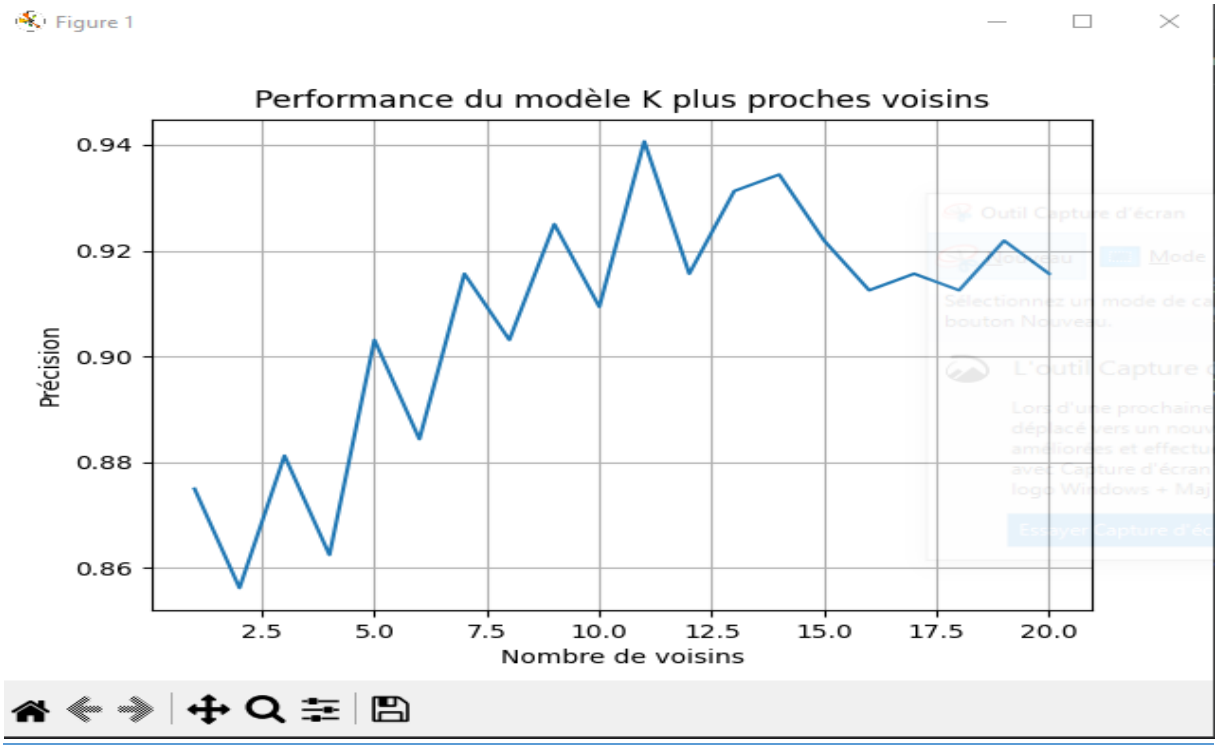
**6.3** Les prédictions sont enregistrées dans un fichier CSV appelé "mobile_test_predictions.csv" en utilisant la fonction "np.savetxt" de la bibliothèque NumPy. Le fichier CSV contiendra les valeurs prédites pour les échantillons de test, séparées par des virgules.

```
# Entraînement du modèle
# Load the test data
mobile_test_data = pd.read_csv("mobile_test_data.csv")

# Make predictions on the test data
pred = clf.predict(mobile_test_data)

# Save the predictions to a CSV file
np.savetxt('mobile_test_predictions.csv', pred, delimiter=',')
```

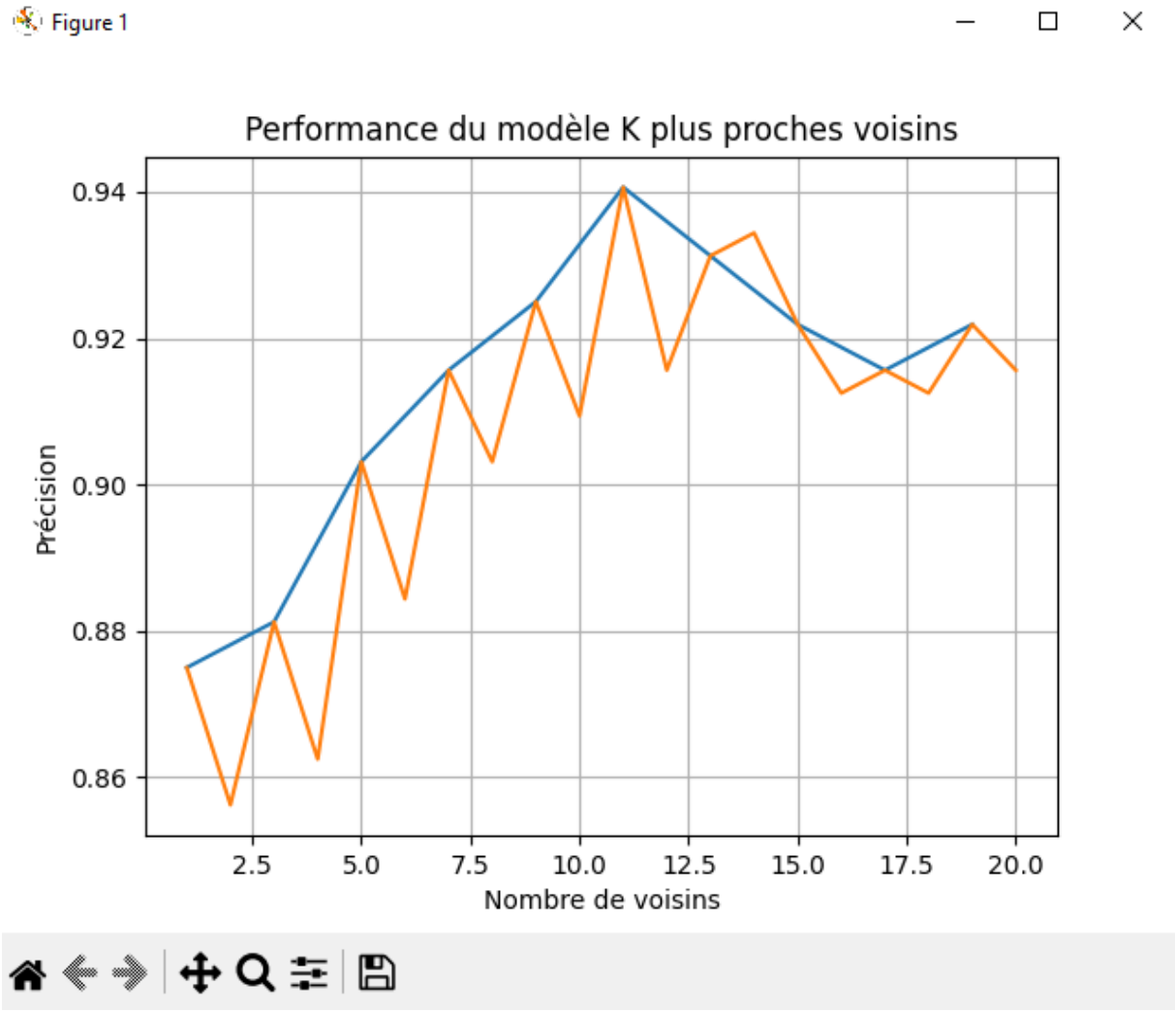
## Le resultat :



**On remarque que k=10 et la meilleur valeur possible pour K plus proche voisin avec un score de 0.94125**

**J'ai remarqué que pour la 2eme partie avec l'analyse avec réseau neurone mon résultat final dans la soumission pour le chalenge na pas augmenter vraiment il donne presque le même score avec le knn**

**Le representation des 2 graphe avec knn et reseau neurone**




**C'est ce que justifie ma remarque d'avant ou le score de mes 2 méthodes d'analyse avec knn et réseau neurone donne presque le même score avec le même k meilleur valeur possible pour K**

**Sur le site du chalenge je suis classé 16 éme avec un score de 0.95**

**Le meiileur score cest 0.96.**