Université Paul Sabatier
Toulouse

Compte Rendu

Introduction à l'apprentissage automatique

M1 EEA

- Module KEAX7IK1 : Introduction à l'apprentissage automatique
- Réalisé par :
  - KABOU Abdeldjalil

2024 / 2025

TP : Mise en œuvre de méthodes de classification

I. Introduction

Dans cette étude, nous allons implémenter et évaluer différentes méthodes de classification à l'aide de la bibliothèque scikit-learn (sklearn) sur des données prétraitées par l'Analyse Factorielle Discriminante (AFD) et l'Analyse en Composantes Principales (ACP). Les techniques de classification que nous explorerons comprennent le K-Means, le Perceptron Multicouche (MLP), les Machines à Vecteurs de Support (SVM), ainsi que la Forêt d'Arbres Décisionnels. L'objectif est d'évaluer les performances de chaque méthode sur ces données et de comparer leur efficacité pour la classification des individus.

II. Travail à effectuer

Pour évaluer la capacité de généralisation de chaque méthode, nous procéderons à l'apprentissage sur un jeu de données généré lors de la première exécution de genere.py, puis nous testerons les performances sur un autre jeu de données obtenu à partir d'une seconde exécution de ce même script.

a. Avec l'AFD :

Desktop > TP > tp_2.py > ...

```python
import numpy as np
import matplotlib.pyplot as plt
import TPClassif2024 as tpc
import dernier_etape as dr

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.cluster import KMeans
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier

# Division des données
X_train, X_test, y_train, y_test = train_test_split(dr.X_AFD, tpc.NoClasse, test_size=0.2,random_state=None)
```

i. Algorithme des centres mobiles :

```python
# Initialiser et entraîner le modèle K-Means
kmeans = KMeans(n_clusters=4, random_state=42)
kmeans.fit(X_train,y_train)
y_predict_k = kmeans.predict(X_test)
conf_matrix_k = confusion_matrix(y_test,y_predict_k)
a_kmeans = accuracy_score(y_test, y_predict_k)
print(f"Précision (K-Means) :", a_kmeans )
```

ii. Perceptron multicouche :

```
# Initialisation du Perceptron Multicouche
Classif_p = MLPClassifier(hidden_layer_sizes=(100,),max_iter=500,random_state=42)
Classif_p.fit(X_train, y_train)
y_predict_p = Classif_p.predict(X_test)
conf_matrix_p = confusion_matrix(y_test,y_predict_p)
a_p = accuracy_score(y_test, y_predict_p)
print(f"Précision (MLPClassifier) :",a_p)
```

iii. Séparateurs `a Vastes Marges :

```
# Initialisation du classifieur SVM
Classif_M = SVC(kernel='rbf',C=1.0,random_state=42)
Classif_M.fit(X_train, y_train)
y_predict_m = Classif_M.predict(X_test)
conf_matrix_m = confusion_matrix(y_test,y_predict_m)
a_SVM = accuracy_score(y_test, y_predict_m)
print(f"Précision (SVM) :",a_SVM)
```

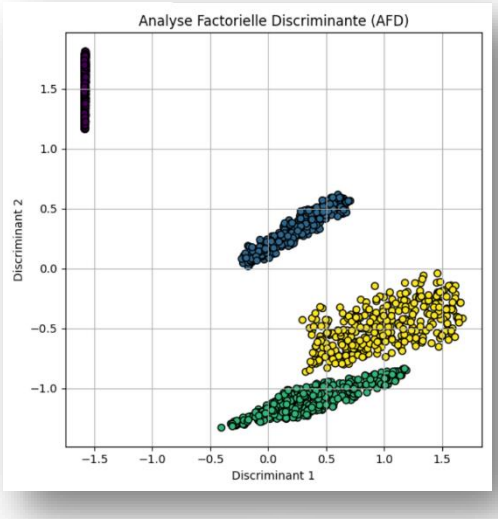
iv. Forêt d'arbres décisionnels :

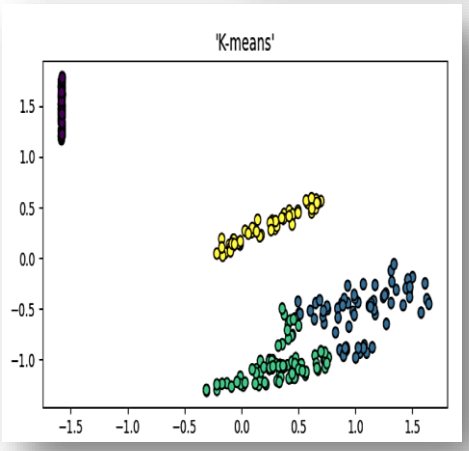
```
# Forêt d'arbres
Classif_F = RandomForestClassifier(n_estimators=100,random_state=42)
Classif_F.fit(X_train, y_train)
y_predict_f = Classif_F.predict(X_test)
conf_matrix_f = confusion_matrix(y_test,y_predict_f)
a_RF = accuracy_score(y_test, y_predict_f)
print(f"Précision (Random Forest) :",a_RF)
```

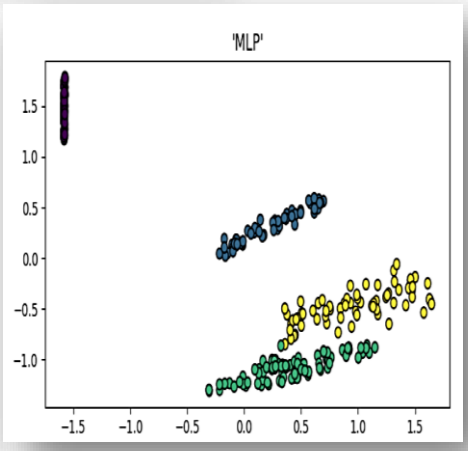
Les résultats obtenus :

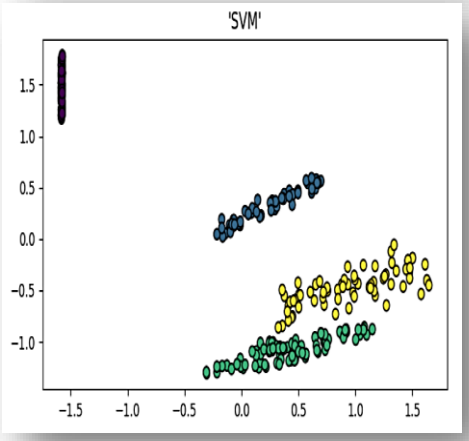
```
Précision (K-Means) : 0.3
Précision (MLPClassifier) : 1.0
Précision (SVM) : 1.0
Précision (Random Forest) : 0.9964285714285714
PS C:\Users\WAEL tech\Desktop>
```

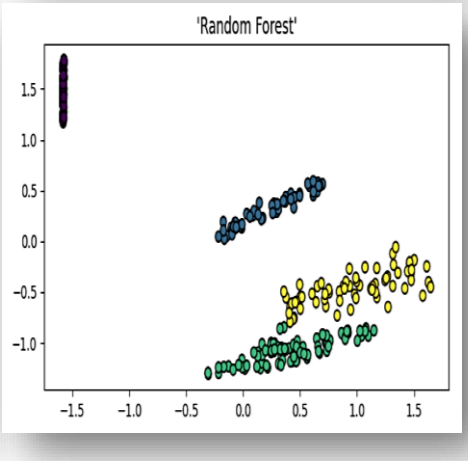
Affichage des résultats :



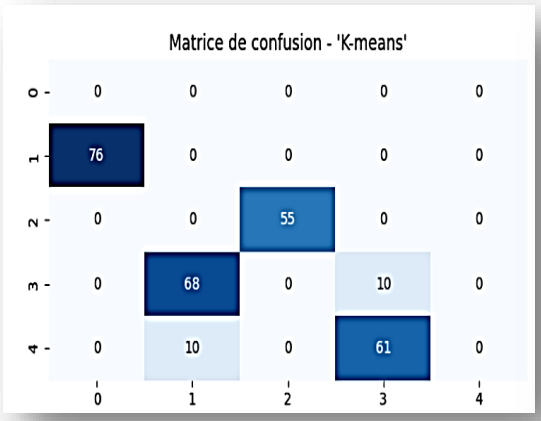


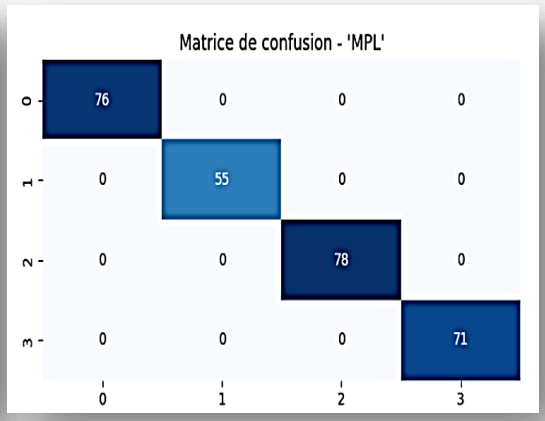


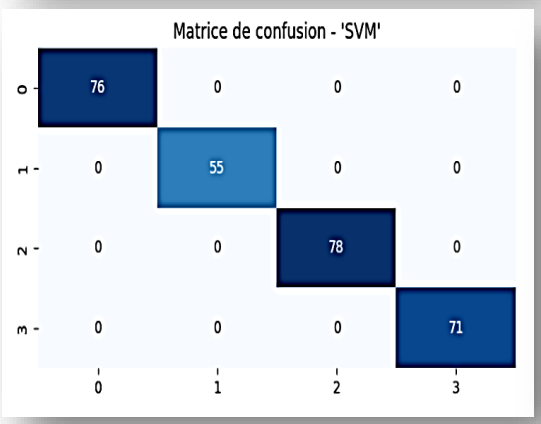


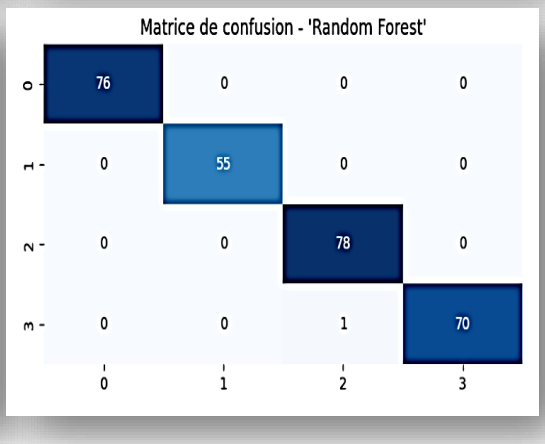


Affichage avec les matrices :









**Le code est ci-dessous dans les annexes*

Observation :

- Pour *K-Means :* La précision est de 0.3. Cette méthode est non supervisée, ce qui signifie qu'elle a du mal à bien séparer les différentes classes et n'est pas adaptée à la classification supervisée.
- Pour MLPClassifier : La précision est de 1.0. Le Perceptron Multicouche a très bien appris les relations dans les données et il est très bon pour prédire les résultats.
- Pour SVM : La précision est de 1.0. Le SVM a aussi bien séparé les classes et fonctionne très bien pour cette tâche.
- Pour Random Forest : La précision est de 0.996. Random Forest donne aussi de très bons résultats, presque aussi bons que ceux du MLP et du SVM, grâce à sa solidité.

b. *Avec l'ACP :*

```
Desktop > TP > tp_2_acp.py > ...
1   import numpy as np
2   import matplotlib.pyplot as plt
3   import seaborn as sns
4   import TPClassif2024 as tpc
5   import dernier_etape as dr
6
7   from sklearn.model_selection import train_test_split
8   from sklearn.metrics import accuracy_score, confusion_matrix
9   from sklearn.cluster import KMeans
10  from sklearn.neural_network import MLPClassifier
11  from sklearn.svm import SVC
12  from sklearn.ensemble import RandomForestClassifier
13
14  # Division des données
15  X_train, X_test, y_train, y_test = train_test_split(dr.X_ACP, tpc.NoClasse, test_size=0.2,random_state=None)
16
```

i. Algorithme des centres mobiles :

```
# Initialiser et entraîner le modèle K-Means
kmeans = KMeans(n_clusters=4, random_state=42)
kmeans.fit(X_train,y_train)
y_predict_k = kmeans.predict(X_test)
conf_matrix_k = confusion_matrix(y_test,y_predict_k)
a_kmeans = accuracy_score(y_test, y_predict_k)
print(f"Précision (K-Means) :", a_kmeans )
```

ii. Perceptron multicouche :

```
# Initialisation du Perceptron Multicouche
Classif_p = MLPClassifier(hidden_layer_sizes=(100,),max_iter=500,random_state=42)
Classif_p.fit(X_train, y_train)
y_predict_p = Classif_p.predict(X_test)
conf_matrix_p = confusion_matrix(y_test,y_predict_p)
a_p = accuracy_score(y_test, y_predict_p)
print(f"Précision (MLPClassifier) :",a_p)
```

iii. Séparateurs `a Vastes Marges :

```
# Initialisation du classifieur SVM
Classif_M = SVC(kernel='rbf',C=1.0,random_state=42)
Classif_M.fit(X_train, y_train)
y_predict_m = Classif_M.predict(X_test)
conf_matrix_m = confusion_matrix(y_test,y_predict_m)
a_SVM = accuracy_score(y_test, y_predict_m)
print(f"Précision (SVM) :",a_SVM)
```

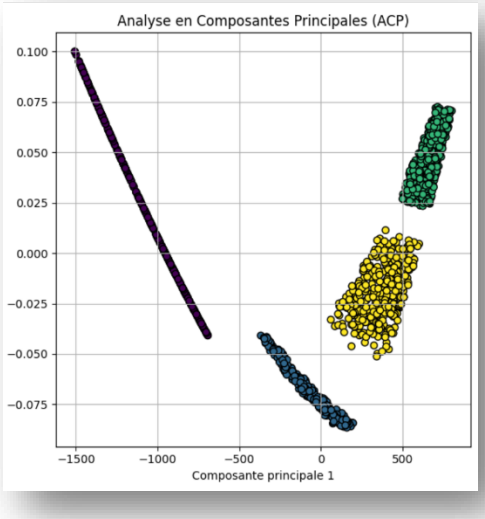
iv. Forêt d'arbres décisionnels :

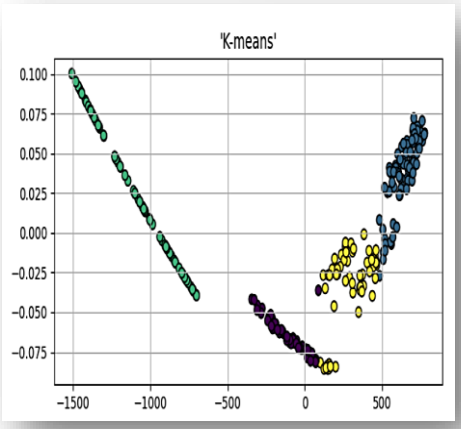
```
# Forêt d'arbres
Classif_F = RandomForestClassifier(n_estimators=100,random_state=42)
Classif_F.fit(X_train, y_train)
y_predict_f = Classif_F.predict(X_test)
conf_matrix_f = confusion_matrix(y_test,y_predict_f)
a_RF = accuracy_score(y_test, y_predict_f)
print(f"Précision (Random Forest) :",a_RF)
```

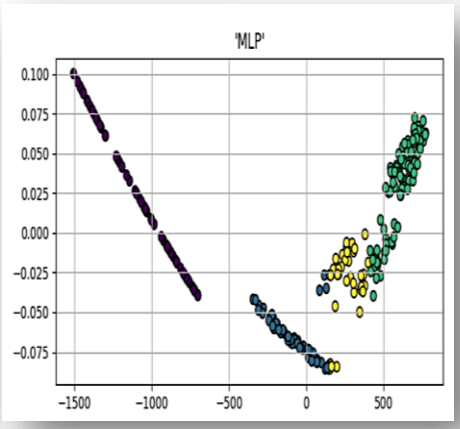
Les résultats obtenus :

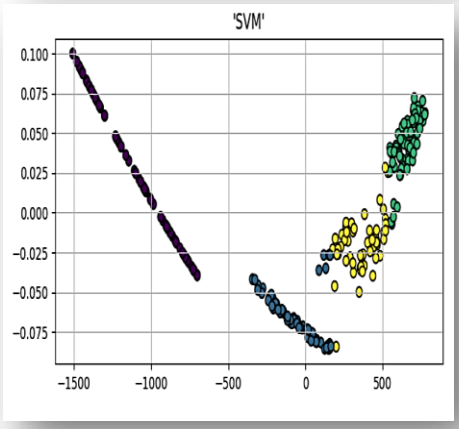
```
Précision (K-Means) : 0.0
Précision (MLPClassifier) : 0.8892857142857142
Précision (SVM) : 0.95
Précision (Random Forest) : 0.9964285714285714
PS C:\Users\WAEL tech\Desktop>
```

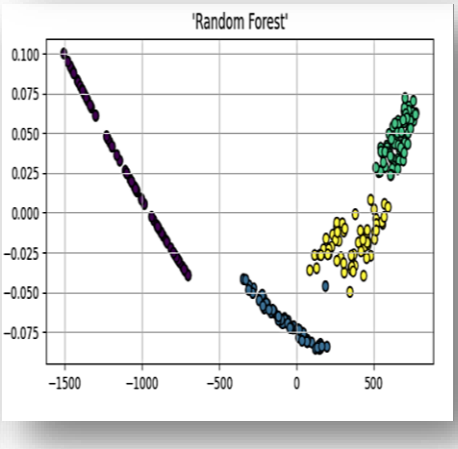
Affichage des résultats :



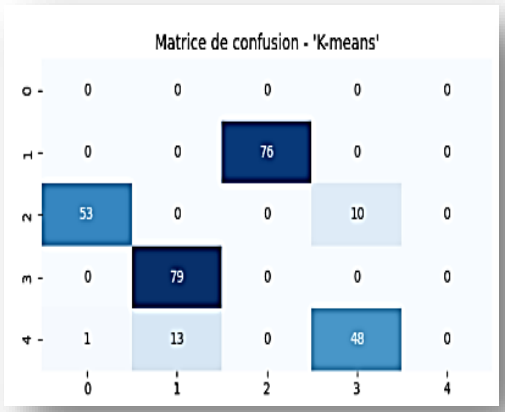


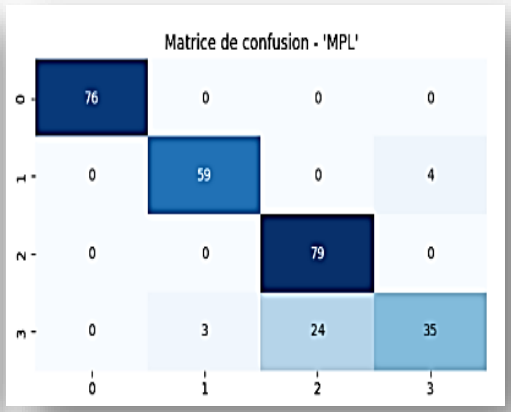


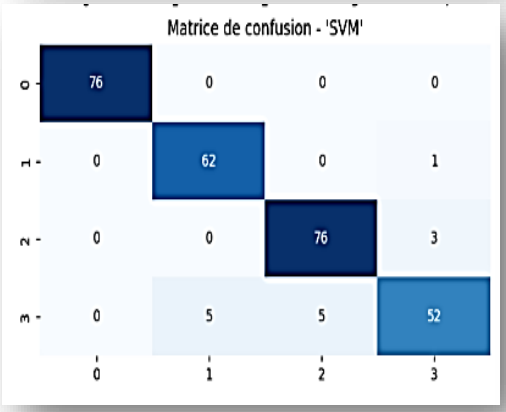


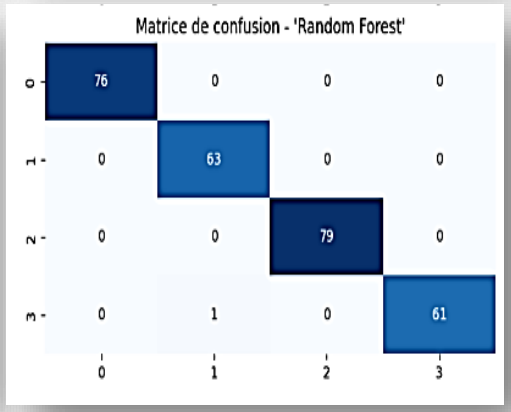


Affichage avec les matrices :









**Le code est ci-dessous dans les annexes*

Observation :

- Pour *K-Means :* Cette méthode affiche une précision nulle, suggérant des difficultés majeures à classifier les données réduites en composantes principales. Elle est moins adaptée àla structure des données obtenues par l'ACP.
- Pour MLPClassifier : Le MLP présente une précision de 0.88, indiquant des performances inférieures par rapport à l'AFD. Cela suggère que le MLP peut rencontrer des défis spécifiques lors de la classification des données en utilisant l'ACP.
- Pour SVM : La précision est de 0.95. Le SVM a également bien séparé les classes, avec une performance très proche de la perfection.
- Pour Random Forest Le RF maintient une précision parfaite de 0.99 sur l'ensemble de test, indiquant une excellente capacité à classifier les données en composantes principales.

.

III. Conclusion

En conclusion, l'Analyse Factorielle Discriminante (AFD) donne de meilleurs résultats que l'Analyse en Composantes Principales (ACP) pour les méthodes MLP et SVM. Toutefois, le Random Forest (RF) se distingue par des performances exceptionnelles dans les deux cas. Le choix entre l'AFD et l'ACP dépendra également de critères supplémentaires tels que la complexité des modèles, l'interprétabilité et les besoins spécifiques du problème à résoudre

Les avantages et inconvénients des différentes méthodes de classification :

- *Centres mobiles (K-Means) :*

Avantages : Simple à mettre en œuvre, Adapté aux ensembles de données volumineux

Inconvénients : Nécessite de spécifier le nombre de clusters à l'avance, Moins performant pour des données complexes ou mal séparées

- Perceptron Multicouche (MLP)

Avantages : Adapté aux problèmes complexes, Capable de modéliser des relations non linéaires

Inconvénients : Sensible au surapprentissage, Nécessite un réglage fin des hyperparamètres (nombre de couches, neurones, taux d'apprentissage)

- Machines à Vecteurs de Support (SVM)

Avantages : Efficace dans des espaces de grande dimension, Peut gérer des données non linéaires grâce à l'utilisation de noyaux

Inconvénients : Sensible au choix du noyau, Peut être coûteux en termes de calcul, surtout pour de grandes quantités de données,

- Forêt d'arbres décisionnels (Random Forest)

Avantages : Robuste, capable de gérer des données non linéaires, Moins sensible au surapprentissage grâce à l'ensemble d'arbres

Inconvénients : Peut-être complexe à interpréter, Peut devenir lent et difficile à interpréter avec un grand nombre d'arbres

Annexes

Avec l'AFD

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import TPClassif2024 as tpc
import dernier_etape as dr
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.cluster import KMeans
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier

# Division des données
X_train, X_test, y_train, y_test = train_test_split(dr.X_AFD, tpc.NoClasse,
test_size=0.2,random_state=None)

# Initialiser et entraîner le modèle K-Means
kmeans = KMeans(n_clusters=4, random_state=42)
kmeans.fit(X_train,y_train)
y_predict_k = kmeans.predict(X_test)
conf_matrix_k = confusion_matrix(y_test,y_predict_k)
a_kmeans = accuracy_score(y_test, y_predict_k)
print(f"Précision (K-Means) :", a_kmeans )

# Initialisation du Perceptron Multicouche
Classif_p = MLPClassifier(hidden_layer_sizes=(100,),max_iter=500,random_state=42)
Classif_p.fit(X_train, y_train)
y_predict_p = Classif_p.predict(X_test)
conf_matrix_p = confusion_matrix(y_test,y_predict_p)
accuracy = accuracy_score(y_test, y_predict_p)
print(f"Précision (MLPClassifier) :",accuracy)

# Initialisation du classifieur SVM
Classif_M = SVC(kernel='rbf',C=1.0,random_state=42)
Classif_M.fit(X_train, y_train)
y_predict_m = Classif_M.predict(X_test)
conf_matrix_m = confusion_matrix(y_test,y_predict_m)
a_SVM = accuracy_score(y_test, y_predict_m)
print(f"Précision (SVM) :",a_SVM)

Classif_F = RandomForestClassifier(n_estimators=100,random_state=42)
# Entraînement du modèle
Classif_F.fit(X_train, y_train)
```

```
y_predict_f = Classif_F.predict(X_test)
conf_matrix_f = confusion_matrix(y_test,y_predict_f)
accuracy = accuracy_score(y_test, y_predict_f)
print(f"Précision (Random Forest) :",accuracy)

plt.figure(figsize=(12, 10))
plt.scatter(X_train[:,0], X_train[:,1],c = y_train,cmap='viridis', edgecolors='k')
plt.title("Analyse Factorielle Discriminante (AFD)")
plt.xlabel("Discriminant 1");plt.ylabel("Discriminant 2");plt.grid()

plt.figure(figsize=(12, 10))
plt.subplot(2,2,1)
plt.scatter(X_test[:,0], X_test[:,1],c = y_predict_k,cmap='viridis', edgecolors='k')
plt.title("'K-means'")
plt.grid()
plt.subplot(2,2,2)
plt.scatter(X_test[:,0], X_test[:,1],c = y_predict_p,cmap='viridis', edgecolors='k')
plt.title("'MLP'")
plt.grid()
plt.subplot(2,2,3)
plt.scatter(X_test[:,0], X_test[:,1],c = y_predict_m,cmap='viridis', edgecolors='k')
plt.title("'SVM'")
plt.grid()
plt.subplot(2,2,4)
plt.scatter(X_test[:,0], X_test[:,1],c = y_predict_f,cmap='viridis', edgecolors='k')
plt.title("'Random Forest'")
plt.grid()
plt.show()

plt.figure(figsize=(15, 10))
plt.subplot(2,2,1)
sns.heatmap(conf_matrix_k, annot = True, fmt = 'd',cmap='Blues', cbar=False)
plt.title("Matrice de confusion - 'K-means'")
plt.subplot(2,2,2)
sns.heatmap(conf_matrix_p, annot = True, fmt = 'd',cmap='Blues', cbar=False)
plt.title("Matrice de confusion - 'MPL'")
plt.subplot(2,2,3)
sns.heatmap(conf_matrix_m, annot = True, fmt = 'd',cmap='Blues', cbar=False)
plt.title("Matrice de confusion - 'SVM'")
plt.subplot(2,2,4)
sns.heatmap(conf_matrix_f, annot = True, fmt = 'd',cmap='Blues', cbar=False)
plt.title("Matrice de confusion - 'Random Forest'")
plt.show()
```

Avec l'ACP

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import TPClassif2024 as tpc
import dernier_etape as dr

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.cluster import KMeans
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier

# Division des données
X_train, X_test, y_train, y_test = train_test_split(dr.X_ACP, tpc.NoClasse,
test_size=0.2,random_state=None)

# Initialiser et entraîner le modèle K-Means
kmeans = KMeans(n_clusters=4, random_state=42)
kmeans.fit(X_train,y_train)
y_predict_k = kmeans.predict(X_test)
conf_matrix_k = confusion_matrix(y_test,y_predict_k)
a_kmeans = accuracy_score(y_test, y_predict_k)
print(f"Précision (K-Means) :", a_kmeans )

# Initialisation du Perceptron Multicouche
Classif_p = MLPClassifier(hidden_layer_sizes=(100,),max_iter=500,random_state=42)
Classif_p.fit(X_train, y_train)
y_predict_p = Classif_p.predict(X_test)
conf_matrix_p = confusion_matrix(y_test,y_predict_p)
a_p = accuracy_score(y_test, y_predict_p)
print(f"Précision (MLPClassifier) :",a_p)

# Initialisation du classifieur SVM
Classif_M = SVC(kernel='rbf',C=1.0,random_state=42)
Classif_M.fit(X_train, y_train)
y_predict_m = Classif_M.predict(X_test)
conf_matrix_m = confusion_matrix(y_test,y_predict_m)
a_SVM = accuracy_score(y_test, y_predict_m)
print(f"Précision (SVM) :",a_SVM)

# Forêt d'arbres
Classif_F = RandomForestClassifier(n_estimators=100,random_state=42)
Classif_F.fit(X_train, y_train)
y_predict_f = Classif_F.predict(X_test)
conf_matrix_f = confusion_matrix(y_test,y_predict_f)
a_RF = accuracy_score(y_test, y_predict_f)
print(f"Précision (Random Forest) :",a_RF)
```

```
plt.figure(figsize=(12, 10))
plt.scatter(X_train[:,0], X_train[:,1],c = y_train,cmap='viridis', edgecolors='k')
plt.title("Analyse en Composantes Principales (ACP)")
plt.xlabel("Composante principale 1");plt.ylabel("Composante principale 2");plt.grid()

plt.figure(figsize=(12, 10))
plt.subplot(2,2,1)
plt.scatter(X_test[:,0], X_test[:,1],c = y_predict_k,cmap='viridis', edgecolors='k')
plt.title("'K-means'")
plt.grid()
plt.subplot(2,2,2)
plt.scatter(X_test[:,0], X_test[:,1],c = y_predict_p,cmap='viridis', edgecolors='k')
plt.title("'MLP'")
plt.grid()
plt.subplot(2,2,3)
plt.scatter(X_test[:,0], X_test[:,1],c = y_predict_m,cmap='viridis', edgecolors='k')
plt.title("'SVM'")
plt.grid()
plt.subplot(2,2,4)
plt.scatter(X_test[:,0], X_test[:,1],c = y_predict_f,cmap='viridis', edgecolors='k')
plt.title("'Random Forest'")
plt.grid()
plt.show()

plt.figure(figsize=(15, 10))
plt.subplot(2,2,1)
sns.heatmap(conf_matrix_k, annot = True, fmt = 'd',cmap='Blues', cbar=False)
plt.title("Matrice de confusion - 'K-means'")
plt.subplot(2,2,2)
sns.heatmap(conf_matrix_p, annot = True, fmt = 'd',cmap='Blues', cbar=False)
plt.title("Matrice de confusion - 'MPL'")
plt.subplot(2,2,3)
sns.heatmap(conf_matrix_m, annot = True, fmt = 'd',cmap='Blues', cbar=False)
plt.title("Matrice de confusion - 'SVM'")
plt.subplot(2,2,4)
sns.heatmap(conf_matrix_f, annot = True, fmt = 'd',cmap='Blues', cbar=False)
plt.title("Matrice de confusion - 'Random Forest'")
plt.show()
```