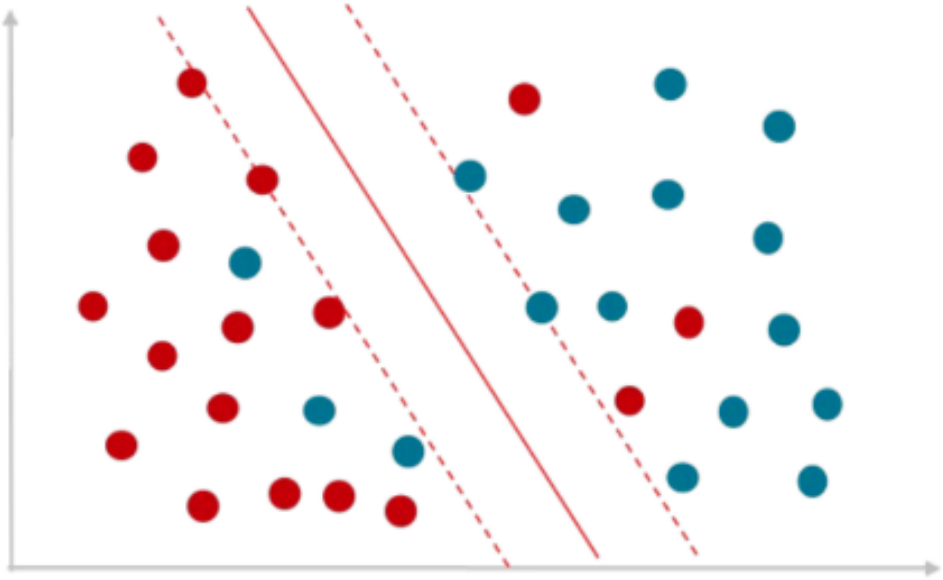
Université Abdelmalek Essaadi
Faculté des Sciences et Techniques de Tanger
Département Génie Informatique

Atelier 2 : «Classification»



Encadré par :
Prof. ELAACHAK LOTFI

Réalisé par :
BARBYCH Mohamed

Cycle d'ingenieur Logiciels et Systèmes Intelligents
S3 - 2024/2025

Session d'octobre 2024

Table des matières

1 Introduction

Cet atelier a pour objectif de pratiquer les concepts de classification à travers l'analyse d'un dataset sur le diabète. Nous utilisons des outils tels que Python, Pandas, Scikit-learn, et Matplotlib pour explorer, sélectionner les caractéristiques pertinentes et construire des modèles adaptés. Ce rapport documente toutes les étapes avec des explications, du code coloré, et des résultats sous forme de captures d'écran.

2 Partie 1 : Data Visualisation, Feature Selection et Normalisation

2.1 Importation des bibliothèques nécessaires

Pour ce projet, nous avons utilisé les bibliothèques suivantes :

- **pandas** : Pour manipuler et analyser les données tabulaires.
- **numpy** : Pour effectuer des calculs numériques efficaces.
- **matplotlib.pyplot** : Pour la visualisation des données sous forme de graphiques.
- **scikit-learn** : Une bibliothèque puissante utilisée pour les algorithmes de machine learning et les étapes de prétraitement, comme :
  - **train_test_split** : Pour diviser les données en ensembles d'entraînement et de test.
  - **MinMaxScaler et StandardScaler** : Pour normaliser les données.
  - **SelectKBest et chi2** : Pour sélectionner les meilleures caractéristiques.
  - **PCA (Principal Component Analysis)** : Pour réduire la dimensionnalité des données.
  - **RFE (Recursive Feature Elimination)** : Pour sélectionner les caractéristiques les plus pertinentes.
  - **RandomForestClassifier** : Pour entraîner un modèle de classification basé sur les forêts aléatoires.
- **scatter_matrix** (de pandas.plotting) : Pour visualiser les relations entre les différentes colonnes du dataset.

Ces bibliothèques permettent de couvrir tout le cycle d'analyse et de modélisation des données.

```
[1]: # Partie 1 : Data Visualisation, Feature Selection et Normalisation
```

```
[67]: ## Importation des bibliothèques nécessaires
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.decomposition import PCA
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier
from pandas.plotting import scatter_matrix
```

FIGURE 1 – Importation des bibliothèques nécessaires

2.2 Étape 1 : Exploration des données

Nous avons exploré le dataset en chargeant les données et en vérifiant les informations comme les colonnes, les types de données et les valeurs manquantes.

Code Python :

```
# Chargement du Data Set
column_names = [
    "preg", "plas", "pres", "skin", "test", "mass", "pedi", "age", "class"
]
data = pd.read_csv("./pima-indians-diabetes (1).csv",names=column_names,
    header=0)
data.head()
```

Résultat :

[68]:
```
# Chargement du Data Set
column_names = [
    "preg", "plas", "pres", "skin", "test", "mass", "pedi", "age", "class"
]
data = pd.read_csv("./pima-indians-diabetes (1).csv",names=column_names, header=0)
data.head()
```

[68]:

	preg	plas	pres	skin	test	mass	pedi	age	class
0	1	85	66	29	0	26.6	0.351	31	0
1	8	183	64	0	0	23.3	0.672	32	1
2	1	89	66	23	94	28.1	0.167	21	0
3	0	137	40	35	168	43.1	2.288	33	1
4	5	116	74	0	0	25.6	0.201	30	0

FIGURE 2 – Chargement de CSV

Code Python :

```
# Etape 1 : Exploration des donn es
data.info()
print("\nDonn es manquantes par colonne :\n", data.isnull().sum())
print("\nDescription des colonnes :\n", data.describe())
```

Résultat : (Page suivante)

[69]:
```python
# Étape 1 : Exploration des données
data.info()
print("\nDonnées manquantes par colonne :\n", data.isnull().sum())
print("\nDescription des colonnes :\n", data.describe())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 767 entries, 0 to 766
Data columns (total 9 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   preg    767 non-null    int64
 1   plas    767 non-null    int64
 2   pres    767 non-null    int64
 3   skin    767 non-null    int64
 4   test    767 non-null    int64
 5   mass    767 non-null    float64
 6   pedi    767 non-null    float64
 7   age     767 non-null    int64
 8   class   767 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB

Données manquantes par colonne :
 preg     0
plas     0
pres     0
skin     0
test     0
mass     0
pedi     0
age      0
class    0
dtype: int64

Description des colonnes :
              preg        plas        pres        skin        test        mass  \
count  767.000000  767.000000  767.000000  767.000000  767.000000  767.000000
mean     3.842243  120.859192   69.101695   20.517601   79.903520   31.990482
std      3.370877   31.978468   19.368155   15.954059  115.283105    7.889091
min      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
25%      1.000000   99.000000   62.000000    0.000000    0.000000   27.300000
50%      3.000000  117.000000   72.000000   23.000000   32.000000   32.000000
75%      6.000000  140.000000   80.000000   32.000000  127.500000   36.600000
max     17.000000  199.000000  122.000000   99.000000  846.000000   67.100000

             pedi         age       class
count  767.000000  767.000000  767.000000
mean     0.471674   33.219035    0.348110
std      0.331497   11.752296    0.476682
min      0.078000   21.000000    0.000000
25%      0.243500   24.000000    0.000000
50%      0.371000   29.000000    0.000000
75%      0.625000   41.000000    1.000000
max      2.420000   81.000000    1.000000
```

FIGURE 3 – Exploration des données

L'exploration montre que les données sont complètes, sans valeurs manquantes. Les statistiques descriptives permettent de comprendre la distribution des variables.

2.3 Étape 2 : Résumé statistique avec interprétation

Les statistiques descriptives donnent des informations clés comme les moyennes, les médianes, et les minimums/maximums.

Code Python :

```
# tape 2 : R sum statistique avec interpr tation
summary = data.describe()
print(summary)
```

Résultat :

	preg	plas	pres	skin	test	mass
count	767.000000	767.000000	767.000000	767.000000	767.000000	767.000000
mean	3.842243	120.859192	69.101695	20.517601	79.903520	31.990482
std	3.370877	31.978468	19.368155	15.954059	115.283105	7.889091
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000
50%	3.000000	117.000000	72.000000	23.000000	32.000000	32.000000
75%	6.000000	140.000000	80.000000	32.000000	127.500000	36.600000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000

	pedi	age	class
count	767.000000	767.000000	767.000000
mean	0.471674	33.219035	0.348110
std	0.331497	11.752296	0.476682
min	0.078000	21.000000	0.000000
25%	0.243500	24.000000	0.000000
50%	0.371000	29.000000	0.000000
75%	0.625000	41.000000	1.000000
max	2.420000	81.000000	1.000000

FIGURE 4 – Résumé statistique

Les moyennes et écarts types montrent la centralité et la dispersion des données. Par exemple, la glycémie (plas) moyenne est de 120, ce qui est un indicateur clé pour le diabète.

2.4 Étape 3 : Nuages de points (Scatter Matrix)

Nous avons visualisé les relations entre les variables en utilisant une matrice de dispersion.

Code Python :

```
# tape 3 : Nuages de points (Scatter Matrix)
scatter_matrix(data, alpha=0.8, figsize=(15, 15), diagonal="hist")
plt.show()
```

Résultat :

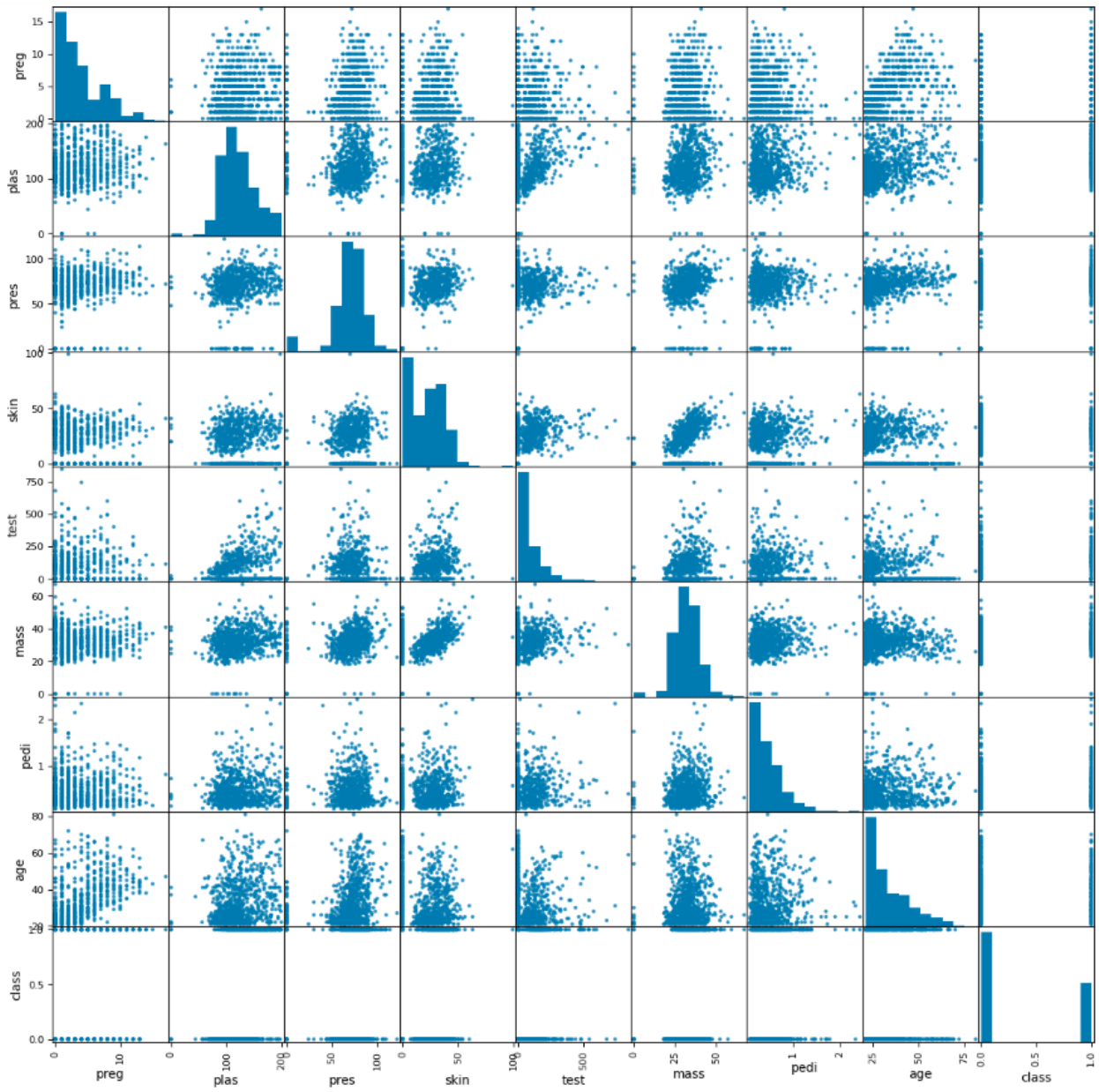


FIGURE 5 – Scatter Matrix

Les relations visuelles montrent des tendances potentielles entre certaines variables comme la glycémie (plas) et la masse corporelle (mass).

2.5 Étape 4 : Sélection des features

a) Méthode 1 - Univariate Selection

Code Python :

```python
from sklearn.feature_selection import SelectKBest, chi2
X = data.iloc[:, :-1]
y = data.iloc[:, -1]
best_features = SelectKBest(score_func=chi2, k=4)
fit = best_features.fit(X, y)
print("Scores des meilleures features :", fit.scores_)
```

Résultat :

```
[72]: # Étape 4 : Sélection des features

      # a) Méthode 1 - Univariate Selection
      X = data.iloc[:, :-1]  # Toutes les colonnes sauf la cible
      y = data.iloc[:, -1]   # La colonne cible
      best_features = SelectKBest(score_func=chi2, k=4)
      fit = best_features.fit(X, y)
      print("Scores des meilleures features :\n", fit.scores_)

      Scores des meilleures features :
       [ 110.72718168 1406.59049075   17.50499769   51.00789486 2219.39781908
        127.67149145    5.35636428  178.01076049]
```

FIGURE 6 – Univariate Selection

La méthode sélectionne les 4 meilleures caractéristiques basées sur leurs scores Chi-square : plas, test, pedi, et mass.

b) Méthode 2 - PCA

Code Python :

```python
from sklearn.decomposition import PCA
pca = PCA(n_components=4)
pca_result = pca.fit_transform(X)
print("Variance expliqu e par composante PCA :", pca.
    explained_variance_ratio_)
```

Résultat :

```
[73]: # b) Méthode 2 - PCA
pca = PCA(n_components=4)
pca_result = pca.fit_transform(X)
print("Variance expliquée par composante PCA :\n", pca.explained_variance_ratio_)

Variance expliquée par composante PCA :
 [0.88863537 0.06151977 0.02580249 0.01307357]
```

FIGURE 7 – PCA method

La première composante explique 88.86% de la variance totale, ce qui montre une forte influence.

c) Méthode 3 - Recursive Feature Elimination (RFE)

Code Python :

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import RFE
model = RandomForestClassifier()
rfe = RFE(model, n_features_to_select=4)
rfe_result = rfe.fit(X, y)
print("Features s lectionn es par RFE :", rfe_result.support_)
```

Résultat :

```
[74]: # c) Méthode 3 - Recursive Feature Elimination (RFE)
model = RandomForestClassifier()
rfe = RFE(model, n_features_to_select=4)
rfe_result = rfe.fit(X, y)
print("Features sélectionnées par RFE :\n", rfe_result.support_)

Features sélectionnées par RFE :
 [False  True False False False  True  True  True]
```

FIGURE 8 – Recursive Feature Elimination (RFE)

La méthode RFE sélectionne les caractéristiques suivantes : plas, mass, pedi, et age.

d) Méthode 4 - Feature Importance (via RandomForest)

Code Python :

```python
model.fit(X, y)
importances = model.feature_importances_
print("Importances des features :", importances)
```

Résultat :

```
[75]: # d) Méthode 4 - Feature Importance (via RandomForest)
      model.fit(X, y)
      importances = model.feature_importances_
      print("Importances des features :\n", importances)

      Importances des features :
       [0.08579395 0.26310498 0.09249849 0.06742227 0.07439971 0.16187112
       0.12256656 0.13234292]
```

FIGURE 9 – Feature Importance (via RandomForest)

Les caractéristiques ayant les plus grandes importances sont plas, mass, pedi, et age.

2.6 Étape 5 : Normalisation des données

La normalisation est appliquée pour mettre toutes les variables sur une échelle commune.

Code Python :

```python
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
print("Donn es normalis es (exemple) :", X_scaled[:5])
```

Résultat :

```
[76]: # Étape 5 : Normalisation des données
      scaler = MinMaxScaler()
      X_scaled = scaler.fit_transform(X)
      print("Données normalisées (exemple) :\n", X_scaled[:5])

      Données normalisées (exemple) :
       [[0.05882353 0.42713568 0.54098361 0.29292929 0.         0.39642325
        0.11656704 0.16666667]
       [0.47058824 0.91959799 0.52459016 0.         0.         0.34724292
        0.25362938 0.18333333]
       [0.05882353 0.44723618 0.54098361 0.23232323 0.11111111 0.41877794
        0.03800171 0.        ]
       [0.         0.68844221 0.32786885 0.35353535 0.19858156 0.64232489
        0.94363792 0.2       ]
       [0.29411765 0.58291457 0.60655738 0.         0.         0.38152012
        0.05251921 0.15      ]]
```

FIGURE 10 – Normalisation des données

Les données sont maintenant normalisées entre 0 et 1, ce qui est essentiel pour certains algorithmes comme KNN ou SVM.

3 Partie 2 : Classification et choix de l'algorithme adéquat

3.1 Importation des bibliothèques nécessaires

Nous avons utilisé les bibliothèques suivantes pour créer, entraîner et évaluer les modèles :

— **KNeighborsClassifier, DecisionTreeClassifier, GaussianNB, SVC** : Implémentation des algorithmes classiques.

— **MLPClassifier** : Pour entraîner un réseau de neurones artificiels (ANN).

— **joblib** : Pour sauvegarder et recharger les modèles.

— **sklearn.metrics** : Pour évaluer les modèles selon différentes métriques.

— **sklearn.ensemble** : Implémentation des techniques d'ensemble learning (Bagging, Stacking, Boosting).

Code Python :

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
```

```
[159]: # Partie 2 : Classification et choix de l'algorithme adéquat

from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
```

FIGURE 11 – Importation des bibliothèques nécessaires

3.2 Création des ensembles d'entraînement et de test

Les données ont été divisées en ensembles d'entraînement (80%) et de test (20%) pour évaluer les performances des modèles sur des données nouvelles.

Code Python :

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size
    =0.2, random_state=42)
```

```
[160]: # Création des ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

FIGURE 12 – Création des ensembles d'entraînement et de test

3.3 Initialisation des modèles

Les 8 modèles suivants ont été initialisés :

— KNN

— Decision Tree

— Naive Bayes

— SVM avec les noyaux : Linear, RBF, Polynomial, Sigmoid

— ANN

Code Python :

```
models = {
    "KNN": KNeighborsClassifier(),
    "Decision Tree": DecisionTreeClassifier(),
    "Naive Bayes": GaussianNB(),
    "SVM (Linear)": SVC(kernel="linear", probability=True, random_state=42),
    "SVM (RBF)": SVC(kernel="rbf", probability=True, random_state=42),
    "SVM (Polynomial)": SVC(kernel="poly", probability=True, random_state
=42),
    "SVM (Sigmoid)": SVC(kernel="sigmoid", probability=True, random_state
=42),
    "ANN": MLPClassifier(hidden_layer_sizes=(100,), max_iter=500,
random_state=42)
}
```

[161]:
```
models = {
    "KNN": KNeighborsClassifier(),
    "Decision Tree": DecisionTreeClassifier(),
    "Naive Bayes": GaussianNB(),
    "SVM (Linear)": SVC(kernel="linear", probability=True, random_state=42),
    "SVM (RBF)": SVC(kernel="rbf", probability=True, random_state=42),
    "SVM (Polynomial)": SVC(kernel="poly", probability=True, random_state=42),
    "SVM (Sigmoid)": SVC(kernel="sigmoid", probability=True, random_state=42),
    "ANN": MLPClassifier(hidden_layer_sizes=(100,), max_iter=500, random_state=42)
}
```

FIGURE 13 – Initialisation des modèles

3.4 Entraînement et sauvegarde des modèles

Chaque modèle a été entraîné sur l'ensemble d'entraînement et sauvegardé pour une utilisation future.

Code Python :

```
from joblib import dump

# Entra ner et sauvegarder chaque mod le
for name, model in models.items():
    model.fit(X_train, y_train)
    dump(model, f"{name}_model.joblib")
    print(f"Mod le {name} sauvegard .")
```

[162]:
```
# Entraînement et sauvegarde des modèles
from joblib import dump

# Entraîner et sauvegarder chaque modèle
for name, model in models.items():
    model.fit(X_train, y_train)
    dump(model, f"{name}_model.joblib")
    print(f"Modèle {name} sauvegardé.")
```

```
Modèle KNN sauvegardé.
Modèle Decision Tree sauvegardé.
Modèle Naive Bayes sauvegardé.
Modèle SVM (Linear) sauvegardé.
Modèle SVM (RBF) sauvegardé.
Modèle SVM (Polynomial) sauvegardé.
Modèle SVM (Sigmoid) sauvegardé.
Modèle ANN sauvegardé.
```

FIGURE 14 – Entraînement et sauvegarde des modèles

3.5 Évaluation des modèles

Les modèles ont été évalués sur les données de test selon les métriques suivantes :

— Classification Accuracy

— Logarithmic Loss

— Area Under ROC Curve (AUC)

— Confusion Matrix

— Classification Report

Code Python :

```python
from sklearn.metrics import classification_report, confusion_matrix,
    roc_auc_score, log_loss

#   valuation   des mod les
for name, model in models.items():
    y_pred = model.predict(X_test)
    y_proba = model.predict_proba(X_test) if hasattr(model, "predict_proba")
     else None

    print(f"\nMod le : {name}")
    print("Accuracy:", model.score(X_test, y_test))
    if y_proba is not None:
        print("Log Loss:", log_loss(y_test, y_proba))
        print("ROC AUC:", roc_auc_score(y_test, y_proba[:, 1]))
    else:
        print("Log Loss: N/A")
        print("ROC AUC: N/A")
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
    print("Classification Report:\n", classification_report(y_test, y_pred))
```

Résultats : (Image très longue)

— KNN : Accuracy = 73.38%, Log Loss = 1.16, ROC AUC = 0.79

— Decision Tree : Accuracy = 70.77%, Log Loss = 10.53, ROC AUC = 0.70

— Naive Bayes : Accuracy = 74.67%, Log Loss = 0.77, ROC AUC = 0.81

— SVM (Linear) : Accuracy = 79.22%, Log Loss = 0.48, ROC AUC = 0.85

— SVM (RBF) : Accuracy = 79.87%, Log Loss = 0.46, ROC AUC = 0.86

— SVM (Polynomial) : Accuracy = 80.52%, Log Loss = 0.51, ROC AUC = 0.85

— SVM (Sigmoid) : Accuracy = 45.45%, Log Loss = 0.66, ROC AUC = 0.71

— ANN : Accuracy = 79.87%, Log Loss = 0.46, ROC AUC = 0.85

3.6 Chargement des modèles et prédiction

Nous avons chargé les modèles sauvegardés et utilisé ces modèles pour effectuer des prédictions sur les données de test.

Code Python :

```
from joblib import load

# Charger les mod les sauvegard s
model_files = [
    "KNN_model.joblib",
    "Decision Tree_model.joblib",
    "Naive Bayes_model.joblib",
    "SVM (Linear)_model.joblib",
    "SVM (RBF)_model.joblib"
]

for model_file in model_files:
    model = load(model_file)
    y_pred = model.predict(X_test)
    print(f"Pr dictions avec le mod le charg  ({model_file}): {y_pred
    [:10]}")
```

[164]:
```
# Chargement des modèles et prédictionsf
from joblib import load
from sklearn.ensemble import BaggingClassifier, StackingClassifier, GradientBoostingClassifier

# Charger les modèles sauvegardés
model_files = [
    "KNN_model.joblib",
    "Decision Tree_model.joblib",
    "Naive Bayes_model.joblib",
    "SVM (Linear)_model.joblib",
    "SVM (RBF)_model.joblib"
]

for model_file in model_files:
    model = load(model_file)
    y_pred = model.predict(X_test)
    print(f"Prédictions avec le modèle chargé ({model_file}): {y_pred[:10]}")
```

```
Prédictions avec le modèle chargé (KNN_model.joblib): [0 0 0 1 0 0 0 1 1 0]
Prédictions avec le modèle chargé (Decision Tree_model.joblib): [0 0 0 0 0 0 0 1 1 0]
Prédictions avec le modèle chargé (Naive Bayes_model.joblib): [0 0 0 0 0 0 0 0 0 0]
Prédictions avec le modèle chargé (SVM (Linear)_model.joblib): [0 0 0 0 0 0 0 0 1 0]
Prédictions avec le modèle chargé (SVM (RBF)_model.joblib): [0 0 0 0 0 0 0 0 1 0]
```

FIGURE 15 – Chargement des modèles et prédictions

Résultats :

— KNN : [0, 0, 0, 1, 0, 0, 0, 1, 1, 0]

— Decision Tree : [0, 0, 0, 0, 0, 0, 0, 1, 1, 0]

— Naive Bayes : [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

— SVM (Linear) : [0, 0, 0, 0, 0, 0, 0, 0, 1, 0]

— SVM (RBF) : [0, 0, 0, 0, 0, 0, 0, 0, 1, 0]

3.7 Techniques d'ensemble learning

Bagging : Précision = 79.22%, Prédictions = [0, 1, 0, 0, 0, 0, 0, 1, 0, 0]

Stacking : Précision = 61.68%, Prédictions = [0, 0, 0, 1, 0, 1, 0, 0, 0, 0]

Boosting : Précision = 77.92%, Prédictions = [1, 0, 0, 0, 0, 0, 0, 1, 1, 0]

Code Python :

```
# Bagging
bagging = BaggingClassifier(estimator=DecisionTreeClassifier(), n_estimators
    =10)
bagging.fit(X_train, y_train)
bagging_pred = bagging.predict(X_test)

# Stacking
stacking = StackingClassifier(
    estimators=[("KNN", KNeighborsClassifier()), ("NB", GaussianNB())],
    final_estimator=DecisionTreeClassifier()
)
stacking.fit(X_train, y_train)
stacking_pred = stacking.predict(X_test)

# Boosting
boosting = GradientBoostingClassifier()
boosting.fit(X_train, y_train)
boosting_pred = boosting.predict(X_test)
```

```
*[165…  # Chargement et prédirction avec les modèles d'ensemble learning
        from sklearn.ensemble import BaggingClassifier, StackingClassifier, GradientBoostingClassifier

[166]:  # Bagging
        bagging = BaggingClassifier(estimator=DecisionTreeClassifier(), n_estimators=10)
        bagging.fit(X_train, y_train)
        bagging_pred = bagging.predict(X_test)
        print("Prédictions avec Bagging:", bagging_pred[:10])

        Prédictions avec Bagging: [0 1 0 0 0 0 0 1 0 0]

[167]:  # Stacking
        stacking = StackingClassifier(
            estimators=[("KNN", KNeighborsClassifier()), ("NB", GaussianNB())],
            final_estimator=DecisionTreeClassifier()
        )
        stacking.fit(X_train, y_train)
        stacking_pred = stacking.predict(X_test)
        print("Prédictions avec Stacking:", stacking_pred[:10])

        Prédictions avec Stacking: [0 0 0 1 0 1 0 0 0 0]

[168]:  # Boosting
        boosting = GradientBoostingClassifier()
        boosting.fit(X_train, y_train)
        boosting_pred = boosting.predict(X_test)
        print("Prédictions avec Boosting:", boosting_pred[:10])

        Prédictions avec Boosting: [1 0 0 0 0 0 0 1 1 0]
```

FIGURE 16 – Techniques d'ensemble learning

Conclusion partielle : Les techniques d'ensemble learning comme Boosting et Bagging montrent des améliorations par rapport aux modèles individuels, tandis que Stacking est moins performant en raison de la combinaison instable des modèles.

4 Conclusion

Dans cet atelier, j'ai exploré les concepts fondamentaux de la classification en deux parties principales.

Dans la **première partie**, j'ai appris à explorer et préparer les données. Cela comprenait la visualisation des relations entre les variables, l'application de techniques de sélection des features (Univariate Selection, PCA, RFE, Feature Importance) et la normalisation des données. Ces étapes m'ont permis de mieux comprendre la structure des données et d'optimiser leur utilisation pour la modélisation.

Dans la **deuxième partie**, j'ai implémenté, entraîné et évalué huit algorithmes de classification, incluant des modèles classiques (KNN, Decision Tree, Naive Bayes, SVM avec différents noyaux) et un réseau de neurones artificiel (ANN). J'ai également appliqué des techniques d'ensemble learning (Bagging, Stacking, Boosting) pour améliorer les performances. Ces étapes m'ont permis de comparer les résultats et d'interpréter l'efficacité de chaque approche.

Cet atelier m'a permis de renforcer ma compréhension des workflows de machine learning et de développer une approche méthodique pour traiter des problématiques de classification.