UFR de Sciences Appliquées et de Technologie

Section : Informatique

Niveau : Master 2 GDIL

EC : Data Mining

UE : Extraction de connaissances

Projet de TP : Etude comparative des modèles de classification

Présenté par :

- Faback Dieng (P26 830)
- Cheikhna Diaw (P31 5381)
- El hadji Leye (P27 11 20)

Doctorant :

✓ Mr KOUA

Année académique : 2022-2023

Plan

Liste des figures

I. INTRODUCTION

L'évolution rapide des technologies a amplifié la pertinence de l'analyse de données dans la prise de décisions éclairées. Dans ce contexte, les modèles de classification jouent un rôle prépondérant en permettant la catégorisation efficace et précise des données dans des domaines aussi variés que la finance, la santé, ou encore la gestion des ressources.

Cette étude comparative s'inscrit dans une volonté d'évaluer et de confronter quatre modèles de classification fondamentaux : la **régression logistique**, l'**arbre de classification**, **Naïve Bayes** et le **Support Vector Machine (SVM).** L'objectif principal réside dans l'identification du modèle offrant les performances les plus adaptées à des données spécifiques, afin de guider les choix et les stratégies d'analyse dans divers contextes.

A travers l'exploration de ces modèles, cette étude vise à fournir un éclairage approfondi sur leurs fonctionnements respectifs, leurs forces et leurs limitations, offrant ainsi une base pour des décisions éclairées dans le cadre d'applications concrètes. Pour cela, nous allons travailler sur un ensemble de données issu du domaine médical, offrant une diversité de caractéristiques, et nous avons évalué les performances de chaque modèle selon des critères établis.

Cette introduction sert de porte d'entrée à une analyse détaillée et comparative des modèles, mettant en lumière leurs performances et leurs implications dans la classification de données réelles. En fin de compte, cette étude vise à apporter des éléments concrets et pratiques pour guider le choix du modèle le plus adapté à des besoins spécifiques en matière de classification de données.

II. PRESENTATION DES MODELES DE CLASSIFICATION

1. REGRESSION LOGISTIQUE

La régression logistique est un algorithme supervisé de classification. Elle mesure la relation entre la variable dépendante catégorielle et une ou plusieurs variables indépendantes en donnant une estimation à la probabilité d'occurrence d'un évènement à travers l'usage de sa fonction logistique.

2. ARBRE DE CLASSIFICATION

Un arbre de décision ou de classification permet d'expliquer une variable cible à partir d'autres variables dites explicatives. L'algorithme permet de déterminer la meilleure caractéristique dans

l’ensemble de données, diviser les données en sous-ensembles contenant les valeurs possibles de la meilleure caractéristique, générer de manière récursive de nouveaux arbres de décision en utilisant les sous-ensembles de données crées et lorsqu’on ne peut plus classifier les données, on s’arrête.

3. NAÏVE BAYES

La méthode Naïve Bayes est une technique de classification simple et efficace, souvent utilisée dans divers domaines, notamment le traitement du langage naturel et la classification de documents. Elle repose sur le théorème de Bayes et suppose une indépendance conditionnelle entre les caractéristiques. Malgré sa simplicité et ses hypothèses simplificatrices, Naïve Bayes peut souvent produire des résultats surprenamment bons, en particulier pour des ensembles de données de taille modérée à grande. Cependant, sa performance peut être affectée si les variables sont fortement corrélées ou si les hypothèses d'indépendance conditionnelle ne sont pas vérifiées.

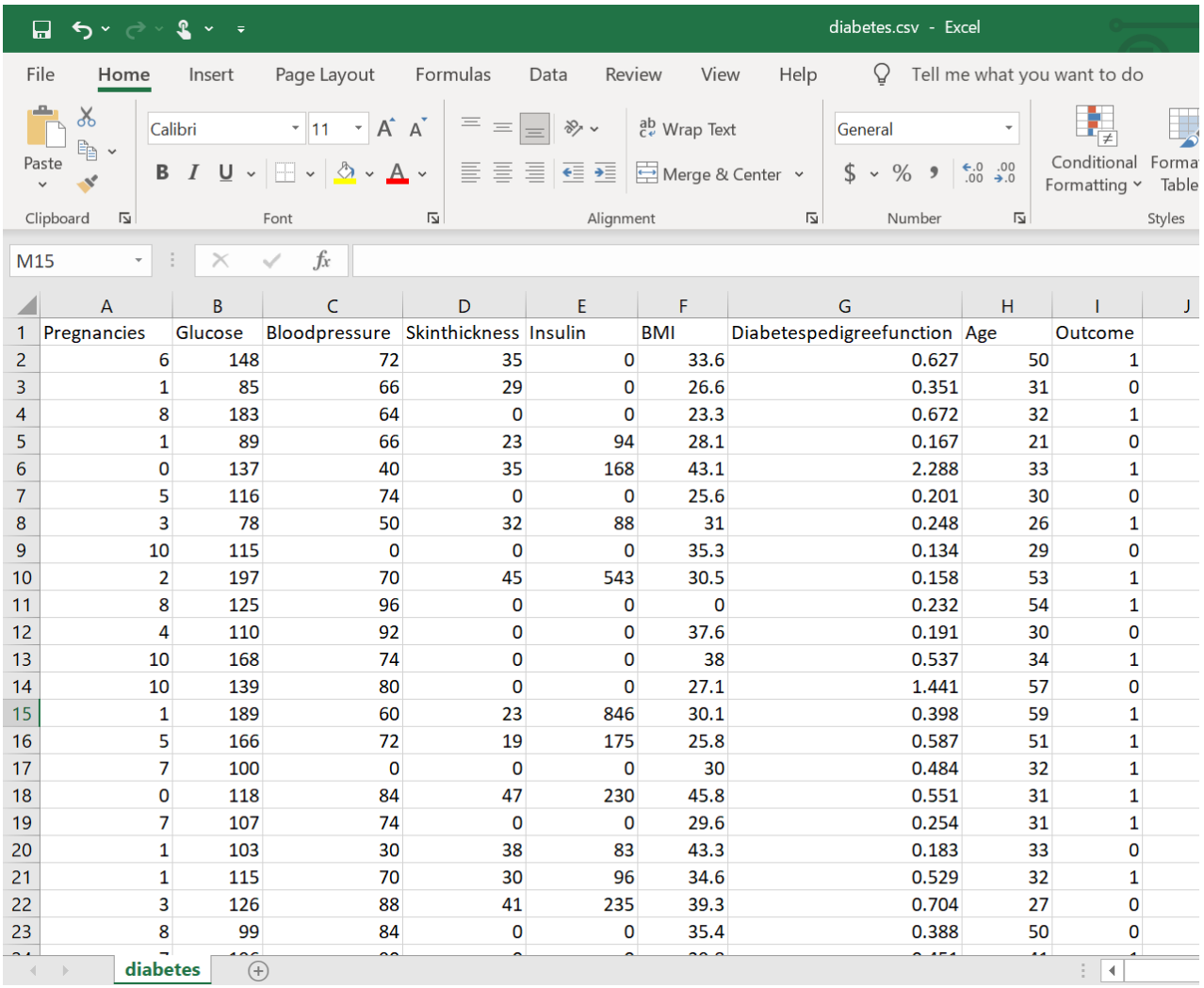
4. SVM (SUPPORT VECTOR MACHINE)

Le SVM est l’un des modèles les plus populaires de l’apprentissage automatique. Il appartient à la famille d’algorithme d’apprentissage supervisé. Son but est de séparer les données en classes.

Dans la suite de notre, nous allons faire une étude de ces quatre modèles.

III. ETUDE DES MODELES DE CLASSIFICATION

Avant de plonger dans l'évaluation de ces modèles, nous avons défini une méthodologie rigoureuse pour comparer leur performance. Nous avons utilisé l’ensemble de données fichier diabetes.csv (voir figure 1), divisé plus tard de manière appropriée en ensembles d'entraînement et de test pour évaluer chaque modèle.



diabetes.csv - Excel

File Home Insert Page Layout Formulas Data Review View Help Tell me what you want to do

Paste Clipboard Calibri 11 Font Wrap Text Merge & Center Alignment General Number Conditional Formatting Styles

M15

	A	B	C	D	E	F	G	H	I	J
1	Pregnancies	Glucose	Bloodpressure	Skinthickness	Insulin	BMI	Diabetespedigreefunction	Age	Outcome	
2	6	148	72	35	0	33.6	0.627	50	1	
3	1	85	66	29	0	26.6	0.351	31	0	
4	8	183	64	0	0	23.3	0.672	32	1	
5	1	89	66	23	94	28.1	0.167	21	0	
6	0	137	40	35	168	43.1	2.288	33	1	
7	5	116	74	0	0	25.6	0.201	30	0	
8	3	78	50	32	88	31	0.248	26	1	
9	10	115	0	0	0	35.3	0.134	29	0	
10	2	197	70	45	543	30.5	0.158	53	1	
11	8	125	96	0	0	0	0.232	54	1	
12	4	110	92	0	0	37.6	0.191	30	0	
13	10	168	74	0	0	38	0.537	34	1	
14	10	139	80	0	0	27.1	1.441	57	0	
15	1	189	60	23	846	30.1	0.398	59	1	
16	5	166	72	19	175	25.8	0.587	51	1	
17	7	100	0	0	0	30	0.484	32	1	
18	0	118	84	47	230	45.8	0.551	31	1	
19	7	107	74	0	0	29.6	0.254	31	1	
20	1	103	30	38	83	43.3	0.183	33	0	
21	1	115	70	30	96	34.6	0.529	32	1	
22	3	126	88	41	235	39.3	0.704	27	0	
23	8	99	84	0	0	35.4	0.388	50	0	

diabetes

Figure 1 : Ensemble de données fichier diabetes.csv

Nous avons, par la suite de notre travail, fait l'étude comparative de ces modèles.

Pour cela, nous avons commencé par installer toutes les bibliothèques nécessaires à savoir pandas, scikit-learn et matplotlib par la commande : ! pip install nom_bibliothèque.

```
# Installation de Pandas : Une bibliothèque pour la manipulation et l'analyse des données.
! pip install pandas
```

Figure 2 : Installation de Pandas

```
# Installation de Scikit-Learn(sklearn) : Une bibliothèque pour l'apprentissage automatique qui inclut divers algorithmes
# et outils pour l'analyse de données.
! pip install scikit-learn
```

Figure 3 : Installation de Scikit-learn

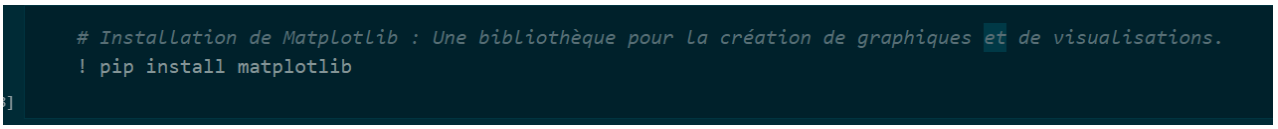


Figure 4 : Installation de Matplotlib

Ainsi, nous avons importé toutes les librairies indispensables pour notre étude.

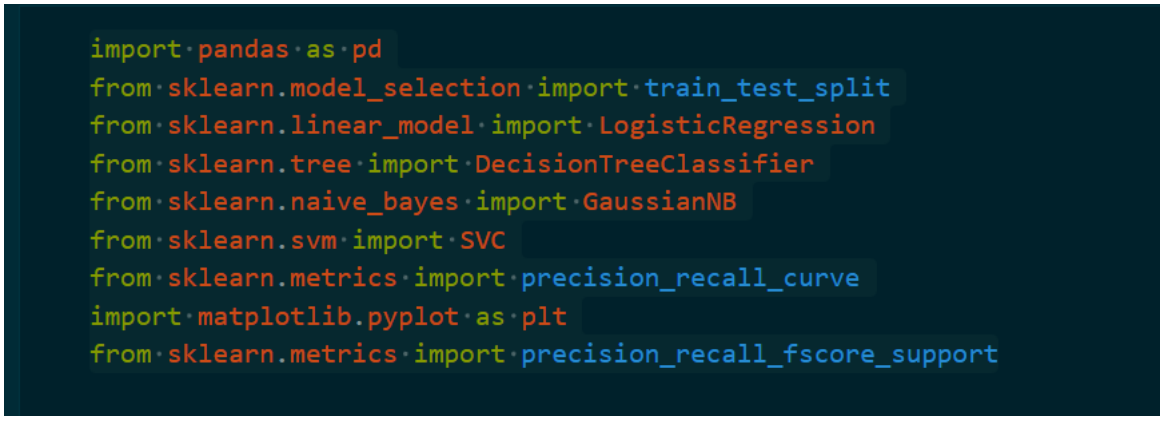


Figure 5 : Importation des librairies

Pour pouvoir manipuler l'ensemble des données (figure 1), il faut au préalable le charger (dans une variable) (figure 6). Puis nous avons vérifié si toutes les données sont bien chargées (voir le couple (768, 9) sur la figure 7) et aussi affiché toutes les colonnes de notre ensemble de données ainsi que toutes les valeurs uniques présentes dans ce dernier ce qui est utile pour comprendre la variété des données dans chaque colonne et évaluer la diversité des informations qu'elles contiennent (figure 8). Ensuite nous avons affiché l'ensemble de données notre dataframe (figure 9).

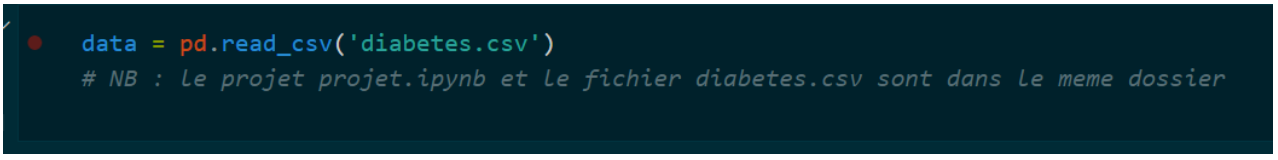


Figure 6 : Chargement de l'ensemble des données fichier diabetes.csv

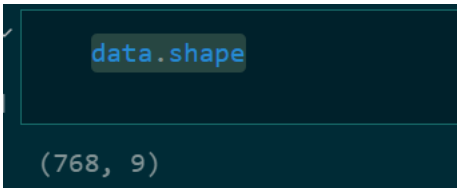


Figure 7 : Afficher les dimensions de notre ensemble de données

```
for col in data.columns:
    print(col, len(data[col].unique()))
```

```
Pregnancies 17
Glucose 136
Bloodpressure 47
Skinthickness 51
Insulin 186
BMI 248
Diabetespedigreefunction 517
Age 52
Outcome 2
```

Figure 8 : Affichage des colonnes ainsi que leur nombre de valeurs uniques présentes

```
# Affichage des données
data
```

	Pregnancies	Glucose	Bloodpressure	Skinthickness	Insulin	BMI	Diabetespedigreefunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

Figure 9 : Afficher notre ensemble de données (Dataframe)

Après cela, nous avons fait une séparation des variables prédictives et de la variable cible.

Les variables prédictives sont extraites du jeu de données initial en supprimant la colonne "Outcome". **X = data.drop('Outcome', axis=1):** Cette ligne crée un DataFrame X en prenant toutes les colonnes de 'data' sauf la colonne 'Outcome'. Ainsi, X contient toutes les variables prédictives.

y = data['Outcome'] : Cette ligne crée une série y qui contient uniquement la colonne 'Outcome' de la variable 'data'. Cette colonne 'Outcome' est considérée comme la variable cible.

Le code sur la figure 10 permet de séparer les variables prédictives (dans X) de la variable cible (dans y) à partir du DataFrame 'data' et affiche un aperçu des premières lignes de chaque ensemble

(X et y) pour une visualisation rapide des données.

```
# Séparation des variables prédictives et de la variable cible
X = data.drop('Outcome', axis=1)
y = data['Outcome']
# Affichage des variables X et y
print("X:")
# Affichage des premières lignes de X
print(X.head())
# Affichage des premières lignes de y
print("\ny:")
print(y.head())
```

Figure 10 : Séparation des variables prédictives et de la variable cible

Pour la suite de notre travail, nous avons fait une division des données en ensembles d'entrainement et de test.

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42): Utilise la fonction **train_test_split** pour diviser les données X (variables prédictives) et y (variable cible) en ensembles d'entraînement et de test. Les paramètres **test_size**=0.2 spécifient que 20% des données seront utilisées comme ensemble de test et les 80% qui restent comme ensemble d'entrainement, tandis que **random_state**=42 fixe la graine aléatoire pour assurer la reproductibilité. Puis nous avons fait un affichage des premières et des nombres lignes des ensembles d'entrainement et de test (figure 11) dans le but de vérifier la répartition des données entre les ensembles d'entraînement et de test.

```python
# Division des 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)

# Affichage des variables
print("X_train:")
print(X_train.head())  # Affichage des premières lignes de X_train
print("\nX_test:")
print(X_test.head())  # Affichage des premières lignes de X_test
print("\ny_train:")
print(y_train.head())  # Affichage des premières lignes de y_train
print("\ny_test:")
print(y_test.head())  # Affichage des premières lignes de y_test

# Afficher le nombre de lignes de chaque variable
print(f"Nombre de lignes dans X_train: {len(X_train)}")
print(f"Nombre de lignes dans X_test: {len(X_test)}")
print(f"Nombre de lignes dans y_train: {len(y_train)}")
print(f"Nombre de lignes dans y_test: {len(y_test)}")
```

Figure 11 : Division des données en ensembles d'entrainement et de test

Ensuite nous avons fait une initialisation des quatre modèles :

- ✓ Régression Logistique ;
- ✓ Arbre de classification ;
- ✓ Naïve Bayes;
- ✓ SVM (Support Vector Machine).

```python
# Initialisation des modèles
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'Decision Tree': DecisionTreeClassifier(),
    'Naive Bayes': GaussianNB(),
    'SVM': SVC(probability=True)
}
```

Figure 12 : Initialisation des modèles

Enfin nous avons fait la prédiction de chaque modèle sur les nouvelles données à travers une boucle « for » à l'intérieur de laquelle on entraine chaque modèle avant de faire la prédiction. Pour chaque modèle contenu dans le dictionnaire **models.items(),** le script entraîne le modèle sur l'ensemble d'entraînement **(X_train, y_train)** et effectue des prédictions sur l'ensemble de test **(X_test)**.

Les prédictions sont affichées pour chaque modèle, ainsi que le nombre de prédictions et la

matrice de confusion pour évaluer les performances de classification (voir figure 13).

Les métriques telles que la précision, le rappel, le F1-score, et l'exactitude (accuracy) sont calculées pour évaluer les performances de chaque modèle (figure 13).

```python
# Prédictions pour chaque modèle sur les nouvelles données
for name, model in models.items():
    model.fit(X_train, y_train)  # Entraîner sur l'ensemble d'entraînement

    # Prédictions sur l'ensemble de test
    predictions = model.predict(X_test)

    # Affichage des prédictions du modèle
    print(f"Prédictions du modèle {name}: {predictions}")

    # Affichage du nombre de prédictions
    print(f"Nombre de prédictions du modèle {name}: {len(predictions)}")

    # Making the Confusion Matrix
    from sklearn.metrics import confusion_matrix
    cm = confusion_matrix(y_test, predictions)
    print(f"La matrice de confusion : ")
    print(cm)

    # Calcul des probabilités prédites pour la courbe de précision-rappel
    y_scores = model.predict_proba(X_test)[:, 1]

    # Calcul des valeurs pour la courbe de précision-rappel
    precision_curve, recall_curve, _ = precision_recall_curve(y_test, y_scores)

    # Affichage des métriques pour chaque modèle
    precision, recall, f1, _ = precision_recall_fscore_support(y_test, predictions, average='binary')

    # Calculer le taux de performance du modèle
    accuracy = model.score(X_test, y_test)
```

Figure 13 : Entrainement et prédiction des données et le calcul des métriques

De plus, le support pour chaque classe est calculé à partir des valeurs de test.

Nous avons ajouté un ensemble de données (Test_Data) pour des prédictions supplémentaires après l'entraînement des modèles. Les classes prédites pour ces données sont également affichées (voir figure 14).

```python
    # Calculer le taux de performance du modèle
    accuracy = model.score(X_test, y_test)

    # Calculer le support pour chaque classe
    support_class_0 = y_test.value_counts()[0]
    support_class_1 = y_test.value_counts()[1]

    print(f"\nMétriques pour le modèle {name}:")
    print(f"Precision: {precision}")
    print(f"Recall: {recall}")
    print(f"F1-score: {f1}")
    print(f"Support - Class 0: {support_class_0}")
    print(f"Support - Class 1: {support_class_1}")
    print(f"Accuracy : {accuracy}\n")

    # Exemple de données à prédire
    Test_Data = [
        [6, 148, 72, 35, 0, 33.6, 0.627, 40],
        [6, 148, 72, 35, 0, 33.6, 0.627, 60]
    ]
    print(f"Les données à prédire : {Test_Data}")
    # Prediction
    predict = model.predict(Test_Data)
    print(f"Les classes prédictes de ces données : {predict}\n")

    # Tracé de la courbe de précision-rappel pour chaque modèle
    plt.plot(recall_curve, precision_curve, label=name)

plt.xlabel('Recall')
plt.ylabel('Precision')
```

Figure 14 : Affichage métriques, le calcul des supports et test sur des données supplémentaires

Et pour finir, nous avons tracé la courbe précision-rappel pour chaque modèle en utilisant les valeurs de précision et de rappel calculées précédemment.

La courbe précision-rappel montre comment la précision et le rappel évoluent pour différentes valeurs seuil de classification. Chaque point de la courbe représente un seuil différent. Un modèle idéal aurait une courbe se rapprochant le plus possible du coin supérieur droit, indiquant une haute précision et un haut rappel pour divers seuils. Cette courbe est particulièrement utile dans les cas où les classes sont déséquilibrées, c'est-à-dire lorsque l'une des classes est beaucoup plus fréquente que l'autre. Elle offre une perspective sur la performance du modèle indépendamment du déséquilibre de classe, en montrant comment le modèle maintient un équilibre entre la précision et le rappel (voir le démo : le code dans son intégralité ; fichier).

```
        print(f"\nMétriques pour le modèle {name}:")
        print(f"Precision: {precision}")
        print(f"Recall: {recall}")
        print(f"F1-score: {f1}")
        print(f"Support - Class 0: {support_class_0}")
        print(f"Support - Class 1: {support_class_1}")
        print(f"Accuracy : {accuracy}\n")

        # Exemple de données à prédire
        Test_Data = [
            [6, 148, 72, 35, 0, 33.6, 0.627, 40],
            [6, 148, 72, 35, 0, 33.6, 0.627, 60]
        ]
        print(f"Les données à prédire : {Test_Data}")
        # Prediction
        predict = model.predict(Test_Data)
        print(f"Les classes prédictes de ces données : {predict}\n")

        # Tracé de la courbe de précision-rappel pour chaque modèle
        plt.plot(recall_curve, precision_curve, label=name)

    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title('Precision-Recall Curve')
    plt.legend()
    plt.show()
```

Figure 15 : La courbe de Précision-Rappel

Ce code (de la figure 13 à la figure 15) est une méthode complète pour entraîner plusieurs modèles de classification, évaluer leurs performances sur un ensemble de test, visualiser les résultats et effectuer des prédictions sur de nouvelles données. Il offre une analyse détaillée de la performance de chaque modèle pour des tâches de classification.

Après l'exécution de ce code, nous avons eu les résultats qui suivent pour le calcul des métriques :

- **Le modèle Régression Logistique :**

```
Métriques pour le modèle Logistic Regression:
Precision: 0.6379310344827587
Recall: 0.6727272727272727
F1-score: 0.6548672566371682
Support - Class 0: 99
Support - Class 1: 55
Accuracy : 0.7467532467532467
```

Figure 16 : Métriques modèle Régression Logistique

❖ **Le modèle Arbre de classification :**

```
Métriques pour le modèle Decision Tree:
Precision: 0.6379310344827587
Recall: 0.6727272727272727
F1-score: 0.6548672566371682
Support - Class 0: 99
Support - Class 1: 55
Accuracy : 0.7467532467532467
```

Figure 17 : Métriques modèle Arbre de classification

❖ **Le modèle Naïve Bayes :**

```
Métriques pour le modèle Naive Bayes:
Precision: 0.6610169491525424
Recall: 0.7090909090909091
F1-score: 0.6842105263157895
Support - Class 0: 99
Support - Class 1: 55
Accuracy : 0.7662337662337663
```

Figure 18 : Métriques modèle Naïve Bayes

❖ **Le modèle SVM (Support Vector Machine) :**

```
Métriques pour le modèle SVM:
Precision: 0.7209302325581395
Recall: 0.5636363636363636
F1-score: 0.6326530612244898
Support - Class 0: 99
Support - Class 1: 55
Accuracy : 0.7662337662337663
```

Figure 19 : Métriques modèle SVM (Support Vector Machine)

IV. CONCLUSION

Régression Logistique et Arbre de Décision : Les performances de la **régression logistique** et de l'**arbre de décision** sont identiques. Leurs précisions, rappels et F1-scores sont équivalents. Cela suggère qu'ils fonctionnent de manière similaire sur cet ensemble de données.

Naïve Bayes : Le modèle **Naïve Bayes** présente une légère amélioration en termes de précision, de rappel et de F1-score par rapport à la **régression logistique** et à l'**arbre de décision**. Cela

pourrait indiquer qu'il prend mieux en compte certaines subtilités ou interactions entre les caractéristiques. Autrement dit, cela suggère qu'il exploite d'une manière différente ou plus efficace les informations contenues dans les données.

SVM : Le modèle **SVM** montre la précision la plus élevée parmi les modèles évalués, mais il a un rappel légèrement inférieur. Cela signifie qu'il est plus précis dans la prédiction de la classe 1 mais pourrait manquer certains exemples positifs par rapport aux autres modèles.

Accuracy : La précision globale (accuracy) est similaire pour le modèle **Naïve Bayes** et le modèle **SVM**, mais le modèle **SVM** a une précision plus élevée pour la classe positive, tandis que le modèle **Naïve Bayes** a un rappel plus élevé pour la même classe.

En fonction de l'objectif spécifique du modèle (maximiser la précision, le rappel ou trouver un équilibre entre les deux), le choix pourrait se faire comme suit :

- Si la précision est primordiale, le **SVM** pourrait être préféré en raison de sa précision plus élevée.
- Si un équilibre entre précision et rappel est crucial, le **Naïve Bayes** peut être un bon compromis.
- Si l'objectif est d'avoir des performances générales stables sans préférence spécifique pour la précision ou le rappel, la **régression logistique** ou l'**arbre de décision** pourraient être des choix raisonnables en raison de leur stabilité.

Le choix final dépendra également de considérations spécifiques au domaine, telles que la tolérance aux faux positifs ou aux faux négatifs dans le contexte d'application.