جامعة ابن طفيل
Ibn Tofaïl University

Rapport de projet

Sous-thème : Réalisation des modèles de classification pour la prédiction des maladies cardiaques

Préparé par :
EL MAHDI ATMANI

Encadré par :
Hatim derrouz

Année : 2023/2024

Liste des figures :

Table des matières

1. Objectives

Les objectifs de ce plan sont les suivants :

1. Réaliser Trois modèles qui prédisent les maladies cardiaques.
2. Comparer les Trois algorithmes puis décider le meilleur et pourquoi.

2. Sujet

Dans ce sujet, nous plongerons dans les nuances de trois algorithmes de classification fondamentaux en apprentissage automatique : SVM (Support Vector Machine), KNN (K-Nearest Neighbors) et RandomForest. En examinant leurs principes, leurs applications et leurs performances dans divers contextes, nous découvrirons comment ces approches peuvent être utilisées pour résoudre des problèmes de classification dans des domaines aussi variés que la santé, la finance et la reconnaissance de formes. En analysant en profondeur leurs forces et leurs faiblesses, nous fournirons également des conseils pratiques sur le choix de l'algorithme le plus adapté à des problèmes spécifiques.

3. Introduction

Cette étude ambitieuse vise à développer Trois modèles d'intelligence artificielle pour prédire les maladies cardiaques en se basant sur un ensemble de données provenant de Kaggle. Ces modèles seront entraînés à identifier les individus les plus à risque en utilisant des facteurs tels que l'âge, le rythme cardiaque et la glycémie. Deux algorithmes d'apprentissage automatique seront comparés pour déterminer le plus performant. Cette recherche offre une opportunité cruciale d'améliorer la détection précoce des maladies cardiaques, permettant ainsi une intervention rapide et une meilleure gestion des risques pour les patients.

Le processus de développement des modèles impliquera une sélection minutieuse des données et une formation intensive pour garantir leur précision et leur fiabilité. L'objectif ultime est de fournir aux professionnels de la santé un outil précieux pour identifier les personnes les plus vulnérables aux maladies cardiaques, ce qui pourrait avoir un impact significatif sur la santé publique en permettant des interventions préventives et des soins ciblés.

4. PARAGRAPHE 1 :

Phase donnée :

- **La collection des données (ou téléchargement à partir de Kaggle).**

4.1 Explication

Le jeu de données sur les maladies cardiaques est un ensemble volumineux de dossiers de patients compilés à partir de cinq sources différentes. Il comprend 1190 instances et 11 caractéristiques, à savoir :

- Âge
- Sexe
- Type de douleur thoracique
- Pression artérielle au repos
- Cholestérol
- Glycémie à jeun
- Résultats de l'ECG au repos
- Fréquence cardiaque maximale atteinte
- Angine induite par l'exercice
- Dépression ST induite par l'exercice par rapport au repos
- Cible (0 = pas de maladie, 1 = maladie)

Le jeu de données peut être utilisé pour entraîner des modèles d'apprentissage automatique afin de prédire si un patient a une maladie cardiaque. Il a été utilisé dans un certain nombre d'études de recherche et constitue une ressource précieuse pour les chercheurs travaillant sur le développement de nouvelles méthodes de diagnostic et de traitement des maladies cardiaques.

	age	sex	chest pain type	resting bp s	cholesterol	fasting blood sugar	resting ecg	max heart rate	exercise angina	oldpeak	ST slope	target
0	40	male	2	140	289	0	0	172	0	0.0	1	normal
1	49	female	3	160	180	0	0	156	0	1.0	2	heart disease
2	37	male	2	130	283	0	1	98	0	0.0	1	normal
3	48	female	4	138	214	0	0	108	1	1.5	2	heart disease
4	54	male	3	150	195	0	0	122	0	0.0	1	normal
...	...	...	...	...	...	...	...	...	...	...	...	...
1185	45	male	1	110	264	0	0	132	0	1.2	2	heart disease
1186	68	male	4	144	193	1	0	141	0	3.4	2	heart disease
1187	57	male	4	130	131	0	0	115	1	1.2	2	heart disease
1188	57	female	2	130	236	0	2	174	0	0.0	2	heart disease
1189	38	male	3	138	175	0	0	173	0	0.0	1	normal

1190 rows × 12 columns

Figure 1: Dataset des maladies cardiaques

4.2 Tâche

Effectuer le prétraitement des données en nettoyant les valeurs manquantes, en normalisant les caractéristiques si nécessaire, et en convertissant les variables catégoriques en variables numériques si nécessaire. Diviser ensuite l'ensemble de données en ensembles d'entraînement et de test pour évaluer les performances des modèles. Entraîner un modèle SVM, un modèle kNN et un modèle RandomForest sur l'ensemble d'entraînement. Évaluer les performances des modèles en utilisant des métriques telles que l'exactitude pour la classification, puis comparer les performances des trois modèles. Enfin, analyser les résultats obtenus et discuter des avantages et des inconvénients de chaque algorithme en termes de précision, de vitesse d'exécution et de capacité à généraliser sur de nouvelles données.

5. PARAGRAPHE 2 :

Phase Prétraitement :

– Traitement des données

5.1 Explication

Le prétraitement des données est une étape cruciale dans l'analyse de données et la création de modèles d'apprentissage automatique. Il implique la préparation des données brutes en vue de les rendre utilisables pour l'analyse et la modélisation. Cette étape comprend plusieurs tâches telles que la suppression des valeurs manquantes, la normalisation des caractéristiques pour mettre toutes les données à la même échelle si nécessaire, et la conversion des variables catégoriques en variables numériques pour permettre l'application d'algorithmes d'apprentissage automatique. Le prétraitement des données garantit la qualité et la cohérence des données, ce qui est essentiel pour obtenir des résultats précis et fiables lors de l'entraînement des modèles.

5.2 Tâche

5.2.1 Visualisation des données

- **Affichage des Informations sur le DataFrame :** la méthode **info** est appelée sur le DataFrame pour afficher des informations sur le DataFrame, y compris les types de données de chaque colonne et le nombre de valeurs non nulles. Cela fournit un aperçu de la structure et du contenu du DataFrame.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1190 entries, 0 to 1189
Data columns (total 12 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   age                  1190 non-null   int64
 1   sex                  1190 non-null   object
 2   chest pain type      1190 non-null   int64
 3   resting bp s         1190 non-null   int64
 4   cholesterol          1190 non-null   int64
 5   fasting blood sugar  1190 non-null   int64
 6   resting ecg          1190 non-null   int64
 7   max heart rate       1190 non-null   int64
 8   exercise angina      1190 non-null   int64
 9   oldpeak              1190 non-null   float64
 10  ST slope             1190 non-null   int64
 11  target               1190 non-null   object
dtypes: float64(1), int64(9), object(2)
memory usage: 111.7+ KB
```

- **Affichage de la Carte de Chaleur de Corrélation :** La carte de chaleur de corrélation pour le DataFrame en utilisant la fonction **heatmap** de **seaborn**. Il fournit une représentation visuelle de la corrélation entre différentes caractéristiques du DataFrame. Les valeurs de corrélation sont annotées sur la carte de chaleur, avec des couleurs plus chaudes indiquant une corrélation positive plus forte et des couleurs plus froides indiquant une corrélation négative plus forte.

```
plt.figure(figsize=(20,10))
sns.heatmap(df.corr(),annot=True,cmap="coolwarm")
```

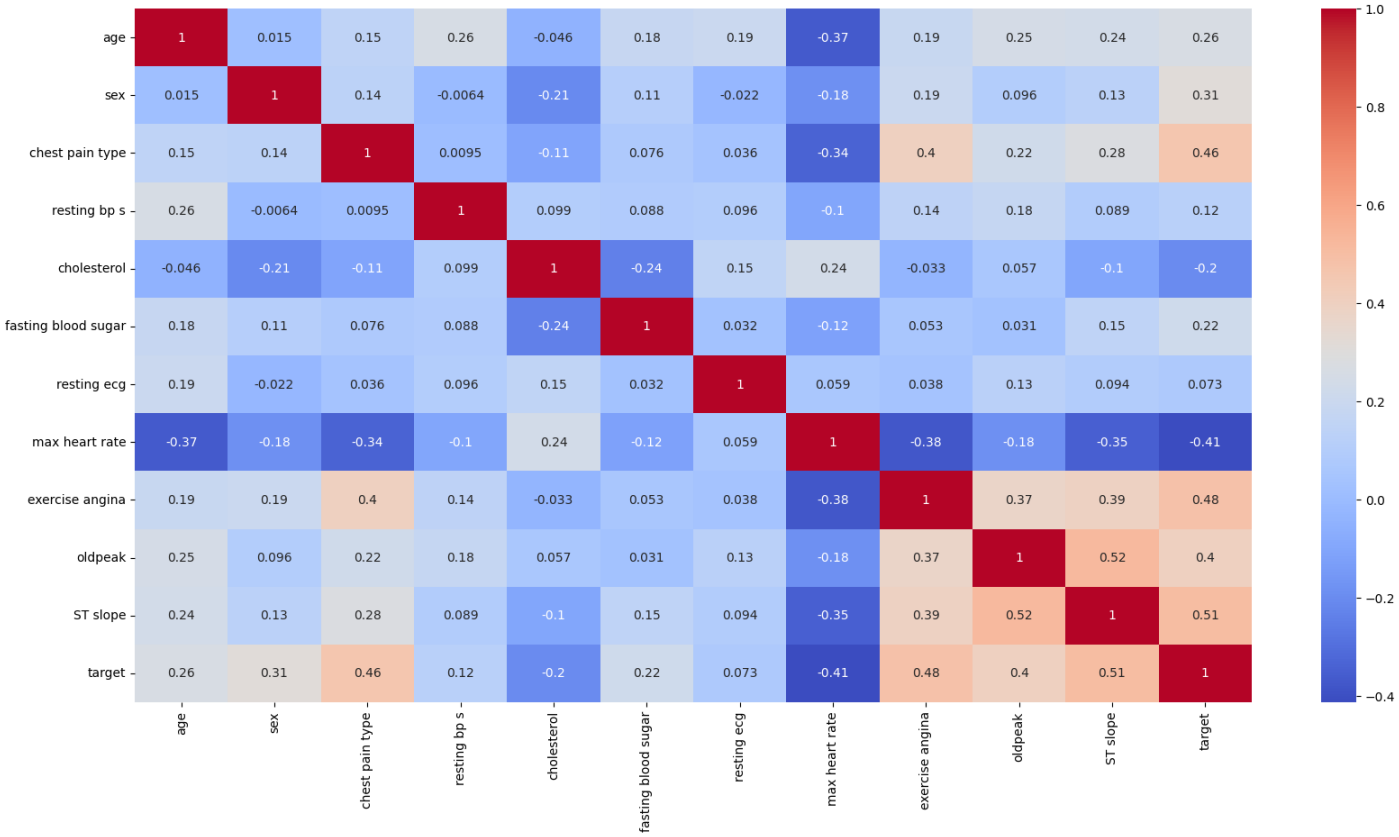

Figure 2 : heatmap Correlation

- **Affichage des Histogrammes :** les histogrammes pour chaque caractéristique du DataFrame en utilisant la fonction **hist**. Les histogrammes fournissent une représentation graphique de la distribution des données dans chaque caractéristique, ce qui permet d'identifier les modèles et les valeurs aberrantes.

```
df.hist(figsize=(20,10),bins = 50);
```

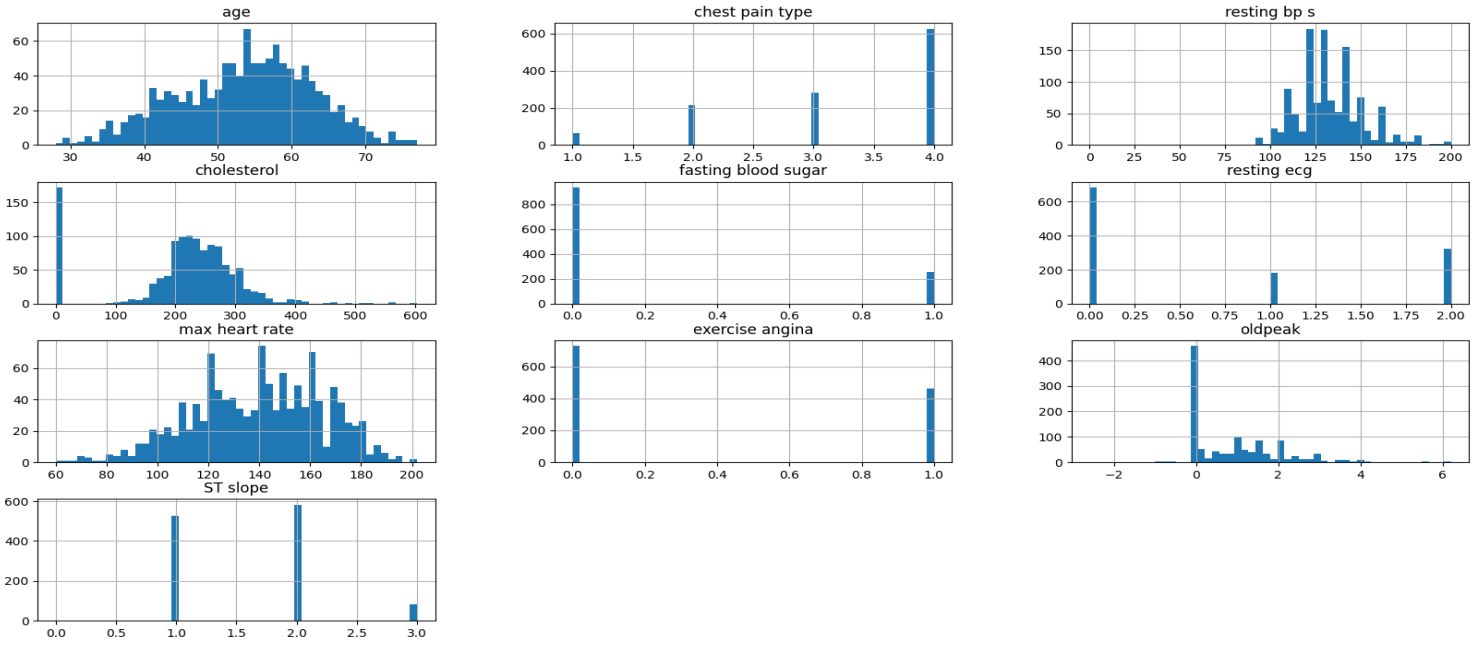

Figure 3: Histogrammes

5.2.2 Observations

1. L'ensemble de données semble propre.
2. Toutes les valeurs sont numériques et non nulles.
3. La colonne 'target' est assez bien équilibrée.
4. La répartition de toutes les colonnes semble bien équilibrée. Seuls « cholestérol » et « oldpeak » pourraient avoir une distribution asymétrique avec de longues queues droites.

5.2.3 Le traitement des datasets

- **Calcul des Pourcentages de Cholestérol :** en calcule le pourcentage de cholestérol pour chaque point de données dans le DataFrame. Il trouve les valeurs minimale et maximale de la colonne 'cholesterol', puis applique la formule, **((df['cholesterol'] - min_cholesterol) / (max_cholesterol - min_cholesterol) * 100).map('{:.0f}'.format)** pour mettre à l'échelle chaque valeur à un pourcentage entre 0 et 100. Le résultat est arrondi à l'entier le plus proche en utilisant la fonction map avec une chaîne de formatage.

- Soit x_i les nombres d'origine dans la liste.
- Soit $x_{\min}$ la valeur minimale dans la liste.
- Soit $x_{\max}$ la valeur maximale dans la liste.
- Soit $\text{Range} = x_{\max} - x_{\min}$

La formule pour convertir chaque x_i en pourcentage p_i est

$$p_i = \left(\frac{x_{\max} - x_{\min}}{\text{Range}}\right) \times 100$$

```
min_cholesterol = df['cholesterol'].min()
max_cholesterol = df['cholesterol'].max()
df['cholesterol'] = ((df['cholesterol'] - min_cholesterol) /
(max_cholesterol - min_cholesterol) * 100).map('{:.0f}'.format)
```

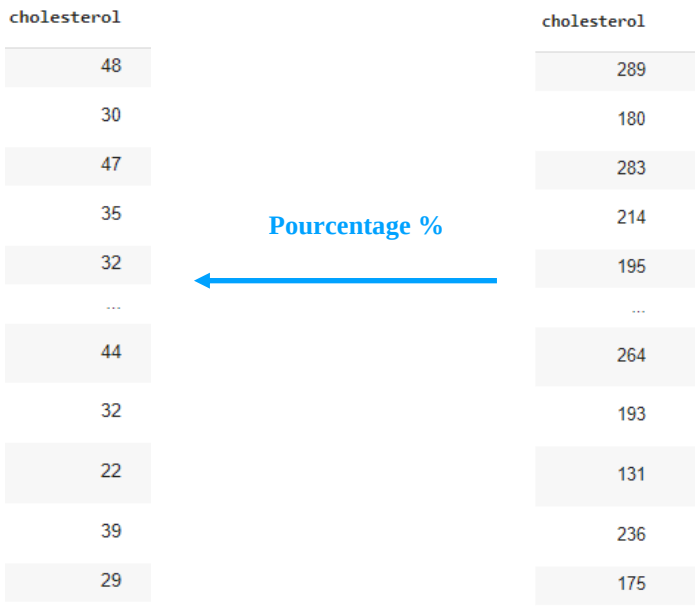

Figure 4:Transformation au Pourcentage

- **Remplacement des Valeurs dans la Colonne ‘sex’ et ‘target’ :** Ici, les valeurs dans la colonne 'sex' sont remplacées par des équivalents numériques. 'female' est remplacé par 0 et 'male' est remplacé par 1, ce qui transforme les données catégoriques en forme numérique adaptée aux algorithmes d'apprentissage automatique.

```
# Replace values in 'sex' column
df['sex'] = df['sex'].replace({'female': 0, 'male': 1})

# Replace values in 'target' column
df['target'] = df['target'].replace({'normal': 0, 'heart disease': 1})
```

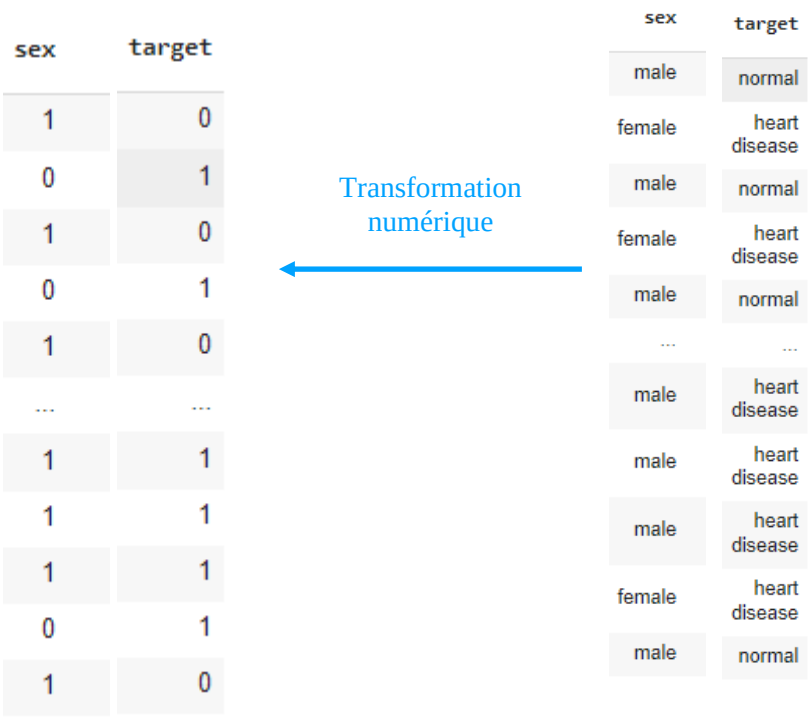

Figure 5: transformation équivalents numériques

6. PARAGRAPHE 2 :

Phase Séparation des données :

– Division des données

6.1 Explication

La division des données consiste à séparer l'ensemble de données en ensembles d'entraînement et de test. L'ensemble d'entraînement est utilisé pour former le modèle, tandis que l'ensemble de test évalue sa performance sur des données inconnues. Cette pratique est cruciale pour estimer avec précision la capacité du modèle à généraliser sur de nouvelles données, aidant ainsi à identifier les problèmes de surapprentissage ou de sous-apprentissage.

6.2 Tâche

6.2.1 Division avec la fonction « train_test_split() »

Définition des Variables X et y : Les caractéristiques (features) sont stockées dans la variable X, et la colonne cible ('**target**') est stockée dans la variable y.

Division des Données : Les données sont séparées en ensembles d'entraînement et de test à l'aide de la fonction **train_test_split**. Les proportions sont spécifiées avec **test_size=0.2 (20%)**, et **random_state=42** est utilisé pour assurer la reproductibilité des résultats. Les ensembles d'entraînement et de test sont stockés dans les variables **X_train, X_test, y_train et y_test**.

```
# Définir une variable X pour les caractéristiques (features) du
dataframe et y pour la colonne target.
X = df.drop('target', axis=1)
y = df['target']

# Diviser les données en ensembles d'entraînement et de test.
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,
random_state=42)
```

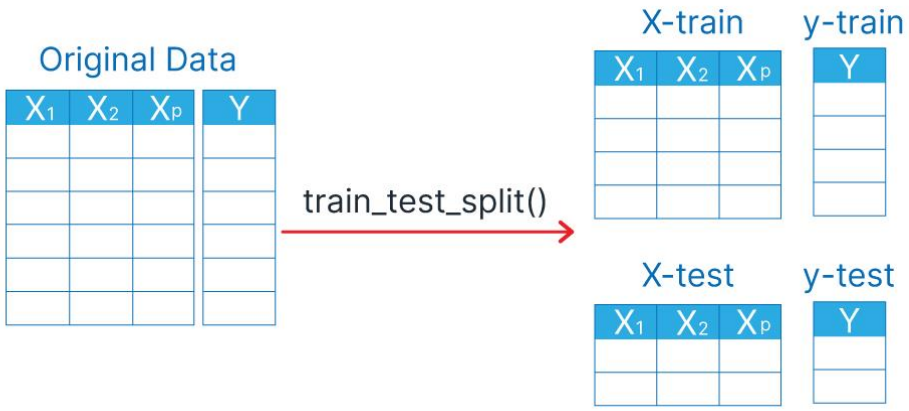

Figure 6: la division de données train_test_split

6.2.2 Division avec la validation croisée

La validation croisée, l'ensemble de données est divisé en plusieurs sous-ensembles, et le modèle est entraîné et évalué plusieurs fois en utilisant différentes combinaisons de ces sous-ensembles. Chaque sous-ensemble est utilisé à la fois pour l'entraînement et la validation, et la performance moyenne sur toutes les itérations est calculée. La validation-croisée aide à garantir que la performance du modèle est cohérente sur différents sous-ensembles de données et fournit une estimation plus robuste de sa performance.

```
scores = cross_val_score(model, X, y, cv=5) # cv=5 means 5-fold cross-
validation
```

L'utilise la fonction **cross_val_score()** de la bibliothèque scikit-learn pour évaluer les performances d'un modèle de classification
cv=5 spécifie que la validation croisée sera effectuée en utilisant une division de l'ensemble de données en 5 plis.

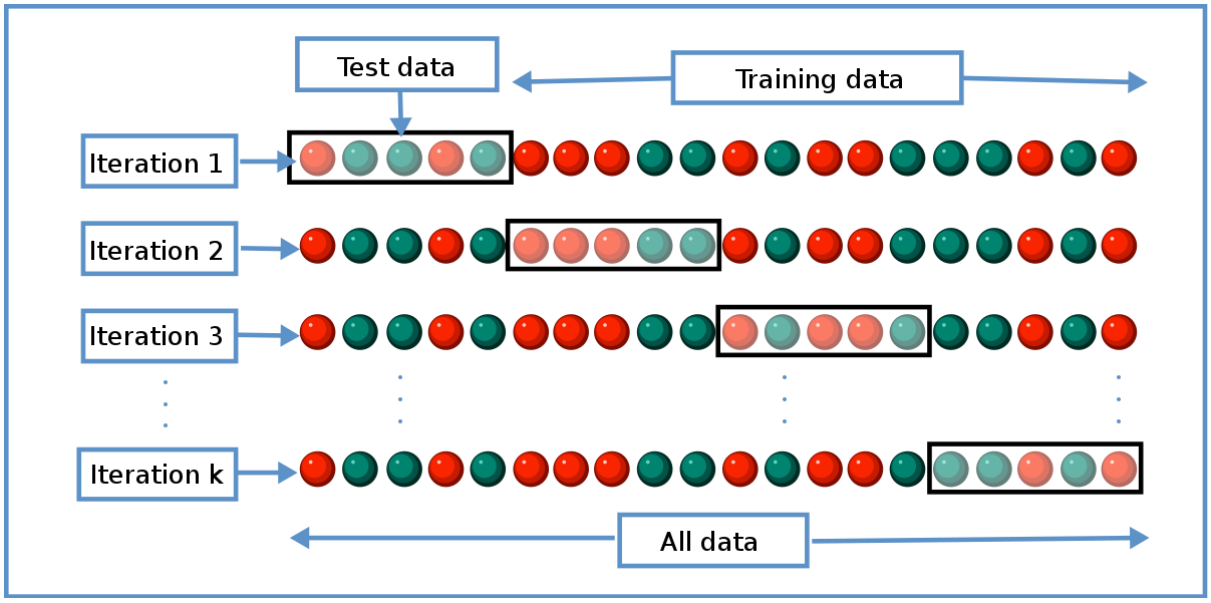

Figure 7: la division de données cross validation

7. PARAGRAPHE 3 :

Phase Entraînement :

– Entraînement des modèles

7.1 Explication

L'entraînement des modèles KNN, RandomForest et SVM implique de fournir les données d'entraînement, ajuster les paramètres si nécessaire, puis laisser chaque algorithme apprendre à partir de ces données pour produire un modèle prédictif. KNN mémorise simplement les données, RandomForest construit un ensemble d'arbres de décision, tandis que SVM trouve l'hyperplan optimal pour séparer les classes. Une fois entraînés, ces modèles peuvent être utilisés pour prédire de nouvelles données.

7.2 Tâche

7.2.1 K plus proches-voisins (KNN)

Le K plus proches voisins (kNN) est un algorithme d'apprentissage supervisé utilisé pour la classification et la régression. Pour classer un nouvel exemple, kNN trouve les k exemples d'entraînement les plus proches dans l'espace des caractéristiques et attribue à l'exemple la classe majoritaire parmi ces k voisins.

La distance dans l'algorithme kNN est généralement calculée à l'aide de métriques telles que la distance euclidienne, la distance de Manhattan ou la distance de Minkowski.

Distance Euclidienne : Elle calcule la distance en ligne droite entre deux points dans l'espace des caractéristiques.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Distance de Manhattan : Aussi appelée distance de la ville ou distance L1, elle mesure la somme des différences absolues entre les coordonnées de deux points. Pour deux points (x_1,y_1) et (x_2,y_2), la distance de Manhattan d est donnée par :

$$d = | x_2 - x_1 | + | y_2 - y_1 |$$

Autre (Minkowski…)

7.2.1.1 Modélisation

- En importe les modules nécessaires : **KNeighborsClassifier** de **sklearn.neighbors** pour construire le modèle KNN et **cross_val_score** de **sklearn.model_selection** pour la validation croisée.
- Puis initialise des listes vides pour stocker les précisions (**accuracies**) et suit la meilleure valeur de k (**best_k**) ainsi que son score moyen correspondant (**knn_best_mean_score**).
- En itère sur différentes valeurs de k de 1 à 30 (inclus) pour trouver le nombre optimal de voisins.
- À chaque itération, il crée un modèle KNN avec la valeur actuelle de k et effectue une validation croisée à 5 plis (**cv=5**) pour évaluer sa performance.
- Le score moyen des résultats de la validation croisée est calculé et stocké dans **mean_score**.
- Il imprime la valeur de k, les scores de validation croisée et le score moyen pour chaque itération.
- Il ajoute le score moyen à la liste accuracies.
- Si le score moyen actuel est supérieur au meilleur score moyen précédent (**knn_best_mean_score**), il met à jour knn_best_mean_score et best_k.
- Après avoir itéré sur toutes les valeurs de k, il trace un graphique montrant les scores moyens en fonction des valeurs de k.
- Il imprime la meilleure valeur de k (**best_k**) ainsi que sa précision correspondante (**knn_best_mean_score**).

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score

accuracies = []
best_k = None
knn_best_mean_score = 0

# Parcourir différentes valeurs de k pour trouver le nombre optimal de voisins pour le KNN
for k in range(1, 30):
    knn_model = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn_model, X, y, cv=5) # cv=5 means 5-fold cross-validation
    mean_score = scores.mean()
    print("k:", k, "Cross-Validation Scores:", scores, "Mean Score:", mean_score)
    accuracies.append(mean_score)
# Mettre à jour le meilleur score moyen et le meilleur k si un meilleur score moyen est trouvé
    if mean_score > knn_best_mean_score:
        knn_best_mean_score = mean_score
        best_k = k

# Affichage des resultas
plt.figure(figsize=(10, 6))
plt.plot(range(1, 30), accuracies, color='blue', linestyle='dashed', marker='o', markerfacecolor='red', markersize=10)
plt.title('mean_score vs. K Value')
plt.xlabel('K Value')
plt.ylabel('mean_score')
plt.xticks(np.arange(1, 30, step=1))
plt.grid()
plt.show()

print("Best K:", best_k, "with Accuracy:", knn_best_mean_score)
```

Figure 8: le code de modélisation KNN

Après avoir parcouru chaque valeur de k de 1 à 30 et calculé le score moyen pour chaque k afin de déterminer le meilleur paramètre avec le score le plus élevé, dans ce cas, le k, nous entraînons notre modèle KNN avec ce paramètre choisi, ici **k = 1**.

```
k: 1 Cross-Validation Scores: [0.63865546 0.73109244 0.84033613 0.88655462 0.94537815] Mean Score: 0.8084033613445378
k: 2 Cross-Validation Scores: [0.6302521  0.77731092 0.7605042  0.68487395 0.74789916] Mean Score: 0.7201680672268906
k: 3 Cross-Validation Scores: [0.64285714 0.72689076 0.79831933 0.68487395 0.76470588] Mean Score: 0.723529411764706
k: 4 Cross-Validation Scores: [0.66386555 0.76470588 0.81092437 0.63865546 0.69327731] Mean Score: 0.7142857142857142
k: 5 Cross-Validation Scores: [0.67226891 0.71428571 0.79411765 0.69327731 0.72268908] Mean Score: 0.7193277310924369
k: 6 Cross-Validation Scores: [0.67226891 0.76890756 0.78991597 0.67647059 0.69747899] Mean Score: 0.7210084033613446
k: 7 Cross-Validation Scores: [0.70168067 0.7394958  0.80252101 0.68487395 0.70168067] Mean Score: 0.7260504201680673
k: 8 Cross-Validation Scores: [0.68907563 0.74789916 0.82352941 0.67226891 0.69327731] Mean Score: 0.7252100840336135
```

Figure 9 : Calcul de Score Moyenne de cross validation

Voici une figure qui permet de visualiser facilement le résultat pour la discision qui mention le moyenne score en fonction de la valeur de **k**.

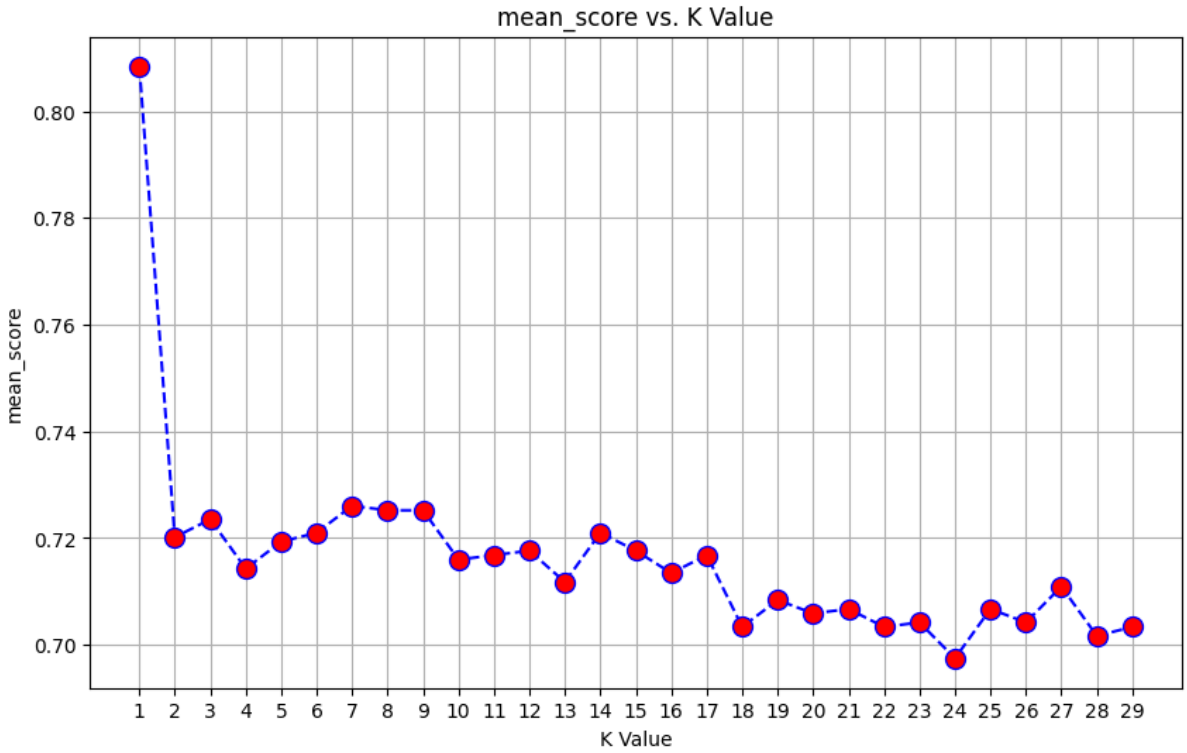

Figure 10 : cross validation moyenne score en fonction de k

7.2.1.2 Entrainement final de modèle.

- **Importations des bibliothèques** : Les bibliothèques nécessaires sont importées, notamment KNeighborsClassifier de sklearn.neighbors pour construire le classifieur KNN et diverses métriques d'évaluation de sklearn.metrics.
- **Instanciation du classifieur KNN** : Le classifieur KNN est instancié avec le nombre optimal de voisins (best_k) déterminé précédemment.
- **Entraînement du modèle** : Le classifieur KNN est entraîné sur les données d'entraînement (X_train, y_train) à l'aide de la méthode fit().
- **Prédiction** : Le classifieur entraîné est utilisé pour faire des prédictions sur les données de test (X_test) à l'aide de la méthode predict().
- **Évaluation des performances** : Diverses métriques de performance telles que l'exactitude, la matrice de confusion, la précision et le rappel sont calculées à l'aide de fonctions de sklearn.metrics.
- **Affichage de l'exactitude :** Le score d'exactitude du classifieur KNN sur les données de test est affiché.
- **Tracé de la matrice de confusion** : La matrice de confusion, qui montre les comptages de vrais positifs, de faux positifs, de vrais négatifs et de faux négatifs, est tracée sous forme de heatmap à l'aide de matplotlib et de seaborn.

```python
#implimenté le knn classifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix,accuracy_score,recall_score,precision_score

cancer_predect = KNeighborsClassifier(n_neighbors=best_k)
cancer_predect.fit(X_train,  y_train)
predicted = cancer_predect.predict(X_test)


accuracy = accuracy_score(predicted,y_test)
print("KNN Accuracy Score: ",accuracy)

conf_matrix = confusion_matrix(y_test, predicted)

# Tracer la matrice de confusion sous forme de heatmap
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```

Figure 11: le code du modèle final kNN

Après d'exécuté le code en haut les résultats suivants s'affichent.

```
KNN Accuracy Score:  0.8067226890756303
```

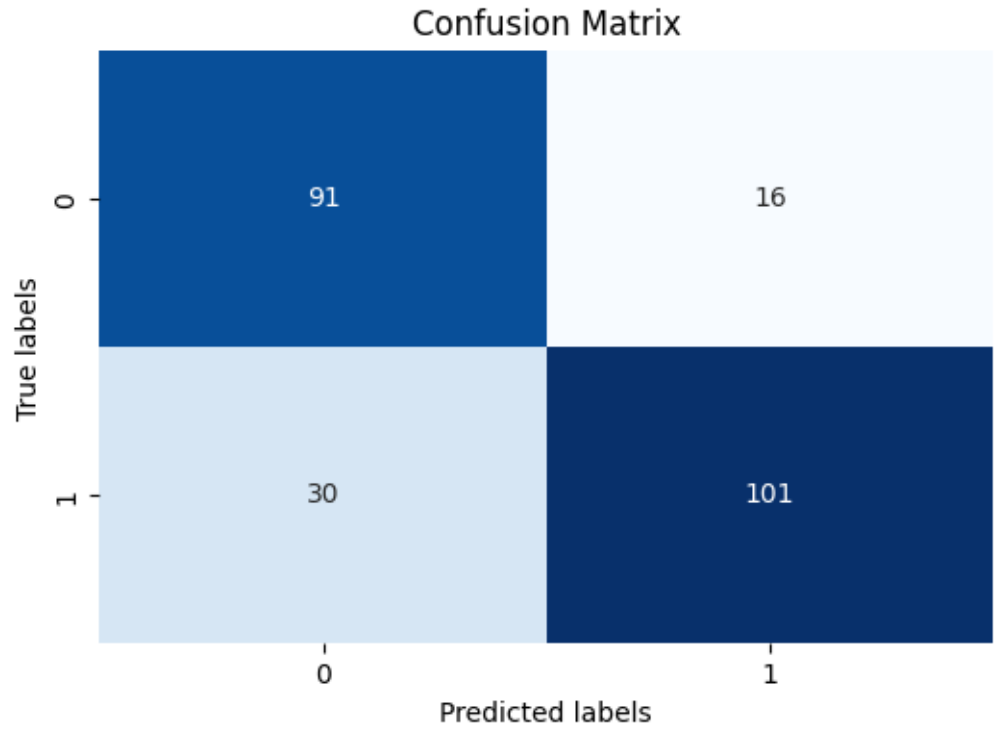

Figure 12: Confusion Matrix de kNN modéle

7.2.1.3 Interprétation des résultats.

La fonction `confusion_matrix()` de scikit-learn pour évaluer la performance du modèle. La matrice de confusion obtenue est visualisée sous forme de heatmap.

La matrice de confusion en haut montre les résultats suivants :

- **91** Vrais Négatifs : Ce sont les échantillons qui ont été correctement prédits comme négatifs.
- **16** Faux Positifs : Ce sont les échantillons qui ont été incorrectement prédits comme positifs.
- **30** Faux Négatifs : Ce sont les échantillons qui ont été incorrectement prédits comme négatifs.
- **101** Vrais Positifs : Ce sont les échantillons qui ont été correctement prédits comme positifs.

La fonction `accuracy_score` compare ensuite les prédictions du modèle aux vraies valeurs de classe et calcule le pourcentage de prédictions correctes, le résultat est :

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Accuracy} = \frac{91+101}{91+16+30+101} = \frac{192}{238}$$

$$= Accuracy \approx 0.8067$$

Dans l'ensemble, le modèle a une précision de 80,67 %. Cela signifie que le modèle a classé correctement 80,67 % des échantillons de l'ensemble de test.

7.2.2 Support Vector Machine (SVM)

La Machine à Vecteurs de Support (SVM) est un autre algorithme d'apprentissage supervisé utilisé pour les tâches de classification et de régression. Contrairement au k plus proches voisins (kNN), SVM vise à trouver l'hyperplan optimal qui sépare le mieux les classes dans l'espace des caractéristiques.

L'hyperplan : Le but principal de SVM est de trouver l'hyperplan qui sépare le mieux les points de données des différentes classes dans l'espace des caractéristiques tout en maximisant la marge, qui est la distance entre l'hyperplan et les points de données (Support Vector) les plus proches de chaque classe. Cet hyperplan est appelé frontière de décision.

Marge : La marge est définie comme la distance entre la frontière de décision et le point de données le plus proche (vecteur de support) de n'importe quelle classe. SVM vise à maximiser cette marge car une marge plus grande indique généralement de meilleures performances de généralisation sur des données non vues.
Vecteurs de Support : Ce sont les points de données qui se trouvent le plus près de la frontière de décision. Ce sont les éléments critiques pour définir la frontière de décision et déterminer la marge. Seuls ces vecteurs de support influencent la position et l'orientation de l'hyperplan.

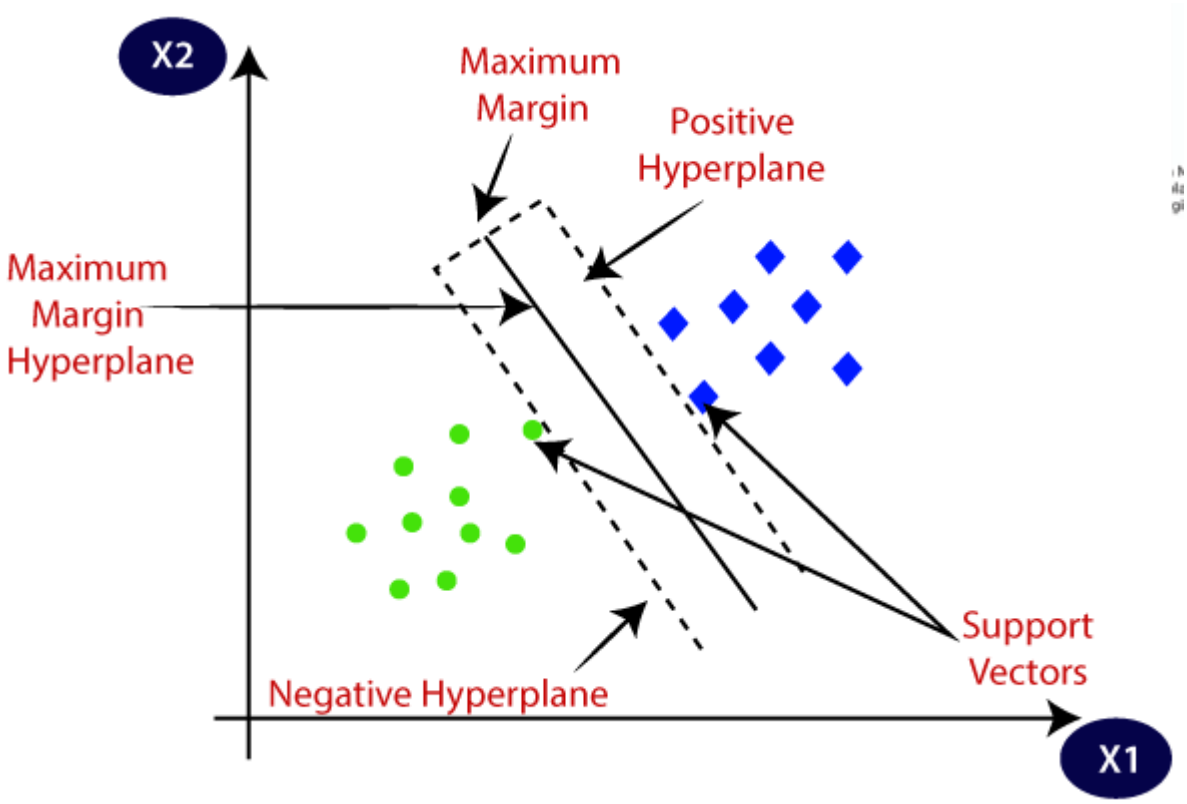

Figure 13: Linear Svm et sont caractéristiques

Fonctions de Noyau Différentes

Dans le contexte des Machines à Vecteurs de Support (SVM), un noyau (kernel) est une fonction qui prend des données en entrée et les transforme dans un espace de dimensions supérieures. Le noyau linéaire, en particulier, est l'un des noyaux les plus simples utilisés dans les SVM

a. Noyau Linear

Le noyau linéaire, la frontière de décision entre les classes est une ligne droite (ou un plan dans des dimensions supérieures). Mathématiquement, le noyau linéaire calcule le produit scalaire entre les paires de points de données. Il est défini comme suit :

$$K(x_i, x_j) = x_i \cdot x_j$$

Ici, x_i et x_j représentent deux points de données. Le produit scalaire x_i, xj mesure essentiellement la similarité entre ces points. Si le produit scalaire est grand, cela signifie que les points sont similaires et appartiennent probablement à la même classe ; s'il est petit, ils sont différents et peuvent appartenir à des classes différentes.

Fig.3

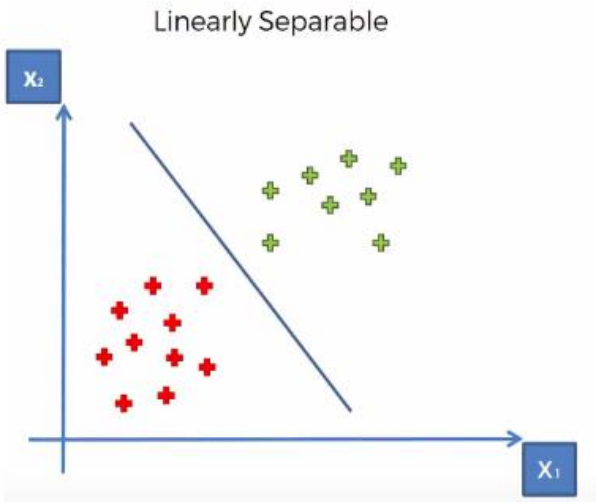

Figure 14: noyau linear

b. Noyau Polynomiale

Voici la formule pour le noyau polynomiale :

$$f(X1, X2) = (X1^T . X2 + 1)^d$$

Ici, d est le degré du polynôme, que nous devons spécifier manuellement.
Supposons que nous ayons deux caractéristiques, X1 et X2, et une variable de sortie Y. En utilisant le noyau polynomial, nous pouvons l'écrire comme suit :

$$X1^T . X2 = \begin{bmatrix} X1 \\ X2 \end{bmatrix} . [X1 \quad X2]$$

$$= \begin{bmatrix} X1^2 & X1.X2 \\ X1.X2 & X2^2 \end{bmatrix}$$

Alors, essentiellement, nous devons trouverX_1^2, x_2^2 et X1.X2, et maintenant nous pouvons voir que 2 dimensions ont été converties en 5 dimensions.

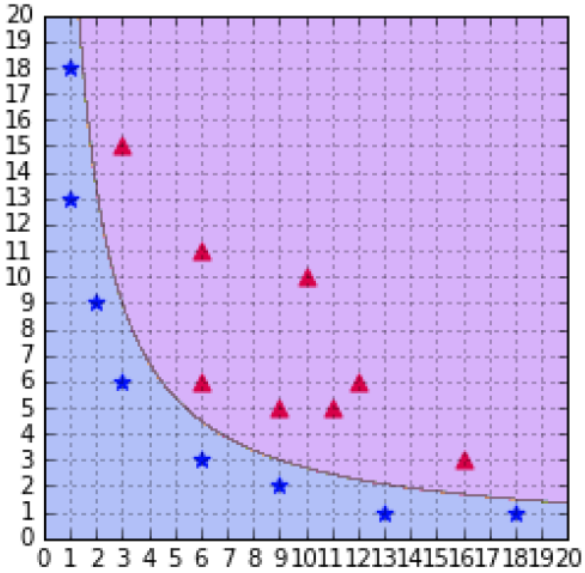

Figure 15: noyau poly

c. Autres (Sigmoïde, RBF…)

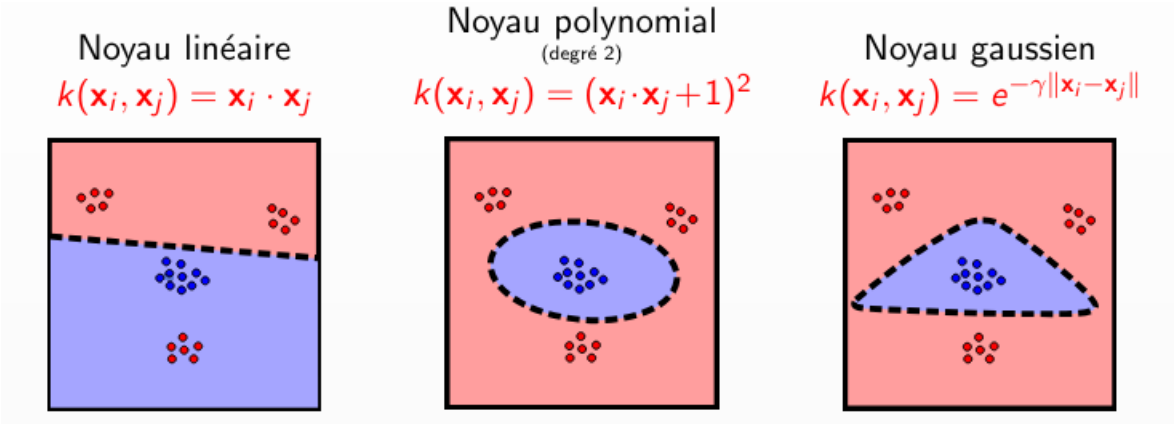

Figure 16: les différents types des noyaux

7.2.2.1 Modélisation

- **Importations :** Il importe les bibliothèques nécessaires, notamment NumPy pour les calculs numériques, Matplotlib pour les graphiques et scikit-learn (sklearn) pour les fonctionnalités d'apprentissage automatique.
- **Division des données** : Il suppose que **'X'** contient vos données de caractéristiques et **'y'** contient les étiquettes correspondantes. La fonction train_test_split de scikit-learn est généralement utilisée pour diviser les données en ensembles d'entraînement et de test. Cependant, dans ce code, cette fonction semble manquer, et 'X ' et 'y ' sont utilisés directement, ce qui implique que la division est effectuée ailleurs ou que X et y représentent déjà les données d'entraînement.
- **Entraînement du modèle SVM :** Il itère sur différents types de noyaux (linéaire, polynomial, RBF et sigmoïde), initialise des modèles SVM avec chaque type de noyau, puis évalue les performances de chaque modèle à l'aide de la validation croisée **(cross_val_score).** La validation croisée est une technique pour évaluer la façon dont les résultats d'une analyse statistique se généraliseront à un ensemble de données indépendant. Le paramètre cv=5 spécifie une validation croisée à 5 plis.
- **Évaluation des performances :** Il imprime les scores de validation croisée pour chaque noyau et calcule le score moyen. Il garde une trace du noyau avec le score moyen le plus élevé **(best_kernel).**

```
from sklearn import svm


# Entraîner des modèles SVM avec différents noyaux.
kernels = ['linear', 'poly', 'rbf', 'sigmoid']
accuracies = []
svm_best_accuracy = 0
best_kernel = None
svm_best_mean_score = 0

for kernel in kernels:
    model = svm.SVC(kernel=kernel)

    scores = cross_val_score(model,  X, y, cv=5)  # cv=5 means 5-fold cross-validation
    print("Kernel " + kernel +" Cross-Validation Scores:", scores)
    mean_score = scores.mean()
    accuracies.append(mean_score)
    if mean_score > svm_best_mean_score:
      svm_best_mean_score = mean_score
      best_kernel = kernel

# affichage les scores d'accuracy
plt.figure(figsize=(6, 4))
plt.plot(range(1, len(kernels) + 1), accuracies, marker='o', linestyle='-')
plt.title('la moyonne des différents noyaux SVM')
plt.xlabel('Noyau SVM')
plt.ylabel('Mean_score')
plt.xticks(range(1, len(kernels) + 1), kernels)
plt.grid(True)
plt.show()
```

Figure 17: le code du modèle final SVM

Après avoir exploré différents types de noyaux SVM tels que linéaire, polynomial, RBF et sigmoïde, et calculé le score moyen pour chaque noyau afin de déterminer le meilleur paramètre avec le score le plus élevé, dans ce cas, le noyau, nous entraînons notre modèle SVM avec ce noyau choisi, ici, par exemple, le noyau linéaire.

```
Kernel linear Cross-Validation Scores: [0.85714286 0.81932773 0.86554622 0.76470588 0.78151261]
Kernel poly Cross-Validation Scores: [0.70168067 0.66806723 0.79411765 0.68067227 0.67226891]
Kernel rbf Cross-Validation Scores: [0.68067227 0.67226891 0.79411765 0.67226891 0.64285714]
Kernel sigmoid Cross-Validation Scores: [0.50840336 0.45378151 0.51260504 0.42436975 0.48319328]
```

Figure 18: Calcul de Score Moyenne de cross validation svm

Voici une figure qui permet de visualiser facilement le résultat pour la discision de noyau qui mention le moyenne score en fonction de noyaux.

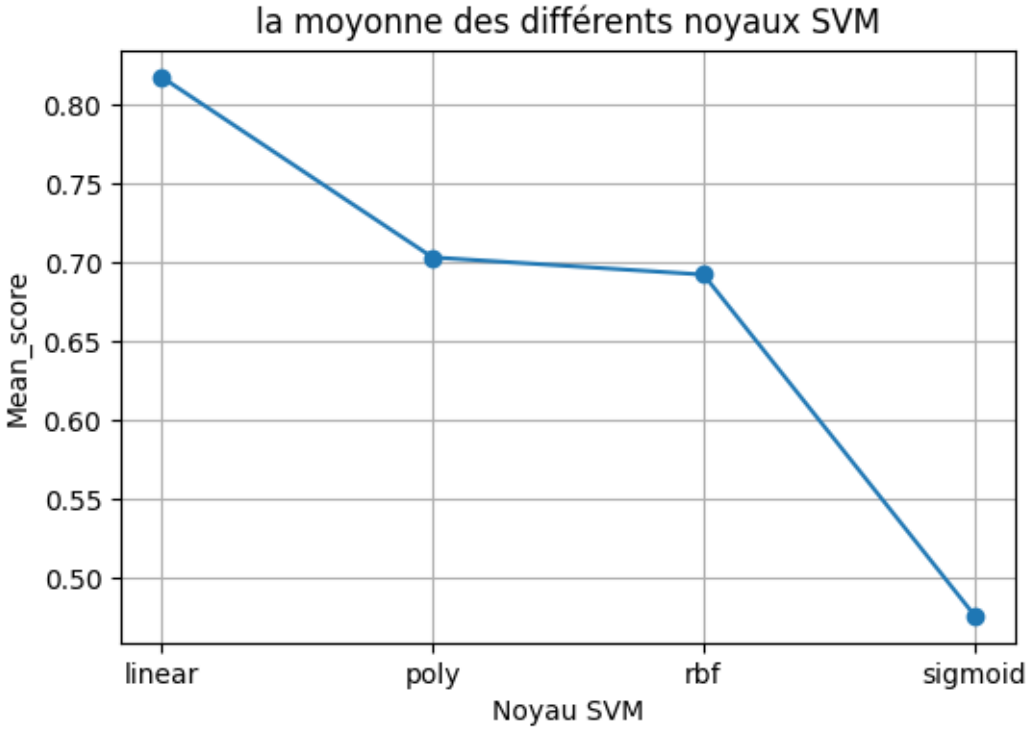

Figure 19: cross validation moyenne score en fonction de noyau

7.2.2.2 Entrainement final de modèle.

En utilise la bibliothèque scikit-learn pour créer un modèle SVM avec le meilleur noyau choisi précédemment. Ensuite, il entraîne ce modèle sur les données d'entraînement (X_{train} et Y_{train}) et prédit les étiquettes sur les données de test (X_{test}).

En utilisant la fonction **accuracy_score** de scikit-learn, il calcule la précision du modèle SVM avec le noyau choisi en comparant les étiquettes prédites (y_{pred_linear}) avec les étiquettes réelles (Y_{test}). Ensuite, il imprime la précision du modèle.

Enfin, il utilise la bibliothèque Seaborn pour tracer la matrice de confusion sous forme de carte thermique à l'aide de la fonction **heatmap**. Cela fournit une représentation visuelle de la matrice de confusion, où les valeurs plus élevées sont indiquées par des couleurs plus claires. Cela aide à visualiser rapidement les performances du modèle en identifiant les classes mal classées.

```python
from sklearn import svm
from sklearn.metrics import accuracy_score

# noyau Linear
model_linear = svm.SVC(kernel=best_kernel)
model_linear.fit(X_train, y_train)
y_pred_linear = model_linear.predict(X_test)


accuracy_linear = accuracy_score(y_test, y_pred_linear)
print(best_kernel,"Kernel Accuracy Score:", accuracy_linear)

conf_matrix = confusion_matrix(y_test, y_pred_linear)

# Tracer la matrice de confusion sous forme de carte thermique.
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```

Figure 20: le code du modèle final SVM

Après d'exécuté le code en haut les résultats suivants s'affichent.

```
Linear Kernel Accuracy Score: 0.8529411764705882
```

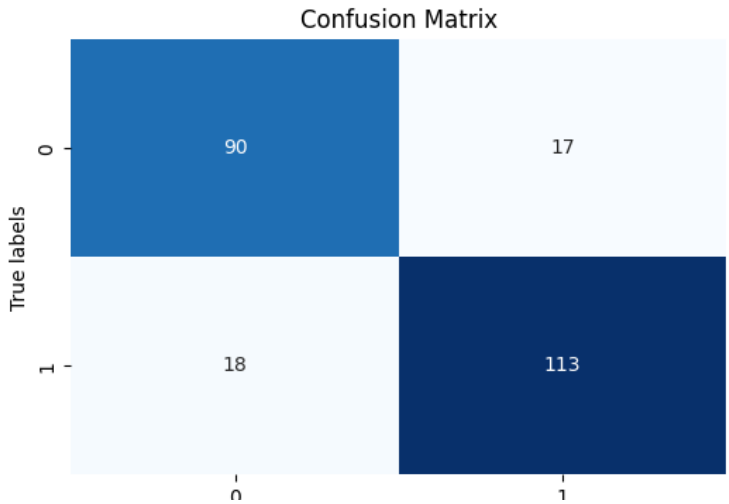

Figure 21 : Confusion Matrice de SVM modèle

7.2.2.3 Interprétation des résultats.

La fonction `confusion_matrix()` de scikit-learn pour évaluer la performance du modèle. La matrice de confusion obtenue est visualisée sous forme de heatmap.

La matrice de confusion en haut montre les résultats suivants :

- **90** Vrais Négatifs : Ce sont les échantillons qui ont été correctement prédits comme négatifs.
- **17** Faux Positifs : Ce sont les échantillons qui ont été incorrectement prédits comme positifs.
- **18** Faux Négatifs : Ce sont les échantillons qui ont été incorrectement prédits comme négatifs.
- **113** Vrais Positifs : Ce sont les échantillons qui ont été correctement prédits comme positifs.

La fonction `accuracy_score` compare ensuite les prédictions du modèle aux vraies valeurs de classe et calcule le pourcentage de prédictions correctes, le résultat est :

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Accuracy} = \frac{90+113}{90+17+18+113} = \frac{203}{238}$$

$$= Accuracy \approx 0.8529$$

Dans l'ensemble, le modèle a une précision de 85,29 %. Cela signifie que le modèle a classé correctement 85,29 % des échantillons de l'ensemble de test.

7.2.3 RandomForest

Un RandomForest (Forêt d'arbres décisionnels) est un algorithme d'apprentissage automatique qui combine plusieurs modèles d'arbres de décision lors de l'apprentissage. Chaque arbre de décision est formé sur une sous-section aléatoire de l'ensemble de données, ce qui apporte de la diversité aux modèles individuels. Lors de la prédiction, chaque arbre donne sa propre prédiction et le résultat final est déterminé par un vote majoritaire (dans le cas de la classification) ou une moyenne (dans le cas de la régression) des prédictions des arbres.

7.2.3.1 Modélisation

- **Importation des bibliothèques** : Les bibliothèques nécessaires sont importées, notamment **RandomForestClassifier** pour créer le modèle RandomForest et cross_val_score pour effectuer la validation croisée.
- **Définition des paramètres à rechercher** : Un dictionnaire params est défini avec les valeurs possibles pour les hyperparamètres du modèle RandomForest, tels que le critère (**criterion**) et le nombre d'estimateurs (**n_estimators**).
- **Initialisation des variables** : Une liste accuracies est initialisée pour stocker les précisions moyennes obtenues pour chaque combinaison de paramètres. Deux autres variables**, best_params** et **rf_best_mean_score**, sont définies pour stocker les meilleurs paramètres et la meilleure précision moyenne obtenue jusqu'à présent.
- **Boucle sur chaque combinaison de paramètres** : Une boucle imbriquée est utilisée pour parcourir toutes les combinaisons de paramètres définies dans params. À chaque itération, un modèle RandomForestClassifier est instancié avec les paramètres actuels, puis la validation croisée est effectuée en utilisant cross_val_score avec 5 plis.
- **Calcul de la précision moyenne** : La moyenne des précisions obtenues pour chaque pli est calculée et stockée dans la variable mean_score.
- **Impression des résultats** : Les précisions obtenues pour chaque combinaison de paramètres sont imprimées à la console pour l'observation. Les précisions moyennes sont également ajoutées à la liste accuracies.
- **Recherche des meilleurs paramètres** : À chaque itération, les paramètres et la précision moyenne sont comparés aux meilleurs paramètres et à la meilleure précision enregistrés jusqu'à présent. Si la précision moyenne actuelle est meilleure que la meilleure précision enregistrée jusqu'à présent, les meilleurs paramètres sont mis à jour.

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score

params = {
    'criterion': ['gini', 'entropy'],
    'n_estimators': [100, 200, 300, 400]
}

accuracies = []
best_params = {}
rf_best_mean_score = 0

# Loop over each parameter combination
for criterion in params['criterion']:
    for n_estimators in params['n_estimators']:

        # Instantiate the RandomForestClassifier model with current parameters
        rf_model = RandomForestClassifier(criterion=criterion, n_estimators=n_estimators, random_state=33)

        # Perform cross-validation
        scores = cross_val_score(rf_model,  X, y, cv=5)

        # Calculate the mean accuracy
        mean_score = scores.mean()

        print("Criterion:", criterion, "n_estimators:", n_estimators, "Cross-Validation Scores:", scores, "Mean Score:", mean_score)

        accuracies.append(mean_score)
        if mean_score > rf_best_mean_score:
            rf_best_mean_score = mean_score
```

```
            best_params = {'criterion': criterion, 'n_estimators': n_estimators}

# Plotting the results
plt.figure(figsize=(10, 6))
plt.plot(range(len(accuracies)), accuracies, color='blue', linestyle='dashed', marker='o', markerfacecolor='red', markersize=10)
plt.title('Accuracy vs. Parameter Combination Index')
plt.xlabel('Parameter Combination Index')
plt.ylabel('Accuracy')
plt.xticks(np.arange(len(accuracies)), rotation=45)
plt.grid()
plt.show()

print("Best Parameters:", best_params)
print("Best Score:", rf_best_mean_score)
```

Figure 22: le code de modélisation RandomForest

Après avoir parcouru chaque combinaison de critères (gini et entropy) et de nombres d'estimateurs (100, 200, 300 et 400) pour le modèle RandomForestClassifier, et calculé la précision moyenne pour chaque combinaison afin de déterminer les meilleurs paramètres avec la précision la plus élevée, nous entraînons notre modèle RandomForest avec ces paramètres choisis.

```
Criterion: gini n_estimators: 100 Cross-Validation Scores: [0.88235294 0.88235294 0.95798319 0.93277311 0.97058824] Mean Score: 0.9252100840336135
Criterion: gini n_estimators: 200 Cross-Validation Scores: [0.8907563  0.88235294 0.95378151 0.93277311 0.96638655] Mean Score: 0.9252100840336135
Criterion: gini n_estimators: 300 Cross-Validation Scores: [0.90336134 0.88655462 0.96218487 0.92436975 0.96638655] Mean Score: 0.9285714285714286
Criterion: gini n_estimators: 400 Cross-Validation Scores: [0.89915966 0.8907563  0.95798319 0.93277311 0.96638655] Mean Score: 0.9294117647058824
Criterion: entropy n_estimators: 100 Cross-Validation Scores: [0.89915966 0.89495798 0.97058824 0.92436975 0.96218487] Mean Score: 0.9302521008403362
Criterion: entropy n_estimators: 200 Cross-Validation Scores: [0.91176471 0.8907563  0.97058824 0.93277311 0.96638655] Mean Score: 0.934453781512605
Criterion: entropy n_estimators: 300 Cross-Validation Scores: [0.90756303 0.89495798 0.96218487 0.92436975 0.96638655] Mean Score: 0.9310924369747899
Criterion: entropy n_estimators: 400 Cross-Validation Scores: [0.90756303 0.89495798 0.96218487 0.92436975 0.96638655] Mean Score: 0.9310924369747899
```

Figure 23 : Calcul de Score Moyenne de cross validation RandomForest

Voici une figure qui permet de visualiser facilement le résultat pour la discision des paramétrer qui mention le moyenne score en fonction de chaque combinaison.

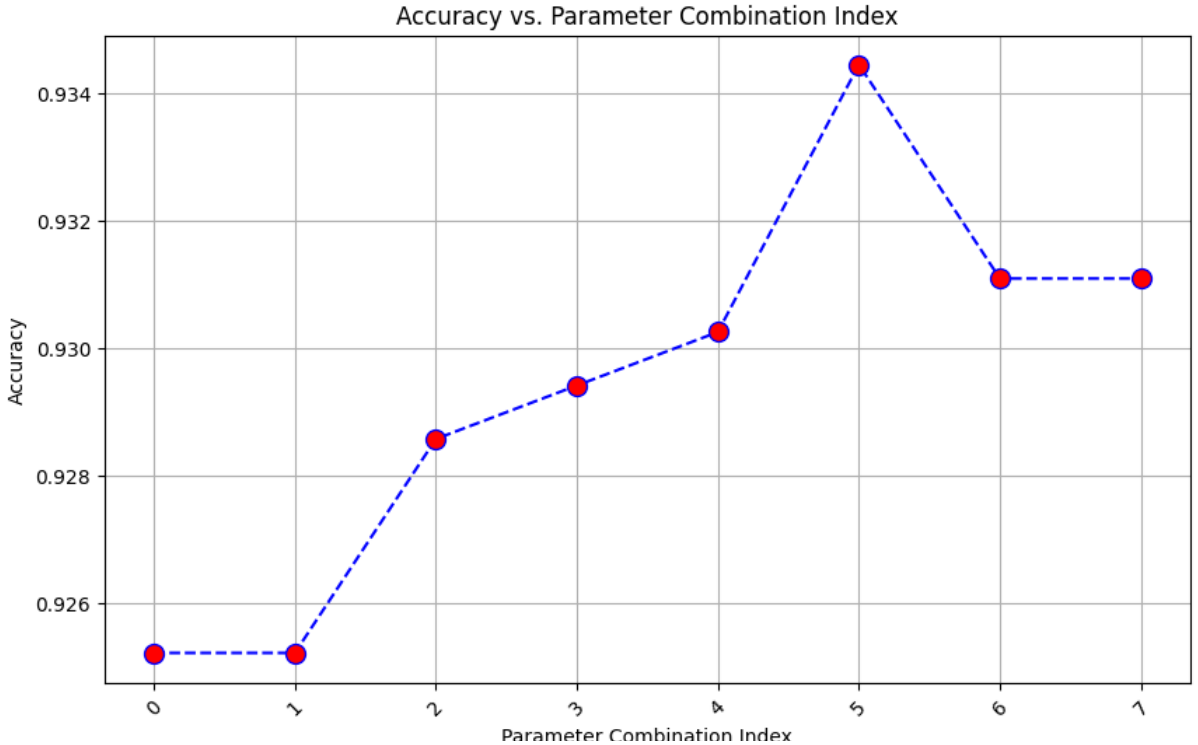

Figure 24: cross validation moyenne score en fonction de combinaisons

7.2.3.2 Entrainement final de modèle.

- **Importation de la classe RandomForestClassifier** : La classe RandomForestClassifier est importée à partir de la bibliothèque scikit-learn.
- **Instanciation du modèle RandomForestClassifier** : Un objet RandomForestClassifier est créé avec les paramètres spécifiés, tels que le critère ('entropy') et le nombre d'estimateurs (200). Le paramètre random_state est également défini pour garantir la reproductibilité des résultats.
- **Entraînement du modèle** : Le modèle RandomForestClassifier est entraîné sur l'ensemble de données d'entraînement X_train avec les étiquettes correspondantes y_train à l'aide de la méthode fit().
- **Prédiction sur l'ensemble de données de test** : Le modèle entraîné est utilisé pour prédire les étiquettes de classe pour l'ensemble de données de test X_test à l'aide de la méthode predict().
- **Calcul de l'exactitude** : L'exactitude du modèle est calculée en comparant les étiquettes prédites avec les étiquettes réelles de l'ensemble de données de test à l'aide de la fonction accuracy_score().
- **Affichage de l'exactitude** : L'exactitude calculée est affichée à l'aide de la fonction print().
- **Calcul et affichage de la matrice de confusion** : La matrice de confusion est calculée à partir des étiquettes réelles et prédites à l'aide de la fonction confusion_matrix() de scikit-learn. Ensuite, la matrice de confusion est affichée sous forme de heatmap à l'aide de la bibliothèque Seaborn pour une visualisation plus claire des performances du modèle.

```
from sklearn import svm
from sklearn.metrics import accuracy_score

# noyau Linear
model_linear = svm.SVC(kernel=best_kernel)
model_linear.fit(X_train, y_train)
y_pred_linear = model_linear.predict(X_test)


accuracy_linear = accuracy_score(y_test, y_pred_linear)
print(best_kernel,"Kernel Accuracy Score:", accuracy_linear)

conf_matrix = confusion_matrix(y_test, y_pred_linear)

# Tracer la matrice de confusion sous forme de carte thermique.
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```

Figure 25: le code du modèle final RandomForest

Après d'exécuté le code en haut les résultats suivants s'affichent.

```
Accuracy Score: 0.957983193277311
```

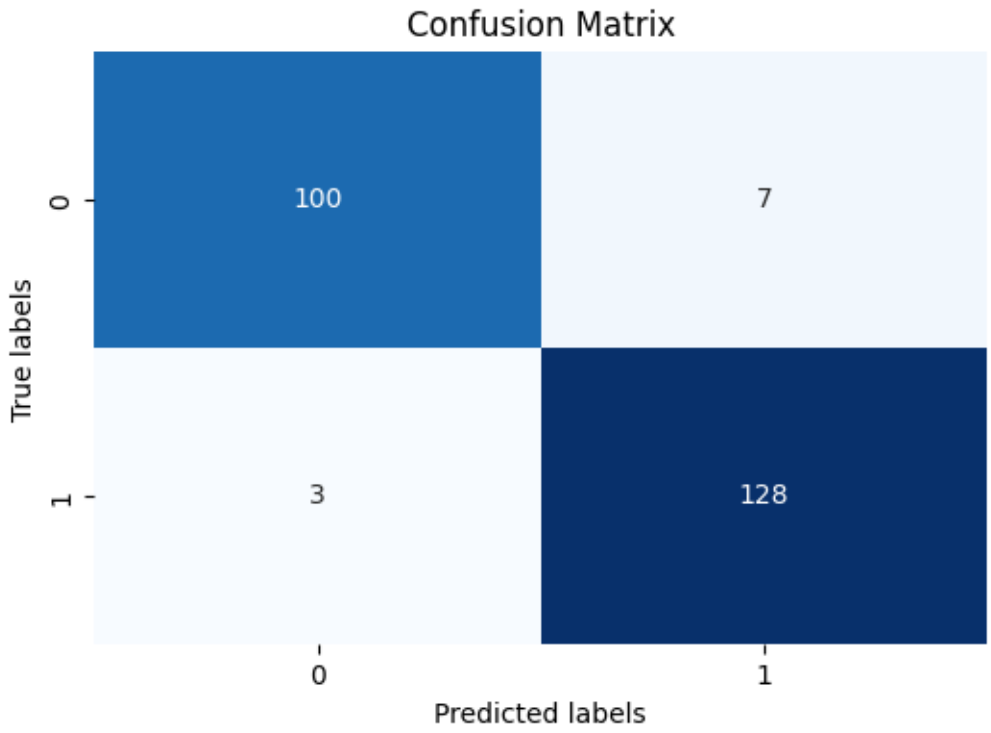

Figure 26: Confusion Matrix de RandomForest modéle

7.2.3.3 Interprétation des résultats.

La fonction `confusion_matrix()` de scikit-learn pour évaluer la performance du modèle. La matrice de confusion obtenue est visualisée sous forme de heatmap.

La matrice de confusion en haut montre les résultats suivants :

- **100** Vrais Négatifs : Ce sont les échantillons qui ont été correctement prédits comme négatifs.
- **7** Faux Positifs : Ce sont les échantillons qui ont été incorrectement prédits comme positifs.
- **3** Faux Négatifs : Ce sont les échantillons qui ont été incorrectement prédits comme négatifs.
- **128** Vrais Positifs : Ce sont les échantillons qui ont été correctement prédits comme positifs.

La fonction `accuracy_score` compare ensuite les prédictions du modèle aux vraies valeurs de classe et calcule le pourcentage de prédictions correctes, le résultat est :

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Accuracy} = \frac{100+128}{100+7+3+128} = \frac{228}{238}$$

$$= Accuracy \approx 0.9579$$

Dans l'ensemble, le modèle a une précision de 95,79 %. Cela signifie que le modèle a classé correctement 95,79 % des échantillons de l'ensemble de test.

Le score de précision du SVM est de 0,95, ce qui signifie que le classificateur SVM a correctement classifié 95 % des points de données. C'est un score de précision très élevé, ce qui suggère que le classificateur SVM est bon pour prédire les étiquettes des points de données.

8. PARAGRAPHE 3 :

Phase Analyse :

– Analyse comparative

8.1 Explication

Analyser les résultats et discuter des avantages et des inconvénients des algorithmes en termes de précision, de vitesse d'exécution et de capacité à généraliser sur de nouvelles données :

- **Précision** : Certains algorithmes offrent une précision élevée sur des ensembles de données complexes, tandis que d'autres sont plus adaptés à des ensembles de données simples.
- **Vitesse d'exécution** : La vitesse peut varier considérablement entre les algorithmes, certains étant beaucoup plus rapides que d'autres.
- **Capacité à généraliser** : Certains algorithmes peuvent mieux généraliser sur de nouvelles données que d'autres, ce qui est crucial pour les performances à long terme.

8.2 Tâche

8.2.1 Précision (Accuracy)

- L'algorithme k-plus proches voisins (KNN) affiche une précision de 0,8067. Ce score indique que KNN a correctement classé environ 80,67 % des échantillons dans l'ensemble de test. Bien que KNN soit une méthode relativement simple, sa performance peut être limitée par sa sensibilité aux données bruitées et à la dimensionnalité élevée, ce qui peut affecter sa capacité à généraliser sur de nouvelles données.

- L'algorithme avec noyau linéaire affiche une précision de 0,8529. Cette précision supérieure peut être attribuée à la capacité des machines à vecteurs de support (SVM) avec un noyau linéaire à trouver des frontières de décision linéaires efficaces dans l'espace des caractéristiques. Cependant, les SVM peuvent être sensibles aux paramètres et nécessitent parfois un prétraitement des données pour obtenir de meilleurs résultats.

- L'algorithme RandomForest affiche la meilleure précision parmi les trois, avec un score de 0,9579. RandomForest est une méthode d'ensemble qui combine plusieurs arbres de décision pour améliorer la précision et réduire le surajustement. Sa capacité à capturer des relations complexes entre les caractéristiques et les étiquettes des données d'entraînement contribue à sa forte performance en termes de précision. Cependant, RandomForest peut être plus lent que d'autres algorithmes en raison de la construction et de l'agrégation de multiples arbres de décision.

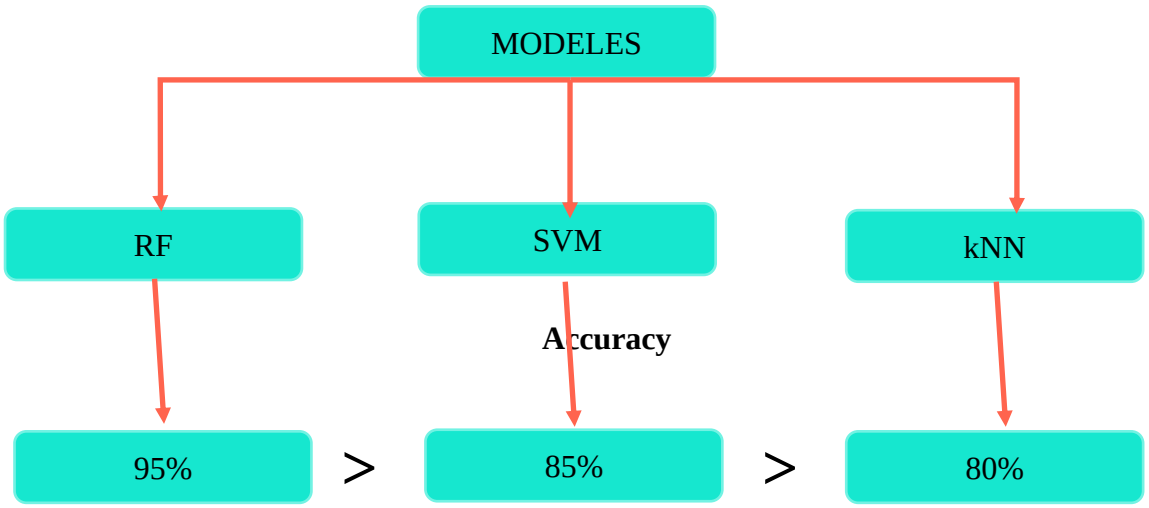

Figure 27: Comparaison d'accuracy

8.2.2 Vitesse d'exécution

- K-plus proches voisins (KNN) est connu pour être relativement lent lorsqu'il s'agit de prédire de nouvelles données. Cette lenteur est due à sa nature de calcul intensif, car il doit calculer les distances entre la nouvelle observation et tous les échantillons d'entraînement à chaque prédiction. Par conséquent, KNN peut être moins adapté aux applications nécessitant des prédictions rapides sur de grandes quantités de données.

- Les machines à vecteurs de support (SVM) avec un noyau linéaire sont généralement plus rapides que KNN. En utilisant des techniques d'optimisation efficaces, les SVM peuvent trouver la frontière de décision linéaire en un temps relativement court une fois qu'ils sont

entraînés. Cependant, la vitesse d'exécution des SVM peut varier en fonction de la taille de l'ensemble de données et du nombre de caractéristiques.

- RandomForest peut être plus lent que les deux autres algorithmes en raison de sa nature d'ensemble. La construction et l'agrégation de multiples arbres de décision peuvent nécessiter plus de temps de calcul, en particulier sur de grandes quantités de données ou avec des paramètres d'arbre complexes. Cependant, les implémentations parallèles et les techniques d'optimisation peuvent être utilisées pour accélérer l'exécution de RandomForest dans certains cas.

CONCLUSION :

En conclusion, l'analyse comparative des algorithmes a permis de mettre en évidence leurs forces et leurs faiblesses dans différents aspects clés tels que la précision, la vitesse d'exécution et la capacité à généraliser sur de nouvelles données.

Premièrement, il est apparu que RandomForest a obtenu la meilleure précision parmi les trois algorithmes examinés, avec un score de 0,9579. Cela souligne l'efficacité de cette méthode d'ensemble dans la classification précise des données, notamment en capturant les relations complexes entre les caractéristiques et les étiquettes.

Deuxièmement, en termes de vitesse d'exécution, les machines à vecteurs de support (SVM) avec un noyau linéaire se sont avérées être les plus rapides parmi les trois. Bien qu'ils offrent une précision élevée, leur temps d'exécution relativement court les rend appropriés pour des applications nécessitant des résultats rapides.

Troisièmement, bien que l'algorithme k-plus proches voisins (KNN) ait montré une précision raisonnable, sa lenteur lors de la prédiction de nouvelles données en fait une option moins attrayante pour les applications nécessitant une exécution rapide.

En résumé, le choix de l'algorithme dépendra des exigences spécifiques de chaque application, en tenant compte du compromis entre la précision, la vitesse d'exécution et la capacité de généralisation. Une évaluation soigneuse des performances sur les données réelles de l'application est essentielle pour choisir l'algorithme le plus approprié pour répondre aux besoins spécifiques de la tâche.