Rapport TP Introduction à l'IA: Cas de classification

Harouna KINDO

September 2024

Contents

1 Introduction

Dans ce rapport, nous appliquons des méthodes de classification supervisée sur un jeu de données portant sur l'expression des gènes du cancer du côlon. L'objectif est de prédire si un échantillon appartient à la classe "normal" ou "tumoral". Donc c'est un cas de classification binaire alors nous allons utilisé des modèles de classification pour nos prédictions. Mais avant d'en arriver aux prédictions, nous allons d'abord procéder à l'analyse exploratoire de notre jeu de données.

2 Préparation du dataset

Les étapes de préparation des données incluent :

- Chargement des données à l'aide de `pandas`.
- Vérification et nettoyage des données.
- Séparation des données en ensembles d'entraînement et de test.

Pour cette phase de préparation et aussi pour l'exploration de nos données notamment avec des graphiques, nous aurons besoin des bibliothèques pandas, matplotlib et seaborn.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Figure 1: Importation des bibliothèques pour préparation et exploration des données

2.1 Chargement et inspection des données

Comme notre dataset est un fichier csv, nous sommes parvenu à la lire grâce à la fonction read_csv de pandas. Le paramètre sep=";" indique à la fonction la nature du separateur utilisé dans la dataset pour distingé les lignes.

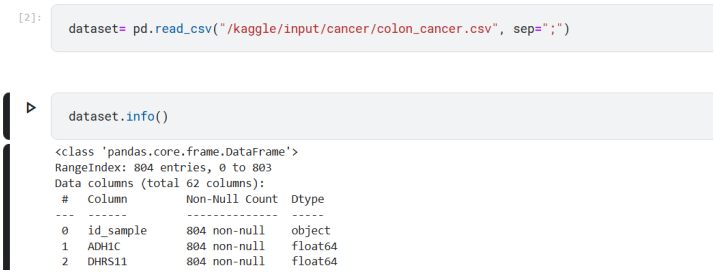

Figure 2: Chargement puis inspection des données

Cette partie du code nous montre la lecture de la dataset puis des informations sur le contenu de cette dataset. En effet, notre dataset compte 804 oberservations 60 features qui sont toutes numériques.

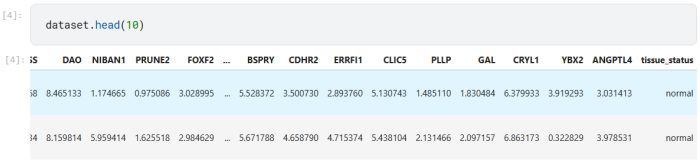

Figure 3: Affichage des données

Cette affichage des données nous permet de confirmer que les fetures sont toutes numériques cependant la target est catégoricielle donc nous allons devoir l'encoder en binaire.

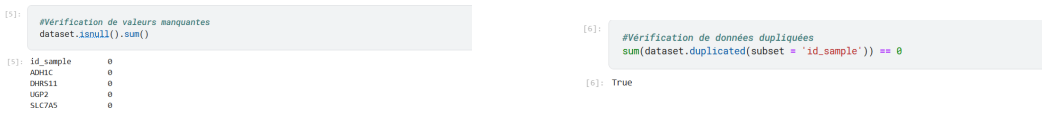

Figure 4: Vérification des données manquantes et dupliquées

Nous remarquons qu'il n'ya ni données manquantes ni données dipluquées dans notre dataset.

Figure 5: Encodage de la target

Grâce à la fonction LabelEncoder de la bibliothèque Sklearn, nous sommes parvenu à transformer la target qui était une variable catégoricielle en variable numérique. Nous rappelons que la fonction LabelEncoder remplace l'ensemble des classe de la variable par des nombre en commençant par 0. Vu que nous avons que deux classes alors nous avons que des 0 et 1 dans notre target maintenant.
Mais avant de faire cette transformation, nous nous assurer de copier notre dataset dans une nouvelle variable. Ainsi,nous aurons toujours une version de la dataset initiale.

2.2 Séparation des features et des labels

Les données sont séparées entre les features (`X`) et la variable cible (`Y`) :

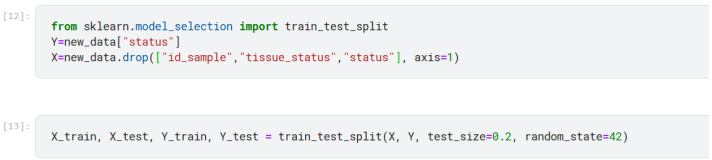

Figure 6: séparation de la dataset

Nous venons ainsi de séparer d'une part nos données en features et en target. Vu que la target est constituée des labels ici qui est représenté par le status des tissues "normal(0)/tumoral(1)" alors notre target est constituée de la colonne "status" qui est la colonne "tissue_status" encodée. Puis les features sont constituées des autres colonnes sauf la colonne des Id.
D'autres part nous avons divisé nos données en données d'entrainement(80%) et de test(20%) grâce à la fonction train_test_split(). Avec la définition

d'un randon state ici 42, je pourrais avoir les mêmes données donc les mêmes resultats à chaque fois que j'exécute mon code.

2.3 Standardisation des données

La standardisation des données est une étape importante dans la préparation des données pour le Machine Learning, surtout lorsque les features ont des unités ou des échelles très différentes. Elle facilite l'apprentissage au modèle et permet à l'algorithme de converger plus rapidement.

Figure 7: standardisation

Un point important avec la standardisation est que ses métrique (Moyenne et Ecart-type) se calcule uniquement sur les données de d'entrainement puis on l'applique aux données d'entrainement et de test. Ici à travers fit_transform(), nous calculons les metriques puis nous l'appliquons aux données d'entrainement et à travers transform(), à l'applique aux données de test.
Tout ceci pour que notre modèle ne puisse pas avoir des informations sur nos données de test. Cela pourrait biaiser la performance du modèle.

3 Exploration des données

Nous analysons la distribution des classes et réalisons des visualisations des gènes. Cette analyse aide à comprendre la structure des données avant d'appliquer des modèles.

3.1 Distribution des classes

Nous avons utilisé `describe()` pour analyser la répartition des classes "normal" et "cancer" :

```
[7]: dataset["tissue_status"].describe()

[7]: count        804
     unique         2
     top       normal
     freq         402
     Name: tissue_status, dtype: object
```

Figure 8: Analyse des classes

A travers cette observation, nous remarquons notre target compte 804 observations puis elle contient vraiment 02 classes et aussi on remarque que le mode(la classe la plus fréquente) est la classe normal avec une fréquence de 402. Ce qui signifie que la classe tumoral a également une fréquence de 402 vu qu'il n'ya pas de données manquantes ni de données dupliquées.
A priori, nous savons donc que nos classes sont équilibrées.

3.2 Distribution des features

```
# Statistique descriptive des variables
for column in X.columns:
    print(X[column].describe())

count    804.000000
mean       5.588943
std        2.889901
min        0.610504
25%        3.174250
50%        5.467024
75%        8.444756
max       10.263846
Name: ADH1C, dtype: float64
```

Figure 9: Analyse des features

La fonction describe() nous permet également d'analyser les contenu de nos features. Comme nos features sont numériques, alors la fonction nous des informations différentes de ceux de la variable "tissue_status qui est catégoricielle. En effet en plus de count(nombre d'observations), elle nous renvoie le mean(la moyenne arithmétique des valeurs de la variable), le std(l'éccart-types de la variable), le min puis le max des valeurs de la variable et enfin elle renvoie les valeurs du 1er, de 2è et du 3è quartile. On rappelle que 25% des données sont inférieures au 1er quertile, 50% au 2ème et 75% au 3ème.

3.3 Visualisation des données

Nous utilisons des histogrammes pour visualiser la distribution des gènes entre les deux classes.

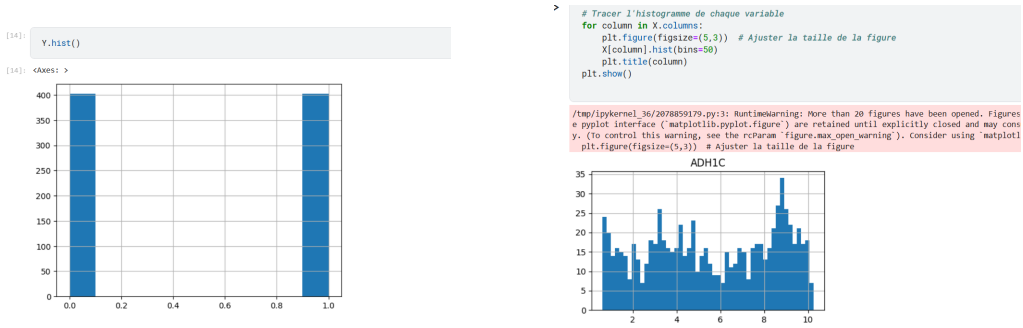

Figure 10: Visualisation

Avec les visualisations, nous pouvons confirmer les analyses statisques que nous avons eu à faire plus haut. Ici pour les classes, par exemple, on remarque bien qu'elles sont équilibrées à travers l'histogramme.

4 Application des algorithmes de classification

Nous appliquérons deux méthodes de classification pour commencer : la Régression Logistique et le SVM.

4.1 Régression Logistique

4.1.1 1er modèle:

Nous entrainons un modèle de régression logistique sur l'ensemble de nos données standardisées:

Figure 11: Prédiction avec la regression logistique

Une fois l'entrainement effectué, vient la phase de prédiction puis la phase d'évaluation du modèle avec les données de test. Ainsi on compare les prédictions faite par le modèle avec les soties de test dont nous disposons. Puis les resultats sont regroupés dans la matrice de confusion
Pour mesurer la performance de notre modèle, nous avons calculer son accuracy(qui calcule le rapport de bonne prédictions sur l'ensemble des données). Ici nous avons obtenu une accuracy de 1 on a plus besoin de calculer d'autres comme le recall ou encore le precesion et Fscore.
Toutefois une accuracy de 1 nous interpelle beaucoup. En effet soit notre modèle a fait du overfiting, soit nos données étaient très clean pour permettre une telle performance.

4.1.2 Nouveau Modèle avec Top 1,2, 3 features:

L'intérêt de trouver les top features du modèle c'est de savoir quels sont les features qui interviennent plus dans la prise de décision dans mon modèle. Ainsi je pourrais reduire le nombre de features tout en ayant de bonnes performances.

```
[24]:
#Coef de mon modèle
# Créer un DataFrame pour stocker les coefficients
coefficients = pd.DataFrame(lRegr.coef_[0], index=X.columns, columns=['beta'])

# Trier les coefficients par ordre croissant
coefficients = coefficients.sort_values(by='beta')

# Afficher les coefficients sous forme de graphique à barres
coefficients.plot.bar(figsize=(15, 3))
plt.xlabel('Caractéristique')
plt.ylabel('Coefficient')
plt.title('Coefficients du modèle de régression logistique')
plt.show()
```

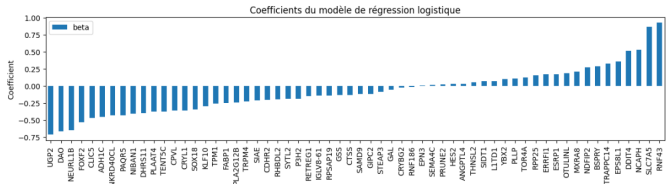

Figure 12: Recherche de top features

Nous remarquons que les tops 4 features sont SLC7A5, RNF43, UGP2 et DAO. Cependant, il est important de vérifier la correlation entre ces features avant de les selectionner. En effet si 02 features sont fortement correlés, cela signifie qu'ils apportent quasiment la même information. Donc choisir les 02 aura le même impact que si l'on en choisissait seulement 1. Alors il serait plus interessant de choisir parmi les tops feautres celle qui ne sont pas fortement corrélés.

```
# Génération de la heatmap de corrélation
var=["SLC7A5","UGP2","DAO", "RNF43"]
new=data[var]
sns.heatmap(new.corr(), cmap='coolwarm', annot=True)
plt.title('Heatmap de corrélation des variables numériques')
plt.show()
```

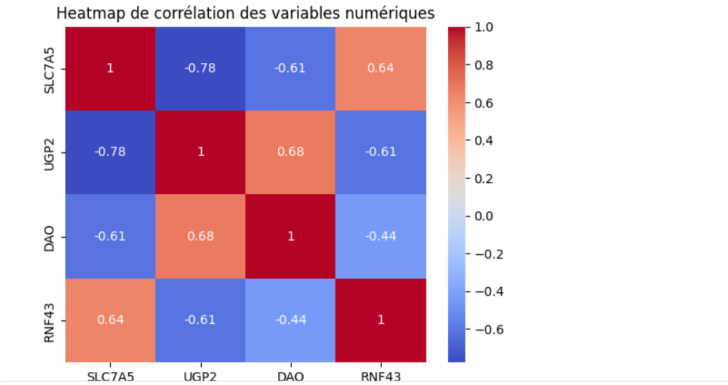

Figure 13: Corrélation entre top4 features

Nous remarquons que les features SLC7A5 et UGP2 sont fortement corrélés alors comme top3 features, nous choisirons SLC7A5, RNF43 et DAO.

```python
feature_sets = [
    ["SLC7A5"],
    ["SLC7A5", "RNF43"],
    ["SLC7A5", "RNF43", "DAO"]
]

# Itérer sur chaque ensemble de caractéristiques
for idx, features in enumerate(feature_sets, 1):
    # Sélectionner les caractéristiques actuelles
    X = data[features]

    # 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)

    # Appliquer le scaling
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Initialiser et entraîner le modèle de régression logistique
    lr = LogisticRegression()
    lr.fit(X_train_scaled, Y_train)

    # Faire des prédictions sur l'ensemble de test
    predictions = lr.predict(X_test_scaled)

    # Évaluer la performance du modèle
    accuracy = accuracy_score(Y_test, predictions)
    conf_matrix = confusion_matrix(Y_test, predictions)
```

Figure 14: Test du modèle avec top1,2,3 feautres

Nous allons entrainer notre mmodèle maintenant sur le top1, le top2 puis le top3 features donc nous avons defini les 03 en tant que dataset. Puis à travers une boucle for, on obtient les 03 resultats.

```python
    # Afficher les résultats
    print(f"\nRésultats pour Xtop{idx} ({', '.join(features)}):")
    print(f"Accuracy du modèle : {accuracy:.4f}")
    print("Matrice de confusion :")
    print(conf_matrix)
    print("-" * 40)
```

```
Résultats pour Xtop1 (SLC7A5):
Accuracy du modèle : 0.9255
Matrice de confusion :
[[74  6]
 [ 6 75]]
----------------------------------------

Résultats pour Xtop2 (SLC7A5, RNF43):
Accuracy du modèle : 0.9565
Matrice de confusion :
[[79  1]
 [ 6 75]]
----------------------------------------

Résultats pour Xtop3 (SLC7A5, RNF43, DAO):
Accuracy du modèle : 0.9938
Matrice de confusion :
[[79  1]
 [ 0 81]]
----------------------------------------
```

Figure 15: Performance du modèle avec top1,2,3 features

Nous remarquons que plus nous ajoutons de features, plus la performance du modèle augmente. Avec les 03 tops features, nous avons une acccuracy de 0.99 ce qui est assez excellent. Donc au lieu de travailler sur les 60features, avec seulement 03features nous pouvons avoir les mêmes résultats.

4.1.3 Etude de cas

Figure 16: Cas

Nous avons ensuite fait une prédiction pour un nouveau cas avec notre dernier modèle.
D'abord nous avons transformer les données du nouveau cas en dataFrame avec pandas puis nous l'avons standardiser avant de procéder à la prédiction. Finalement notre modèle a prédit le nouveau cas comme normal avec une probabilité de 67%. Ce qui n'est pas trop fameux pour une prédiction de cancer. En effet cela signifie que le patient a 37% de chance d'avoir un cancer or nous savons qu'avec les cancers, le traitement est efficace lorsque le cancer est diagnostiqué tôt. Alors pour plus de prudence, nous recommanderons plus de test pour notre patient.

4.2 SVM (Support Vector Machine)

4.2.1 1er modèle

Figure 17: Prédiction avec le modèle SVM

Notre modèle SVM contient 02 paramètres: random_state qui nous permettra d'avoir la même distribution des données à chaque fois que nous exécutons notre code puis probability=true qui nous permettra de récupérer la probabilité de prédiction des différentes classes.

4.2.2 Test du modèle avec top1,2,3 feautres

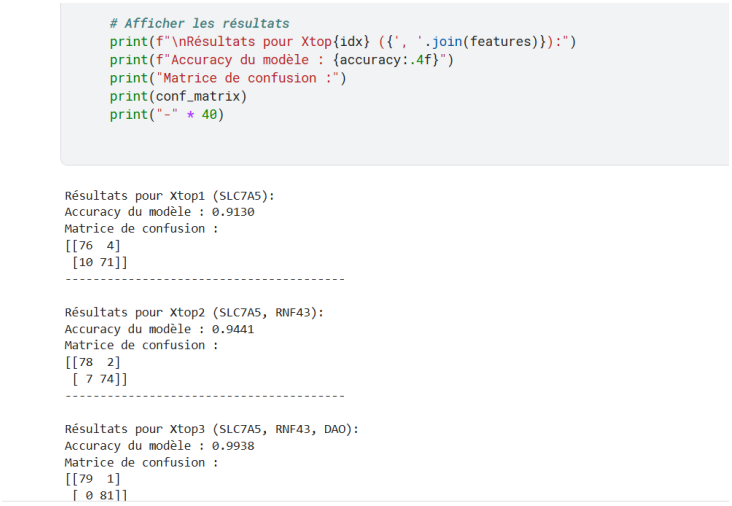

Figure 18: Prédiction avec le modèle SVM

Comparativement à la regression logistiques, nous voyons que avec seule-

ment les 03 tops features, nous avons les informations necessaires pour realiser nos prédictions.

4.2.3 Etude de cas

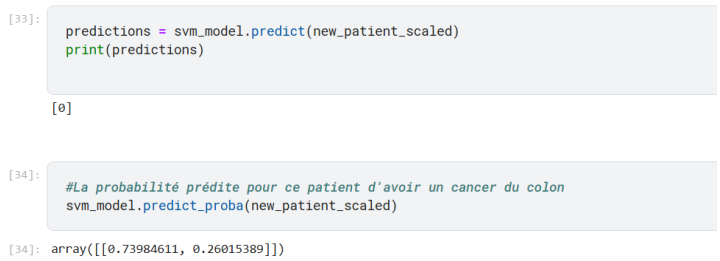

Figure 19: Prédiction d'un cas particulier

Nous remarquons que avec le modèle SVM, notre patient est normal à 73%. Ce qui est mieux par rapport aux 67% de la regression logistique.

5 Partie autonomie

5.1 K-NN

Figure 20: Implémentation du K-NN

Nous avons implémenté le K- NN avec les hyperparamètres par défauts puis on remarque la même accuracy de 1. Ce qui est bon signe. Toute fois

nous allons chercher le nombre de voisin optimal pour notre modèle car par défaut il en prend 5.

Figure 21: Nombre de voisins optimale

Ainsi nous observer grâce à une gridSearch que le nombre de voisin optimale est 3. C'est-à-dire qu'en obervant seulement 03 voisins le modèle est capable de prédire la classe d'une nouvelle observation. Ce qui plus performant.

5.2 Decision tree et Random Forrest

Figure 22: Implémentation des 02 modèles

Comme l'on pourrait s'en doutter, avec ces 02 modèles aussi nous avons une accuracy de 1. Toutfois nous pouvons également faire une GridSearch pour rechercher les hyperparamètres optimaux pour ces 02 modèles. Cela sera interresant car on pourrait notamment économiser des ressources. En attendant, nous allons chercher les top feautres qui interviennent plus da,s la prise de décision de ces modèles.

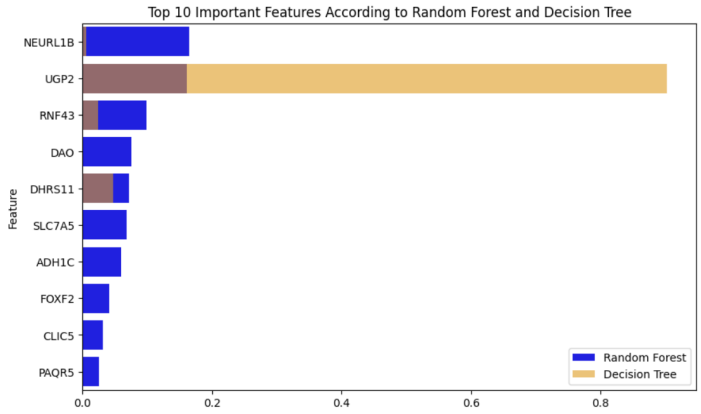

Figure 23: feature importance

Grâce à ce graphe, nous pouvons observer que pour la decision tree, il n'ya que 04 features (NEURL1B, UGP2, RNF43, DHRS11) qui interviennent dans la prise de décision. Donc on pourrait juste entrainer notre modèle sur ces 04 features puis avoir les mêmes resultats. Par contre pour le random forrest, plusieurs features interviennent dans la prise de décision cependant il ya des tops features avec lesquels on pourrait travailler puis avoir de bons résultats.

6 Conclusion

Ce rapport montre l'application des algorithmes de classification supervisée sur un dataset de cancer du côlon. Nous avons évalué la performance des modèles en utilisant plusieurs métriques et fait une prédiction pour un nouveau cas clinique. Particulièrement, notre dataset était bien clean et a surement été conçu sur mesure vu les performances de nos modèles. Toutefois nous restons conscient que dans la réalités il ne sera pas aisé d'avoir de telle performance dès les premières implémentations. En ce moment, nous verrons vraiment l'importance des gridSearch et des tops features entre autre pour améliorer et optimiser nos résultats.