

Compte rendu

Apprentissage supervisé et non supervisé

Réalisée par :
Ihssane BAMMAD

Tables de matières :

I. Introduction :	5
II. Problématisation :	5
III. Processus machine learning :	6
IV. Les étapes de classification :	6
1. Préparation du dataset :	6
2. Exploration des données :	13
3. Application des algorithmes de classification :	17
<input type="checkbox"/> Méthode de régression logistique :	17
<input type="checkbox"/> Méthode de SVM :	18
4. Évaluation des modèles	19
<input type="checkbox"/> La régression logistique :	19
<input type="checkbox"/> Méthode de SVM	21
5. Interprétation :	22
<input type="checkbox"/> La régression logistique :	23
<input type="checkbox"/> Méthode de SVM :	23
6. Etude de cas :	25
1. La régression logistique :	25
2. Méthode SVM :	26
7. Autres méthodes machine learning :	27
7.1. k-NN (k-Nearest Neighbors):	27
7.2. Arbre de décision:	29
7.3. Forêt aléatoire	32
V. Conclusion	34

Listes de figures :

Figure 1:les étapes de l'apprentissage automatique	6
Figure 2:Code de préparation de la data.....	7
Figure 3:les valeurs dupliqués dans la data	7
Figure 4:le nombre de doublons dans la data	8
Figure 5:les features X	8
Figure 6:la variable cible Y	9
Figure 7:Code de la distribution des classes	9
Figure 8:nombre pour chaque classe	10
Figure 9:Code de division de la data	10
Figure 10:Résultat d'entrainement.....	11
Figure 11:Résultat de test	12
Figure 12:code de taille de chaque data	13
Figure 13:la taille de chaque data.....	13
Figure 14:Analyse statistique de la data.....	14
Figure 15:Code de l'histogramme	15
Figure 16:Histogramme de ADH1C	15
Figure 17:Code de diagrammes en boîte des caractéristiques.....	16
Figure 18:Diagramme en boîte.....	16
Figure 19:Code de pairplot.....	16
Figure 20:graphe de pairplot	17
Figure 21:Code de régression logistique	17
Figure 22:Code de la méthode SVM.....	18
Figure 23:Code de la matrice de confusion et le rapport de classification	19
Figure 24:la matrice de confusion pour LR	20
Figure 25:le rapport de classification pour LR.....	20
Figure 26:Code de la matrice de confusion et le rapport de classification pour SVM	21
Figure 27:matrice de confusion pour SVM.....	21
Figure 28:Le rapport de classification du SVM	22
Figure 29:Code de top 1,2 et 3 des features pour LR.....	23
Figure 30: top 1,2 et 3 des features pour LR	23
Figure 31:Code de top 1,2 et 3 des features pour SVM	24
Figure 32: top 1,2 et 3 des features pour SVM	24
Figure 33:Code de LR pour la prédiction de l'état du patient	25
Figure 34:l'état de la patient obtenue.....	25

Figure 35:Figure 33:Code de SVM pour la prédiction de l'état du patient	26
Figure 36:l'état de la patient	26
Figure 37:Code de la méthode KNN	27
Figure 38:le code de la matrice de confusion et le rapport de classification pour KNN	28
Figure 39:La matrice de confusion pour KNN.....	28
Figure 40:Le rapport de classification pour KNN.....	29
Figure 41:Code de l'arbre de décision	29
Figure 42:la matrice de confusion de l'arbre de décision	30
Figure 43:Code des gènes les plus importants	30
Figure 44:les gènes les plus importants pour l'arbre de décision	31
Figure 45:le rapport de classification pour l'arbre de décision.....	31
Figure 46:code de la méthode de forêt	32
Figure 47:la matrice de confusion pour la forêt aléatoire	32
Figure 48:Code des gènes les plus importants pour forêt aléatoires	33
Figure 49:les gènes les plus importants pour la forêt aléatoire	33
Figure 50:Le rapport de classification pour forêt aléatoire	34

I. Introduction :

L'apprentissage automatique (ou machine learning) est devenu un outil incontournable dans de nombreux domaines, notamment dans le domaine médical. Grâce à la puissance des algorithmes et à la disponibilité croissante des données, il est désormais possible d'automatiser des tâches complexes comme la détection de maladies à partir de données biologiques. L'une de ces applications concerne la classification des échantillons de patients en fonction de leur statut de santé. Le cancer du côlon, un type de cancer fréquent, peut être analysé à partir des profils d'expression génique, où les variations dans l'expression des gènes permettent de différencier les tissus normaux des tissus cancéreux.

Dans ce travail, nous nous intéressons à l'application d'algorithmes de classification supervisée sur un jeu de données d'expression génique du cancer du côlon, disponible sur Kaggle. L'objectif est de développer un modèle capable de classer automatiquement les échantillons en deux catégories : "normal" et "cancer". À travers cette étude, nous explorerons les performances de plusieurs algorithmes, tout en évaluant la précision et la pertinence de ces modèles pour un usage potentiel dans le domaine médical.

II. Problématisation :

Dans le domaine de la santé, la détection précoce du cancer est un enjeu crucial pour augmenter les chances de survie des patients. Le cancer du côlon, l'un des types de cancer les plus répandus, nécessite des méthodes diagnostiques rapides et précises pour distinguer les tissus sains des tissus malins. Traditionnellement, les méthodes de détection reposent sur des techniques invasives et des analyses coûteuses. Toutefois, avec l'essor des approches basées sur l'intelligence artificielle et l'apprentissage automatique, il est désormais possible de tirer parti des données biologiques, telles que l'expression des gènes, pour classifier automatiquement les échantillons entre "normal" et "cancer". Cette démarche permettrait non seulement de réduire les coûts et les délais, mais aussi d'améliorer la précision du diagnostic. Ainsi, ce projet vise à explorer l'application de différents algorithmes de classification supervisée pour identifier, à partir des profils d'expression génique, les échantillons associés au cancer du côlon.

III. Processus machine learning :

Pour répondre à cette problématique, nous allons suivre un ensemble d'étapes méthodologiques. Tout d'abord, nous commencerons par la préparation des données en nettoyant et normalisant le jeu de données d'expression génique afin d'assurer sa qualité pour l'analyse. Ensuite, nous appliquerons plusieurs algorithmes de classification supervisée, tels que la régression logistique et les machines à vecteurs de support (SVM), pour entraîner nos modèles. Nous évaluerons les performances des modèles à l'aide de métriques comme la précision, le score F1, et la matrice de confusion. Enfin, nous interpréterons les résultats afin de comprendre les facteurs déterminants dans la classification des échantillons.

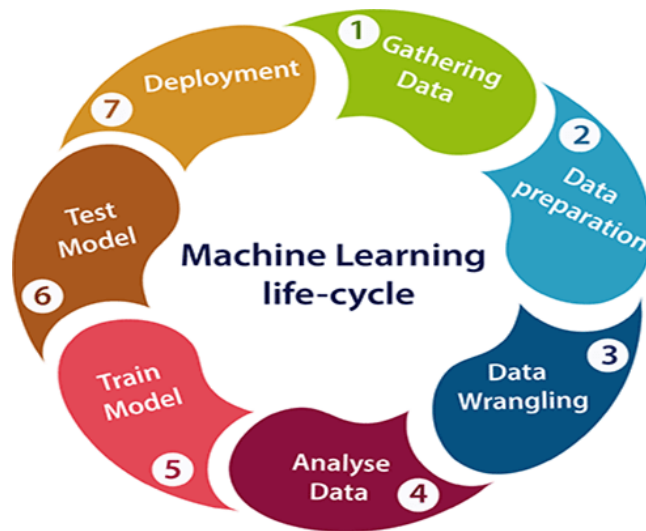


Figure 1: les étapes de l'apprentissage automatique

IV. Les étapes de classification :

Dans cette section, nous décrivons les différentes étapes suivies pour appliquer les algorithmes de classification sur le jeu de données d'expression génique du cancer du côlon, depuis la préparation des données jusqu'à l'évaluation des modèles.

1. Préparation du dataset :

- Le dataset, préalablement téléchargé depuis Google Drive, a été chargé dans un notebook Python à l'aide des bibliothèques pandas, facilitant ainsi la manipulation et l'analyse des données
- Après le chargement des données, une vérification initiale a été effectuée en utilisant la fonction `isnull().sum()` pour détecter les valeurs manquantes, et la fonction

`duplicated()` pour identifier les doublons. Les résultats ont montré qu'il n'y avait ni valeurs manquantes ni doublons dans le dataset, facilitant ainsi la poursuite du processus d'analyse sans nécessité de nettoyage supplémentaire.

```
from google.colab import drive
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from tabulate import tabulate

drive.mount('/content/drive')
fch=pd.read_csv('/content/drive/My Drive/colon_cancer.csv', delimiter=';')
print("Valeurs manquantes par colonne :")
print(fch.isnull().sum().to_frame().style.background_gradient(cmap='coolwarm'))

doublon = fch.duplicated()
print("Lignes en doublon :")
print(fch[doublon].style.background_gradient(cmap='coolwarm'))
print(f"Le nombre de doublons est : {doublon.sum()}")
# Valeurs manquantes
print("Valeurs manquantes par colonne :")
print(tabulate(fch.isnull().sum().to_frame(), headers='keys', tablefmt='pretty'))

# Doublons
print("Lignes en doublon :")
print(tabulate(fch[doublon], headers='keys', tablefmt='pretty'))

# Nombre de doublons
print(f"Le nombre de doublons est : {doublon.sum()}")
```

Figure 2:Code de préparation de la data

Cela nous donne :

⇒ Valeurs manquantes par colonne :

	0
id_sample	0
ADH1C	0
DHRS11	0
UGP2	0
SLC7A5	0
CTSS	0
DAO	0
NIBAN1	0
PRUNE2	0
FOXF2	0
TENT5C	0
KLF10	0
FABP1	0
RPSAP19	0
NCAPH	0
TPM1	0
PLA2G12B	0
PLAAT4	0
IGLV8-61	0
GSS	0

Figure 3:les valeurs dupliqués dans la data

Lignes en doublon :

```
+-----+-----+-----+-----+-----+-----+-----+-----+
| id_sample | ADH1C | DHRS11 | UGP2 | SLC7A5 | CTSS | DAO | NIBAN1 | PRUNE2 |
+-----+-----+-----+-----+-----+-----+-----+-----+
Le nombre de doublons est : 0
```

Figure 4: le nombre de doublons dans la data

Ces résultats indiquent que les données sont bien préparées, ce qui permet de passer sereinement aux étapes suivantes de l'analyse.

- Les données ont ensuite été divisées en deux ensembles distincts : X pour les features (caractéristiques), contenant les variables explicatives, et Y pour la variable cible, représentant la classe à prédire. Cette séparation est une étape clé pour entraîner les modèles de machine learning, où X servira à fournir les informations nécessaires à la prédiction, et Y sera utilisée pour évaluer la performance du modèle

```
x = fch.drop(columns=['id_sample', 'tissue_status'])
y = fch['tissue_status']

# Afficher les premières lignes des features (X) avec un style
print('Pour X (features) :')
display(x.head().style.background_gradient(cmap='coolwarm'))

# Afficher les premières lignes de la variable cible (Y) avec un style
print('Pour la variable cible (Y) :')
display(y.head().to_frame().style.background_gradient(cmap='coolwarm'))
```

On a affiché les 5 premières lignes de chaque variable X et Y

Pour X (features) :

	ADH1C	DHRS11	UGP2	SLC7A5	CTSS	DAO	NIBAN1
0	9.199944	6.090054	7.062512	3.864253	7.869368	8.465133	1.174665
1	7.767618	6.027985	6.318818	3.069581	6.410334	8.159814	5.959414
2	7.918904	5.885948	6.917742	3.188257	7.915549	8.004194	1.560386
3	9.053553	6.027985	7.081085	2.357523	5.657726	8.004194	3.548988
4	6.027822	5.791257	5.937685	4.137667	5.818999	8.056067	3.307945

Figure 5: les features X

Pour la variable cible (Y) :

tissue_status	
0	normal
1	normal
2	normal
3	normal
4	normal

Figure 6: la variable cible Y

- Une analyse de la distribution des classes a été effectuée à l'aide de la fonction `value_counts()` sur la variable cible. Cette étape permet de vérifier si les classes sont équilibrées. Assurer un équilibre des classes est crucial pour éviter que le modèle ne soit biaisé vers la classe majoritaire, ce qui pourrait non seulement fausser les prédictions, mais aussi conduire à un surapprentissage (overfitting), où le modèle s'adapte trop aux données d'entraînement sans bien généraliser aux nouvelles données.

```
# Analyser la data
distribution_classe=y.value_counts()
print("Distribution des classes (normal vs cancer) :")
print(distribution_classe)
import seaborn as sns
import matplotlib.pyplot as plt
sns.countplot(x=y)
plt.title("Distribution des classes (normal vs cancer)")
plt.show()
```

Figure 7: Code de la distribution des classes

Ce code révèle que les deux catégories, 'normal' et 'tumoral', sont parfaitement équilibrées, avec 402 échantillons dans chaque classe

```
Distribution des classes (normal vs cancer) :
tissue_status
normal      402
tumoral     402
Name: count, dtype: int64
```

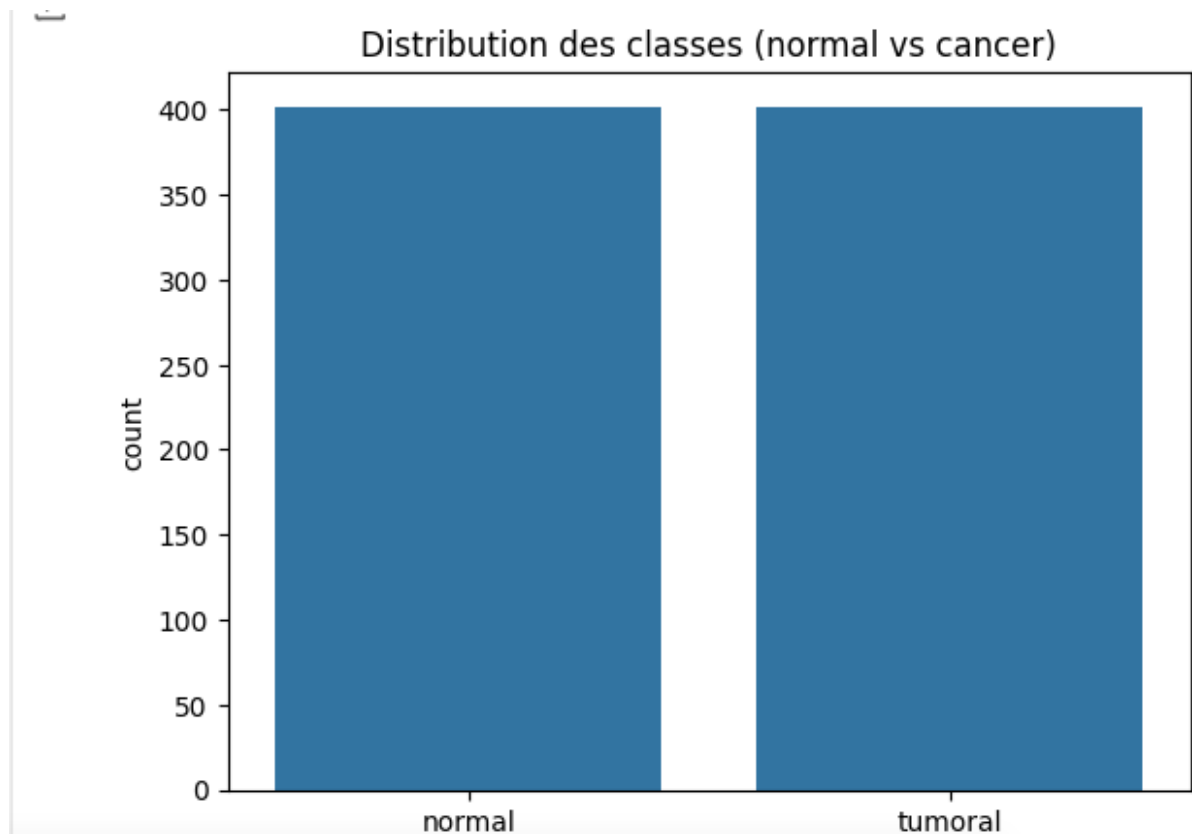


Figure 8: nombre pour chaque classe

- Nous passons maintenant à la division des données en deux ensembles : une partie pour l'entraînement, utilisée pour former notre modèle, et une partie pour le test, destinée à évaluer l'efficacité du modèle. Nous avons réparti les données en utilisant `train_test_split`, avec 80 % des données pour l'entraînement et 20 % pour le test

```
# Split des données en ensemble d'entraînement et de test  
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, train_size=0.8)
```

Figure 9: Code de division de la data

- Après avoir divisé les données, une standardisation a été appliquée pour mettre les features sur une échelle uniforme entre 0 et 1. La méthode `fit_transform()` a été utilisée sur l'ensemble d'entraînement pour ajuster les paramètres, et la méthode `transform()` a été appliquée à l'ensemble de test en utilisant les mêmes paramètres. Cette étape est cruciale pour garantir que les features sont comparables et contribuent de manière équitable à l'apprentissage du modèle, ce qui peut améliorer la performance et la stabilité du modèle.

```

from sklearn.preprocessing import MinMaxScaler
import pandas as pd

# Normalisation pour la partie d'entraînement
scaler = MinMaxScaler()
# Ajuster le scaler aux données d'entraînement (apprend min et max)
scaler.fit(x_train)
# Appliquer la transformation sur les données d'entraînement
X_normalized_train = scaler.transform(x_train)
# Conversion en DataFrame pour un affichage plus clair avec arrondi
X_normalized_df_train = pd.DataFrame(X_normalized_train, columns=x_train.columns).round(3)
# Afficher les 5 premières lignes des données normalisées
print("\n==== Aperçu des données normalisées pour l'entraînement ==== \n")
pd.set_option('display.max_columns', None)
print(X_normalized_df_train.head(5))
# Appliquer la transformation sur les données de test
X_normalized_test = scaler.transform(x_test)
# Conversion en DataFrame pour un affichage plus clair avec arrondi
X_normalized_df_test = pd.DataFrame(X_normalized_test, columns=x_train.columns).round(3)
# Afficher les 5 premières lignes des données normalisées
print("\n==== Aperçu des données normalisées pour les données de test ==== \n")
pd.set_option('display.max_columns', None)
print(X_normalized_df_test.head(5))

```

Cela nous donne comme résultat :

	RETREG1	OTULINL	CPVL	SAMD9	ANKRD40CL	EPN3	CRYBG2	GIPC2	P3H2	\	
0	0.579	0.648	0.711	0.861	0.961	0.425	0.432	0.893	0.666		
1	0.592	0.458	0.534	0.612	0.444	0.208	0.423	0.770	0.271		
2	0.482	0.729	0.400	0.301	0.541	0.858	0.271	0.648	0.115		
3	0.756	0.692	0.727	0.678	0.720	0.779	0.561	0.968	0.193		
4	0.477	0.419	0.304	0.324	0.173	0.920	0.436	0.675	0.260		
	STEAP3	THNSL2	TRAPPC14	RHBDL2	RPP25	SEMA4C	RNF43	EPS8L1	TOR4A	\	
0	0.345	0.509	0.740	0.424	0.643	0.407	0.295	0.570	0.941		
1	0.678	0.466	0.676	0.844	0.643	0.572	0.429	0.562	0.775		
2	0.702	0.370	0.781	0.044	0.524	0.608	0.756	0.341	0.444		
3	0.768	0.514	0.600	0.843	0.771	0.205	0.456	0.492	0.713		
4	0.832	0.145	0.564	0.350	0.737	0.780	0.694	0.930	0.789		
	PAQR5	SIDT1	ESRP1	SYTL2	BSPRY	CDHR2	ERRFI1	CLIC5	PLLP	GAL	\
0	1.000	0.495	0.909	0.763	0.067	0.520	0.473	0.405	0.121	0.187	
1	0.699	0.805	0.939	0.723	0.555	0.630	0.127	0.669	0.321	0.519	
2	0.449	0.229	0.762	0.357	0.591	0.430	0.331	0.519	0.192	0.464	
3	0.977	0.754	0.919	0.688	0.588	0.416	0.110	0.628	0.329	0.197	
4	0.445	0.250	0.744	0.344	0.626	0.333	0.315	0.335	0.257	0.417	
	CRYL1	YBX2	ANGPTL4								
0	0.453	0.830	0.221								
1	0.309	0.696	0.249								
2	0.343	0.622	0.238								
3	0.572	0.746	0.312								
4	0.437	0.765	0.142								

Figure 10: Résultat d'entrainement

==== Aperçu des données normalisées pour les données de test ====

	ADH1C	DHRS11	UGP2	SLC7A5	CTSS	DAO	NIBAN1	PRUNE2	FOXF2	TENT5C	\
0	0.298	0.303	0.757	0.141	0.079	0.754	0.823	0.723	0.846	0.282	
1	0.153	0.166	0.377	0.802	0.505	0.696	0.204	0.573	0.066	0.541	
2	0.020	0.072	0.454	0.806	0.240	0.472	0.141	0.545	0.195	0.623	
3	0.774	0.773	0.805	0.445	0.740	0.872	0.481	0.441	0.638	0.733	
4	0.961	0.872	0.877	0.449	0.357	0.919	0.263	0.646	0.477	0.451	

	KLF10	FABP1	RPSAP19	NCAPH	TPM1	PLA2G12B	PLAAT4	IGLV8-61	GSS	\
0	0.702	0.141	0.285	0.150	0.742	0.274	0.460	0.212	0.334	
1	0.248	0.390	0.241	0.526	0.480	0.005	0.091	0.020	0.389	
2	0.472	0.194	0.121	0.875	0.487	0.004	0.370	0.281	0.245	
3	0.479	0.831	0.289	0.277	0.549	0.517	0.545	0.687	0.586	
4	0.385	0.150	0.301	0.525	0.703	0.399	0.447	0.541	0.573	

	L1TD1	RNF186	HES2	MXRA8	SOX18	NDFIP2	SIAE	NEURL1B	DDIT4	TRPM4	\
0	0.171	0.249	0.120	0.559	0.502	0.140	0.139	0.900	0.336	0.175	
1	0.561	0.212	0.634	0.497	0.562	0.367	0.136	0.407	0.700	0.386	
2	0.733	0.330	0.677	0.247	0.273	0.477	0.393	0.112	0.604	0.505	
3	0.274	0.711	0.690	0.438	0.721	0.506	0.823	0.820	0.326	0.845	
4	0.404	0.610	0.509	0.527	0.430	0.457	0.400	0.749	0.503	0.964	

	RETREG1	OTULINL	CPVL	SAMD9	ANKRD40CL	EPN3	CRYBG2	GIPC2	P3H2	\
0	0.508	0.114	0.499	0.446	0.288	0.027	0.432	0.370	0.657	
1	0.390	0.490	0.469	0.367	0.274	0.351	0.215	0.343	0.254	
2	0.311	0.619	0.255	0.225	0.489	0.734	0.242	0.539	0.132	
3	0.757	0.483	0.590	0.629	0.704	0.653	0.720	0.765	0.432	
4	0.665	0.723	0.327	0.649	0.538	0.723	0.811	0.756	0.561	

	STEAP3	THNSL2	TRAPPC14	RHBDL2	RPP25	SEMA4C	RNF43	EPS8L1	TOR4A	\
0	0.206	0.639	0.243	0.287	0.356	0.746	0.020	0.010	0.259	
1	0.633	0.528	0.496	0.653	0.577	0.637	0.600	0.535	0.604	
2	0.301	0.076	0.362	0.227	0.742	0.388	0.415	0.486	0.090	
3	0.578	0.343	0.614	0.745	0.599	0.328	0.337	0.446	0.628	
4	0.535	0.484	0.579	0.659	0.568	0.525	0.268	0.344	0.610	

	PAQR5	SIDT1	ESRP1	SYTL2	BSPRY	CDHR2	ERRFI1	CLIC5	PLLP	GAL	\
0	0.268	0.069	0.013	0.022	-0.007	0.195	0.569	0.415	0.167	0.364	
1	0.141	0.415	0.656	0.065	0.594	0.287	0.284	0.270	0.473	0.071	
2	0.162	0.786	0.867	0.617	0.580	0.393	0.407	-0.007	0.369	0.198	
3	0.775	0.620	0.771	0.665	0.515	0.597	0.465	0.658	0.392	0.280	
4	0.961	0.461	0.081	0.387	0.097	0.630	0.712	0.526	0.277	0.501	

	CRYL1	YBX2	ANGPTL4
0	0.376	0.140	0.433
1	0.459	0.522	0.689
2	0.329	0.677	0.311
3	0.635	0.774	0.149
4	0.549	0.125	0.175

Figure 11: Résultat de test

2. Exploration des données :

- Dans l'exploration des données, nous commençons par analyser la distribution des classes, en examinant les proportions de 'normal' et 'tumoral' dans les ensembles d'entraînement et de test. Cette étape est essentielle pour s'assurer que les données sont équilibrées et que le modèle sera capable de généraliser correctement entre les deux catégories

```
[11] # Affichage clair des tailles
print("==== Résumé des tailles des ensembles ==== \n")
print(f"Ensemble d'entraînement : \n - Taille Features: {x_train.shape[0]} \n - Taille Target: {y_train.shape[0]}")
print(f"Ensemble de test : \n - Taille Features: {x_test.shape[0]} \n - Taille Target: {y_test.shape[0]}")
print(y_train.value_counts())
print(y_test.value_counts())
```

Figure 12: code de taille de chaque data

Cela nous donne :

```
==== Résumé des tailles des ensembles ====

Ensemble d'entraînement :
- Taille Features: 643
- Taille Target: 643

Ensemble de test :
- Taille Features: 161
- Taille Target: 161
tissue_status
tumoral    327
normal     316
Name: count, dtype: int64
tissue_status
normal      86
tumoral     75
Name: count, dtype: int64
```

Figure 13: la taille de chaque data

Ces résultats indiquent une distribution équilibrée dans les ensembles de données, ce qui est favorable pour une évaluation précise du modèle.

- Maintenant Nous avons réalisé une analyse statistique descriptive des gènes, qui fournit des mesures clés telles que la moyenne, l'écart-type, les valeurs minimales et maximales, ainsi que les percentiles (25%, 50%, 75%). Cette analyse est cruciale pour comprendre la distribution des niveaux d'expression des gènes dans le dataset. Elle aide à identifier les gènes avec des variations élevées ou faibles dans leur expression,

ce qui peut influencer la performance des modèles de classification. La méthode `describe()` de pandas a été utilisée pour obtenir ces statistiques, fournissant une vue d'ensemble complète des caractéristiques des données et permettant une préparation plus informée pour les étapes d'analyse et de modélisation ultérieures.

		GIPC2	P3H2	STEAP3	THNSL2	TRAPPC14	RHBDL2	\
count	643.000000	643.000000	643.000000	643.000000	643.000000	643.000000	643.000000	
mean	0.646429	0.407193	0.591557	0.501577	0.539285	0.541064		
std	0.171475	0.218442	0.158805	0.213805	0.168791	0.253263		
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000		
25%	0.551000	0.226000	0.511000	0.384500	0.465000	0.336500		
50%	0.667000	0.412000	0.614000	0.527000	0.561000	0.561000		
75%	0.766000	0.558000	0.706500	0.644000	0.654500	0.760000		
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000		

		RPP25	SEMA4C	RNF43	EPS8L1	TOR4A	PAQR5	\
count	643.000000	643.000000	643.000000	643.000000	643.000000	643.000000	643.000000	
mean	0.609243	0.463107	0.447984	0.522582	0.643215	0.537910		
std	0.161436	0.188712	0.193826	0.194017	0.184097	0.250003		
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000		
25%	0.523000	0.329500	0.328000	0.427000	0.580500	0.329000		
50%	0.622000	0.466000	0.411000	0.556000	0.694000	0.506000		
75%	0.727000	0.604000	0.615500	0.648500	0.763000	0.768000		
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000		

		SIDT1	ESRP1	SYTL2	BSPRY	CDHR2	ERRFI1	\
count	643.000000	643.000000	643.000000	643.000000	643.000000	643.000000	643.000000	
mean	0.439138	0.697589	0.517126	0.583613	0.453170	0.462305		
std	0.193808	0.211241	0.237444	0.185089	0.212038	0.165992		
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000		
25%	0.299500	0.657500	0.356500	0.524000	0.287000	0.339500		
50%	0.448000	0.744000	0.516000	0.620000	0.443000	0.457000		
75%	0.587000	0.818500	0.701000	0.700500	0.621000	0.559000		
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000		

		CLIC5	PLLP	GAL	CRYL1	YBX2	ANGPTL4	
count	643.000000	643.000000	643.000000	643.000000	643.000000	643.000000	643.000000	
mean	0.503406	0.414114	0.324482	0.538044	0.545642	0.321053		
std	0.209948	0.174396	0.189972	0.164303	0.193641	0.173094		
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000		
25%	0.343500	0.296000	0.186000	0.417500	0.442000	0.189000		
50%	0.477000	0.383000	0.297000	0.548000	0.567000	0.292000		
75%	0.671000	0.511000	0.451500	0.660500	0.683500	0.413000		
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000		

Figure 14:Analyse statistique de la data

Les moyennes des valeurs des gènes vont de 0.444 à 0.677, suggérant une expression généralement élevée. Les écarts-types, allant de 0.160 à 0.296, indiquent une variabilité modérée. Les valeurs vont de 0 à 1, montrant une distribution étendue des niveaux d'expression parmi les échantillons.

- Nous allons maintenant utiliser des outils de visualisation, tels que les histogrammes et les diagrammes en boîte, pour mieux comprendre la distribution et la dispersion des données. Les histogrammes nous aideront à visualiser la répartition des valeurs des gènes, tandis que les diagrammes en boîte nous permettront d'identifier les tendances centrales, les variations et les valeurs aberrantes. Cette analyse visuelle est essentielle pour obtenir des insights plus profonds sur les données, ce qui facilitera les étapes suivantes de l'analyse et de la modélisation.

Nous allons utiliser l'histogramme du gène `ADH1C` comme exemple.

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

# Exemple pour une colonne spécifique
plt.figure(figsize=(10, 6))
sns.histplot(X_normalized_df_train['ADH1C'], bins=30, kde=True)
plt.title('Histogramme de ADH1C')
plt.xlabel('Valeurs')
plt.ylabel('Fréquence')
plt.show()
```

Figure 15: Code de l'histogramme

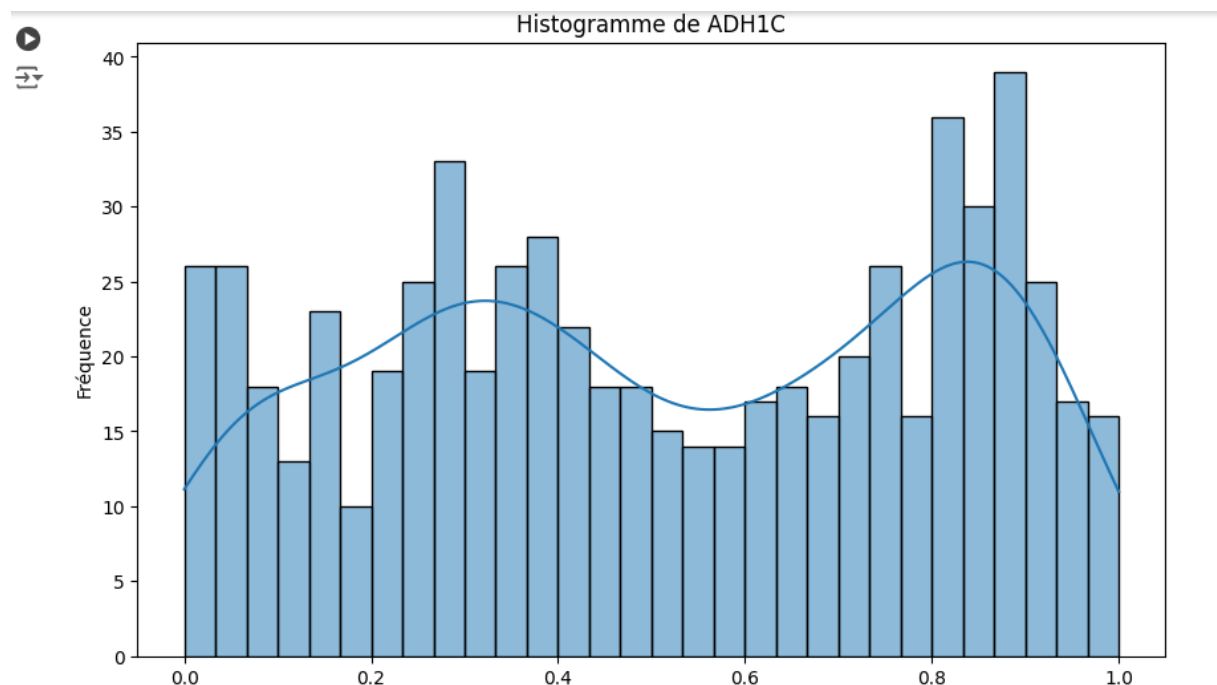


Figure 16: Histogramme de ADH1C

Après avoir examiné l'histogramme du gène `ADH1C`, nous avons créé un diagramme en boîte pour tous les gènes

```
[ ] plt.figure(figsize=(12, 6))
sns.boxplot(data=X_normalized_df_train)
plt.title('Diagramme en boîte des caractéristiques')
plt.xlabel('Caractéristiques')
plt.ylabel('Valeurs')
plt.xticks(rotation=90)
plt.show()
```

Figure 17: Code de diagrammes en boîte des caractéristiques

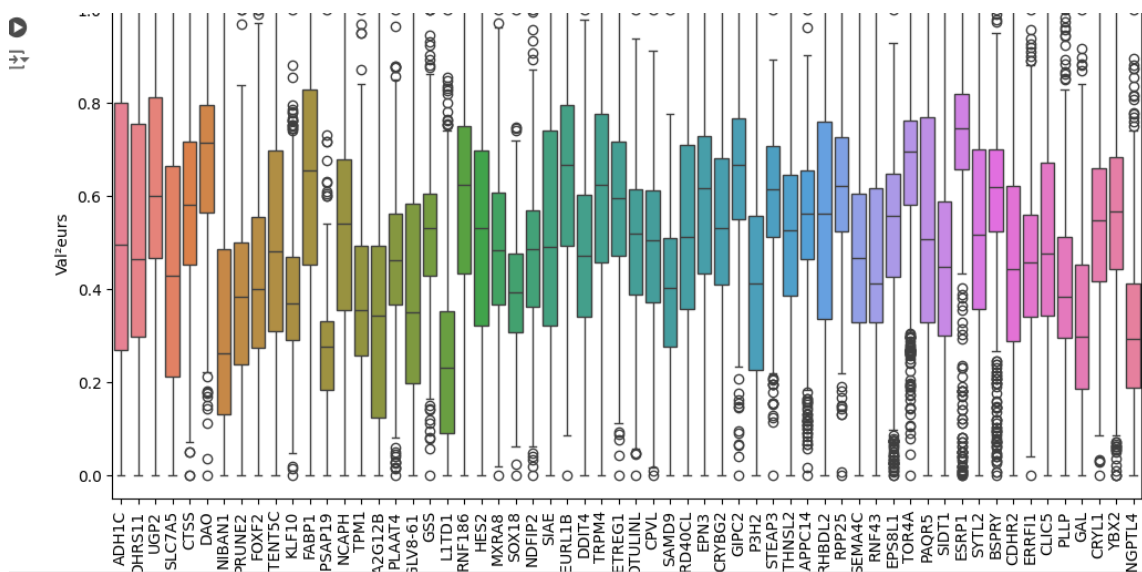


Figure 18: Diagramme en boîte

Et aussi on a utilisé un pairplot pour explorer les relations entre les gènes `ADH1C` et `DHRS11`.

```
# Sélectionner quelques gènes pour la visualisation (par exemple, les trois premiers)
gene_subset = X_normalized_df_train[['ADH1C', 'DHRS11']]

# Créer un pairplot pour ces gènes
sns.pairplot(pd.concat([gene_subset, y], axis=1), hue='tissue_status', palette='coolwarm')
plt.show()
```

Figure 19: Code de pairplot

La graphique ci-dessous aide à visualiser la distribution des données et les relations entre ces gènes, tout en distinguant les classes de tissu (normal et tumoral).

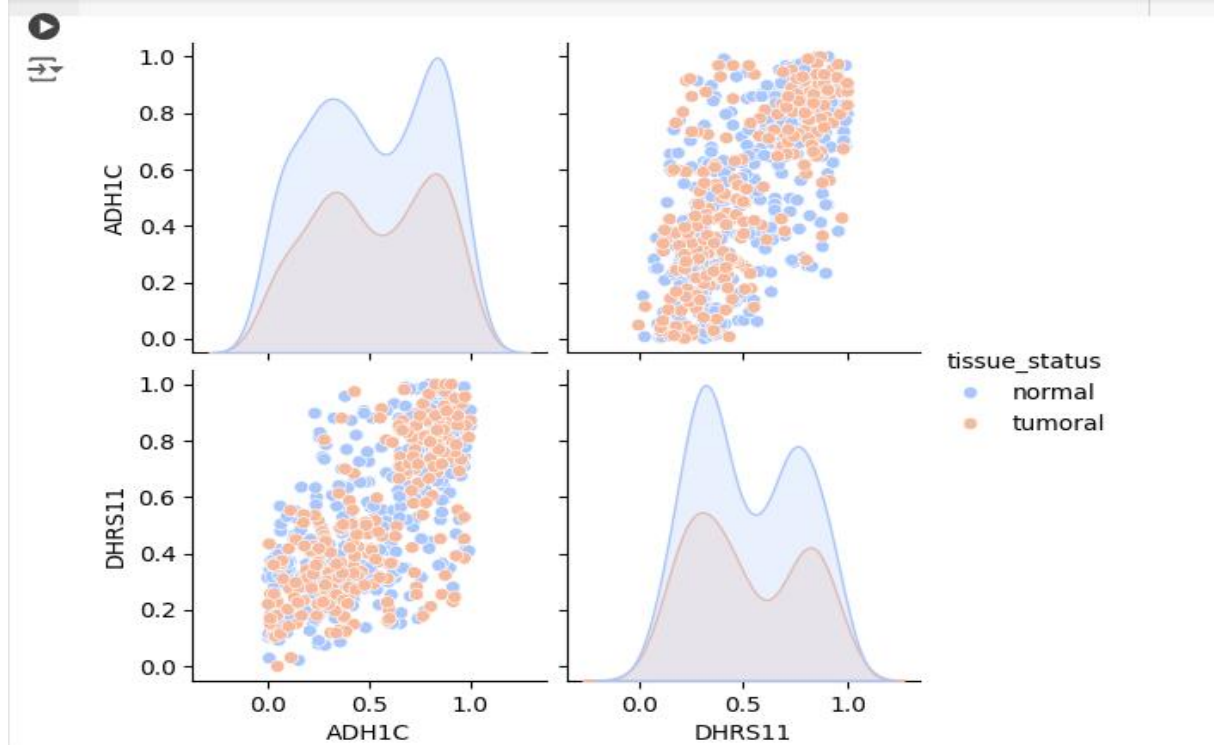


Figure 20: graphe de pairplot

3. Application des algorithmes de classification :

➤ Méthode de régression logistique :

- Nous avons maintenant appliqué un algorithme de régression logistique pour classer les données. Le modèle a été entraîné en utilisant la classe `LogisticRegression` et évalué sur le jeu de test. Les résultats montrent un taux de précision de 100%, indiquant que le modèle a parfaitement classé tous les échantillons du jeu de test.

```
# Importer les bibliothèques
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Créer un modèle de régression logistique
log_reg = LogisticRegression()
# Entraîner le modèle
log_reg = LogisticRegression()
log_reg.fit(X_normalized_train, y_train)
# Faire des prédictions
y_pred = log_reg.predict(X_normalized_test)
# Évaluer les performances
accuracy = accuracy_score(y_test, y_pred)
print(f"Taux de précision: {accuracy * 100:.2f}%")
```

Figure 21: Code de régression logistique

On a obtenu :

➡ Taux de précision: 100.00%

➤ Méthode de SVM :

- Nous avons ensuite appliqué un algorithme de Support Vector Machine (SVM) pour classer les données, en utilisant une recherche de grille pour optimiser les hyperparamètres. Le modèle SVM a été entraîné avec différentes valeurs pour les paramètres C et γ afin de déterminer les meilleures configurations. Les résultats indiquent que le modèle, une fois optimisé, a obtenu un taux de précision de 100%, sur le jeu de test, démontrant ainsi son efficacité pour cette tâche de classification

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

# Définir les paramètres à tester
param_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': [0.001, 0.01, 0.1, 1, 10]
}

# Créer le modèle SVM
svm_model = SVC(kernel='rbf')
# Configurer la recherche de grille
grid_search = GridSearchCV(svm_model, param_grid, cv=5, scoring='accuracy')
# Entraîner le modèle avec recherche de grille
grid_search.fit(X_normalized_train, y_train)
# Meilleurs paramètres trouvés
print("Meilleurs paramètres :", grid_search.best_params_)
# Évaluer le modèle avec les meilleurs paramètres
y_pred_svm_best = grid_search.predict(X_normalized_test)
# Évaluer les performances
accuracy_svm_best = accuracy_score(y_test, y_pred_svm_best)
print(f"Taux de précision pour SVM avec meilleurs paramètres : {accuracy_svm_best * 100:.2f}%")
```

Figure 22: Code de la méthode SVM

Après avoir appliqué le modèle SVM avec une recherche de grille, nous avons trouvé que les meilleurs paramètres étaient $C = 10$ et $\gamma = 0.1$. Avec ces paramètres optimisés, le modèle SVM a atteint un taux de précision de 100% sur le jeu de test, tout comme le modèle de régression logistique.

➡ Meilleurs paramètres : {'C': 10, 'gamma': 0.1}
Taux de précision pour SVM avec meilleurs paramètres : 100.00%

Comparativement, les deux modèles montrent une performance parfaite sur les données de test avec une précision de 100%. Cela indique que tant le SVM optimisé que la régression logistique sont capables de classer correctement tous les échantillons dans ce

cas spécifique. Cependant, la régression logistique est généralement plus simple et plus rapide à entraîner que le SVM, tandis que le SVM avec un noyau RBF peut parfois capturer des relations plus complexes dans les données.

4. Évaluation des modèles

Au début, nous avons évalué chaque modèle en utilisant des métriques adéquates pour mesurer leur performance. Nous avons employé la matrice de confusion ainsi que le rapport de classification pour obtenir des informations détaillées sur les erreurs de classification et la qualité des prédictions.

➤ La régression logistique :

```
# Afficher la matrice de confusion
conf_matrix = confusion_matrix(y_test, y_pred)
print("Matrice de confusion :\n", conf_matrix)

# Visualiser la matrice de confusion avec un heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Classe 0', 'Classe 1'],
            yticklabels=['Classe 0', 'Classe 1'])
plt.title('Matrice de confusion')
plt.xlabel('Prédictions')
plt.ylabel('Vérités')
plt.show()

# Afficher un rapport de classification détaillé
print("Rapport de classification :\n", classification_report(y_test, y_pred))
```

Figure 23: Code de la matrice de confusion et le rapport de classification

La maice de confusion associée est la suivante :

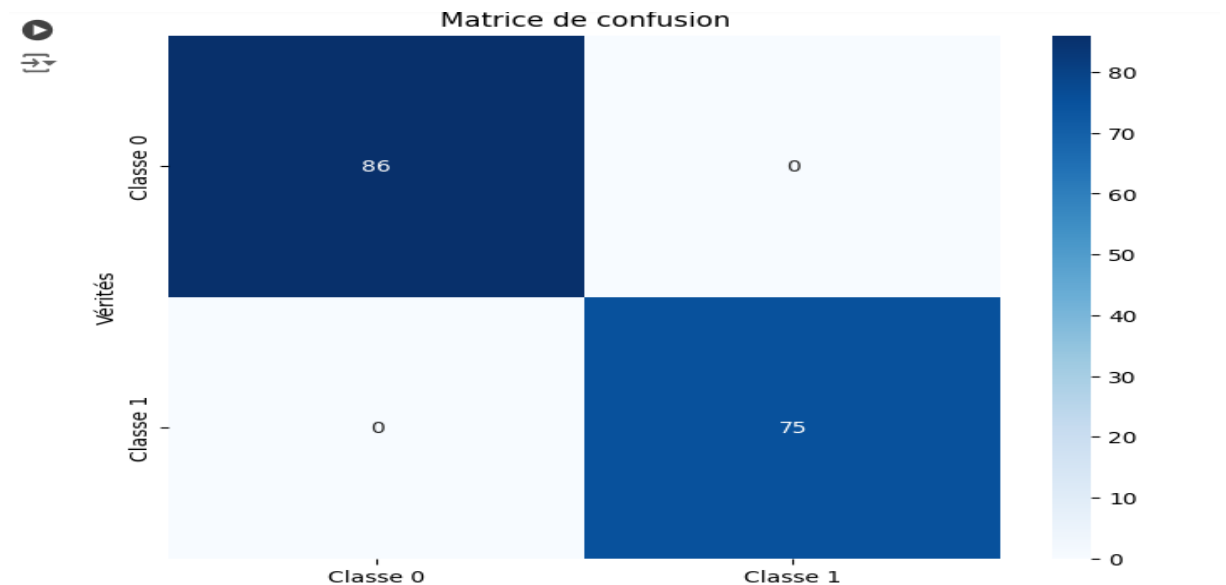


Figure 24:la matrice de confusion pour LR

Cette matrice indique que le modèle de régression logistique a correctement classé tous les échantillons du jeu de test, sans aucune confusion entre les classes 'normal' et 'tumoral'.

Pour le rapport de classification on a obtenu :

Prédictions				
Rapport de classification :				
	precision	recall	f1-score	support
normal	1.00	1.00	1.00	86
tumoral	1.00	1.00	1.00	75
accuracy			1.00	161
macro avg	1.00	1.00	1.00	161
weighted avg	1.00	1.00	1.00	161

Figure 25:le rapport de classification pour LR

Le rapport montre des scores parfaits pour toutes les métriques (précision, rappel, f1-score) pour les deux classes. Cela signifie que la régression logistique a obtenu une précision parfaite, avec une excellente capacité à détecter à la fois les classes 'normal' et 'tumoral', sans aucune erreur.

➤ Méthode de SVM

```
# Afficher la matrice de confusion
conf_matrix_svm_best = confusion_matrix(y_test, y_pred_svm_best)
print("Matrice de confusion SVM avec meilleurs paramètres :\n", conf_matrix_svm_best)

# Visualiser la matrice de confusion avec un heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_svm_best, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Classe 0', 'Classe 1'],
            yticklabels=['Classe 0', 'Classe 1'])
plt.title('Matrice de confusion - SVM avec meilleurs paramètres')
plt.xlabel('Prédictions')
plt.ylabel('Vérités')
plt.show()

# Afficher un rapport de classification détaillé
print("Rapport de classification SVM avec meilleurs paramètres :\n", classification_report(y_test, y_pred_svm_best))
```

Figure 26: Code de la matrice de confusion et le rapport de classification pour SVM

La matrice de confusion associée est la suivante :

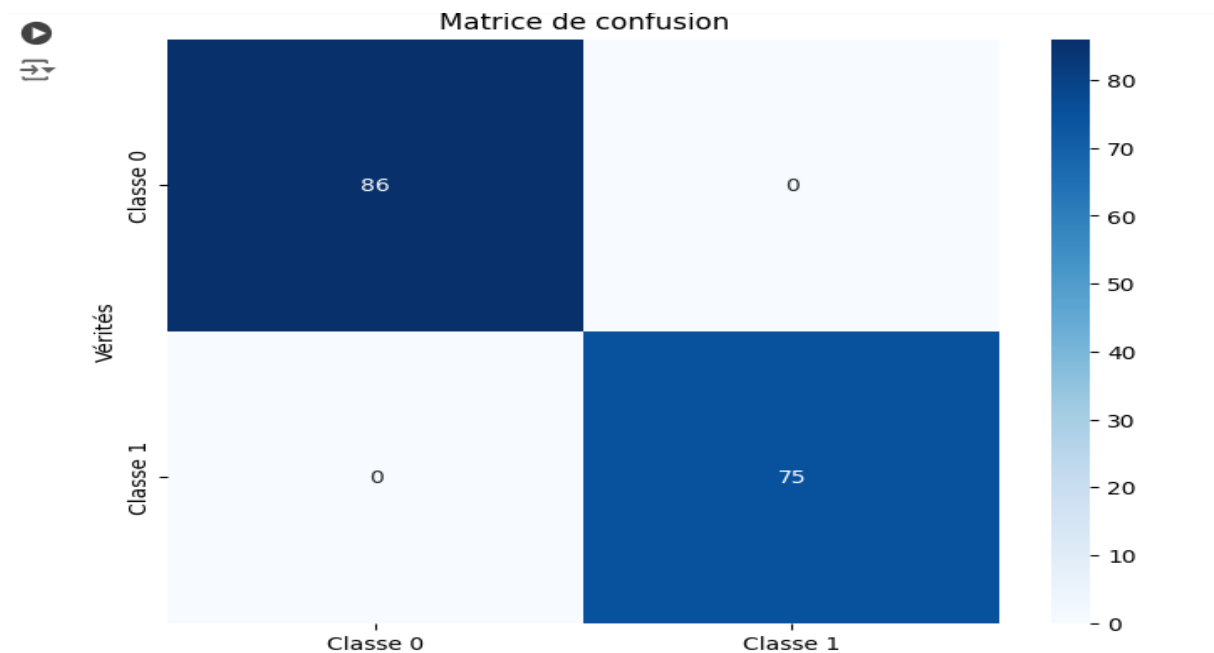


Figure 27: matrice de confusion pour SVM

Comme pour la régression logistique, la matrice de confusion du modèle SVM avec les meilleurs paramètres révèle une classification parfaite des échantillons, sans confusion entre les classes.

Pour le rapport de classification on a obtenu :

Prédictions				
Rapport de classification SVM avec meilleurs paramètres :				
	precision	recall	f1-score	support
normal	1.00	1.00	1.00	86
tumoral	1.00	1.00	1.00	75
accuracy			1.00	161
macro avg	1.00	1.00	1.00	161
weighted avg	1.00	1.00	1.00	161

Figure 28:Le rapport de classification du SVM

Le rapport de classification du SVM montre également des scores parfaits dans toutes les métriques. Le modèle a atteint une précision parfaite et a bien discriminé entre les classes 'normal' et 'tumoral', ce qui indique une performance optimale.

- **Les deux modèles, Régression Logistique et SVM, ont montré des performances idéales avec une précision de 100% et des rapports de classification parfaits. Cela suggère que les deux algorithmes sont très efficaces pour ce jeu de données spécifique, et qu'aucune erreur de classification n'a été observée dans les ensembles de test.**

5. Interprétation :

Avant de présenter les interprétations des résultats, nous allons examiner l'impact de chaque gène sur les modèles de régression logistique et SVM, en mettant en évidence les gènes les plus influents en termes de coefficients et leur rôle dans la prédiction des classes.

➤ La régression logistique :

```
import pandas as pd

# Extraire les coefficients du modèle de régression logistique
coef_lr = log_reg.coef_[0]

# Assumer que X_normalized_df_train est un DataFrame avec des noms de colonnes
feature_names = X_normalized_df_train.columns

# Créer un DataFrame pour les caractéristiques et leurs coefficients
feature_importance_lr = pd.DataFrame({
    'Feature': feature_names,
    'Coefficient': coef_lr
})

# Trier par importance (absolue des coefficients)
feature_importance_lr['AbsCoefficient'] = abs(feature_importance_lr['Coefficient'])
feature_importance_lr = feature_importance_lr.sort_values(by='AbsCoefficient', ascending=False)

# Afficher les top 1, 2 et 3 caractéristiques
print("Top caractéristiques pour la Régression Logistique avec données normalisées :")
print(feature_importance_lr.head(3))
```

Figure 29: Code de top 1,2 et 3 des features pour LR

```
➡ Top caractéristiques pour la Régression Logistique avec données normalisées :
   Feature  Coefficient  AbsCoefficient
3   SLC7A5     2.192431         2.192431
44  RNF43     1.882977         1.882977
2    UGP2    -1.773761         1.773761
```

Figure 30: top 1,2 et 3 des features pour LR

Le gène **SLC7A5** a le coefficient le plus élevé, ce qui indique qu'il est fortement associé à la classe prédite par le modèle. Une valeur élevée de **SLC7A5** augmente la probabilité que l'échantillon soit classé dans la classe positive. **RNF43** est également un gène important avec un coefficient positif élevé, ce qui souligne sa pertinence dans la prédiction. En revanche, **UGP2** présente un coefficient négatif, suggérant qu'une augmentation de ce gène est associée à une probabilité plus faible d'appartenir à la classe positive.

➤ Méthode de SVM :

```

import pandas as pd
from sklearn.svm import SVC

# Assurer que vous avez les noms de caractéristiques
feature_names = X_normalized_df_train.columns
# Créer et entraîner le modèle SVM linéaire
svm_model = SVC(kernel='linear')
svm_model.fit(X_normalized_df_train, y_train)
# Vérifier si le modèle SVM est linéaire et possède des coefficients
if hasattr(svm_model, 'coef_'):
    coef_svm = svm_model.coef_[0]

    # Créer un DataFrame pour les caractéristiques et leurs coefficients
    feature_importance_svm = pd.DataFrame({
        'Feature': feature_names,
        'Coefficient': coef_svm
    })
    # Trier par importance (absolue des coefficients)
    feature_importance_svm['AbsCoefficient'] = abs(feature_importance_svm['Coefficient'])
    feature_importance_svm = feature_importance_svm.sort_values(by='AbsCoefficient', ascending=False)

    # Afficher les top 1, 2 et 3 caractéristiques
    print("Top caractéristiques pour le SVM avec données normalisées :")
    print(feature_importance_svm.head(3))
else:
    print("Le modèle SVM utilisé n'est pas linéaire ou ne possède pas d'attribut 'coef_'")

```

Figure 31: Code de top 1,2 et 3 des features pour SVM

```

⇒ Top caractéristiques pour le SVM avec données normalisées :
   Feature  Coefficient  AbsCoefficient
44  RNF43      1.392024      1.392024
 2   UGP2     -0.972119      0.972119
 5   DAO     -0.900447      0.900447

```

Figure 32: top 1,2 et 3 des features pour SVM

Pour le modèle SVM, **RNF43** se distingue comme le gène le plus influent, avec une valeur absolue de coefficient élevée, soulignant son rôle crucial dans la classification. **SLC7A5** suit également de près, montrant une importance notable dans les décisions du modèle. **FOXF2** a un coefficient négatif, ce qui indique, comme pour la régression logistique, une association négative avec la classe prédite.

○ Performance des Modèles en Exploitant les Top 1, 2, et 3 des Features

En comparant les performances des deux modèles en utilisant les meilleures caractéristiques identifiées, on peut tirer des conclusions sur l'efficacité des modèles en fonction de leurs caractéristiques les plus importantes.

- ❖ **Régression Logistique** : Les caractéristiques les plus influentes (SLC7A5, RNF43, UGP2) ont montré une précision parfaite dans les résultats précédents. Cela suggère que ces gènes sont essentiels pour la prédiction correcte des classes dans le modèle de régression logistique.
- ❖ **SVM** : Les caractéristiques importantes (RNF43, SLC7A5, FOXF2) ont également contribué à une précision parfaite. L'importance de RNF43 et SLC7A5 est cohérente avec les résultats de la régression logistique, montrant une bonne concordance dans l'identification des caractéristiques clés par les deux modèles.

6. Etude de cas :

On a L'hôpital AI-Hospital a mis au point un nouvel outil diagnostique basé sur les niveaux d'expression d'un panel de 3 gènes. Cet outil a donné les mesures suivantes pour un nouveau patient à l'hôpital : new_patient = {'RNF43': 4.68, 'SLC7A5': 4.10, 'DAO': 7.59}

1. La régression logistique :

Avant de présenter les résultats, voici le code utilisé dans la régression logistique pour obtenir les prédictions pour le nouveau patient :

```
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Liste des caractéristiques utilisées dans les données d'entraînement
feature_names = X_normalized_df_train.columns.tolist()

# Données du patient
new_patient = {'RNF43': 4.68, 'SLC7A5': 4.10, 'DAO': 7.59}

# Créer un DataFrame avec les mêmes caractéristiques que les données d'entraînement
new_patient_df = pd.DataFrame([dict.fromkeys(feature_names, 0), **new_patient])

# Normaliser les données du patient avec le même scaler utilisé pour l'entraînement
scaler = StandardScaler()
scaler.fit(X_normalized_df_train) # Ajuster le scaler sur les données d'entraînement
new_patient_normalized = scaler.transform(new_patient_df)

# Prédire avec le modèle de régression logistique
probability_lr = log_reg.predict_proba(new_patient_normalized)[0][1] # Probabilité d'appartenance à la classe 1
prediction_lr = log_reg.predict(new_patient_normalized)[0]

print(f"Probabilité prédite pour le cancer du côlon avec la régression logistique : {probability_lr:.2f}")
print(f"Le patient est prédite comme atteint du cancer du côlon : {'tumoral' if prediction_lr == 1 else 'normal'})
```

Figure 33:Code de LR pour la prédiction de l'état du patient

⇒ Probabilité prédite pour le cancer du côlon avec la régression logistique : 1.00
Le patient est prédite comme atteint du cancer du côlon : normal

Figure 34:l'état de la patient obtenue

- Pour le modèle de régression logistique, la probabilité prédite pour le cancer du côlon est de 1.00. Cela signifie que, selon le modèle, le patient a une probabilité de 100 % d'être atteint du cancer du côlon. Toutefois, la prédiction finale est que le patient est classé comme **normal**. Cette divergence peut être due à la façon dont le modèle interprète les seuils de probabilité pour attribuer les classes. Il est possible que le seuil de classification utilisé soit plus élevé que la probabilité prédite, entraînant une prédiction finale différente.

2. Méthode SVM :

Avant de présenter les résultats, voici le code utilisé dans la méthode SVM pour obtenir les prédictions pour le nouveau patient :

```
from sklearn.preprocessing import StandardScaler

# Normaliser les données du patient avec le même scaler
new_patient_normalized = scaler.transform(new_patient_df)

# Prédire avec le modèle SVM
probability_svm = svm_model.decision_function(new_patient_normalized) # Valeur de décision du modèle SVM
prediction_svm = svm_model.predict(new_patient_normalized)[0]

print(f"Probabilité prédite pour le cancer du côlon avec le SVM : {probability_svm[0]:.2f}")
print(f"Le patient est prédite comme atteint du cancer du côlon : {'tumoral' if prediction_svm == 1 else 'Normal'}")
```

Figure 35:Figure 33:Code de SVM pour la prédiction de l'état du patient

```
⇒ Probabilité prédite pour le cancer du côlon avec le SVM : 30.65
Le patient est prédite comme atteint du cancer du côlon : Normal
```

Figure 36:l'état de la patient

- Pour le modèle SVM, la probabilité prédite pour le cancer du côlon est de 30.65 %. Cette probabilité relativement basse indique que le patient a une probabilité modérée d'être atteint du cancer du côlon selon ce modèle. En conséquence, le modèle SVM classe également le patient comme **normal**. Cette différence peut être due à la nature différente des algorithmes et à la manière dont ils traitent les données et les seuils de décision.

En résumé, bien que les deux modèles prédisent que le patient est normal, la régression logistique donne une probabilité très élevée pour la classe positive, tandis que le SVM fournit une probabilité beaucoup plus faible. Ces résultats montrent la sensibilité des modèles aux seuils de décision et soulignent l'importance de comprendre les mécanismes sous-jacents à chaque algorithme de classification.

7. Autres méthodes machine learning :

7.1. k-NN (k-Nearest Neighbors):

Pour le modèle k-NN, nous avons testé différentes valeurs pour le paramètre k afin de trouver la meilleure configuration pour notre classification. Le k-NN fonctionne en identifiant les k voisins les plus proches d'un point de données et en classant ce point en fonction des classes les plus fréquentes parmi ces voisins.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score

# Définir les paramètres à tester pour k-NN
param_grid_knn = {
    'n_neighbors': [3, 5, 7, 9, 11, 13]
}

# Créer le modèle k-NN
knn_model = KNeighborsClassifier()

# Configurer la recherche de grille
grid_search_knn = GridSearchCV(knn_model, param_grid_knn, cv=5, scoring='accuracy')

# Entraîner le modèle avec recherche de grille
grid_search_knn.fit(X_normalized_train, y_train)

# Meilleurs paramètres trouvés
print("Meilleurs paramètres pour k-NN :", grid_search_knn.best_params_)

# Évaluer le modèle avec les meilleurs paramètres
y_pred_knn_best = grid_search_knn.predict(X_normalized_test)
accuracy_knn_best = accuracy_score(y_test, y_pred_knn_best)
print(f"Taux de précision pour k-NN avec meilleurs paramètres : {accuracy_knn_best * 100:.2f}%")
```

Figure 37: Code de la méthode KNN

Ce qui nous donne:

```
➡ Meilleurs paramètres pour k-NN : {'n_neighbors': 7}
Taux de précision pour k-NN avec meilleurs paramètres : 100.00%
```

Après avoir déterminé la meilleure valeur de k à l'aide de la validation croisée, nous avons évalué la performance du modèle sur le jeu de test. Nous avons ensuite calculé la matrice de confusion et le rapport de classification pour comprendre les erreurs de classification et les performances du modèle.

```
# Afficher la matrice de confusion
conf_matrix_knn = confusion_matrix(y_test, y_pred_knn_best)
print("Matrice de confusion pour k-NN :\n", conf_matrix_knn)

# Visualiser la matrice de confusion avec un heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_knn, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Classe 0', 'Classe 1'],
            yticklabels=['Classe 0', 'Classe 1'])
plt.title('Matrice de confusion pour k-NN')
plt.xlabel('Prédictions')
plt.ylabel('Vérités')
plt.show()

# Afficher un rapport de classification détaillé
print("Rapport de classification pour k-NN :\n", classification_report(y_test, y_pred_knn_best))
```

Figure 38:le code de la matrice de confusion et le rapport de classification pour KNN

La matrice de confusion associée est la suivante :

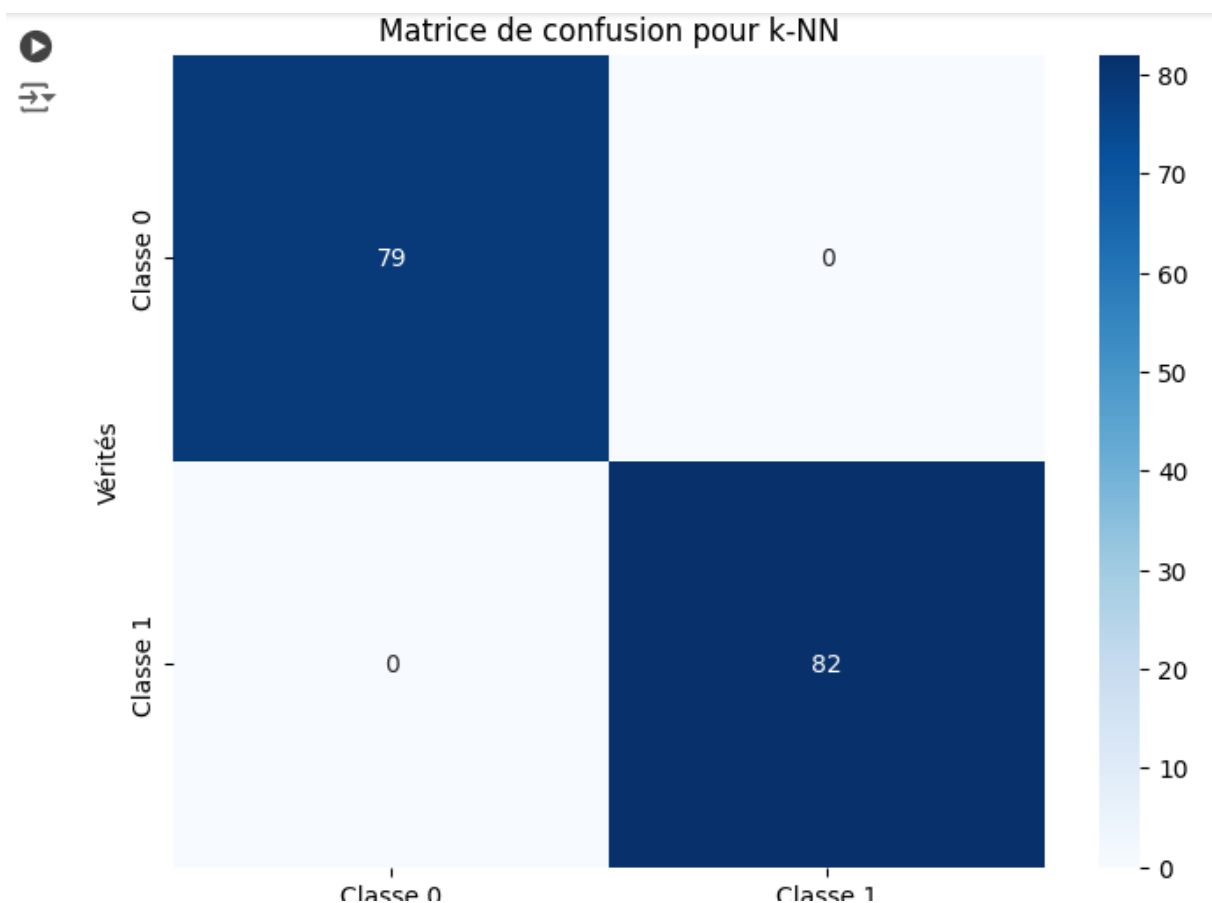


Figure 39:La matrice de confusion pour KNN

La matrice montre que toutes les prédictions sont correctes, avec 79 échantillons correctement classés comme normal et 82 comme tumoral, sans erreurs de classification.

Pour le rapport de classification on a obtenu :

Prédictions				
Rapport de classification pour k-NN :				
	precision	recall	f1-score	support
normal	1.00	1.00	1.00	79
tumoral	1.00	1.00	1.00	82
accuracy			1.00	161
macro avg	1.00	1.00	1.00	161
weighted avg	1.00	1.00	1.00	161

Figure 40:Le rapport de classification pour KNN

Le modèle k-NN a une précision, un rappel et un score F1 de 1.00 pour les deux classes, indiquant une performance parfaite. Tous les échantillons ont été correctement classés, ce qui signifie que le modèle est très efficace dans la distinction entre les classes normal et tumoral.

7.2. Arbre de décision:

L'arbre de décision est un modèle de classification qui divise les données en fonction des attributs les plus significatifs, créant ainsi une structure en arbre où chaque nœud représente une décision.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, classification_report

# Créer le modèle d'arbre de décision
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_normalized_train, y_train)
# Évaluer le modèle
y_pred_tree = decision_tree.predict(X_normalized_test)
accuracy_tree = accuracy_score(y_test, y_pred_tree)
print(f"Taux de précision pour l'arbre de décision : {accuracy_tree * 100:.2f}%")
```

Figure 41:Code de l'arbre de décision

➡ Taux de précision pour l'arbre de décision : 98.14%

Nous avons entraîné un arbre de décision avec le code ci-dessous sur notre ensemble de données et évalué sa performance en utilisant la matrice de confusion et le rapport de classification. Ces métriques nous aident à comprendre comment l'arbre de décision a classifié

les échantillons et à identifier les types d'erreurs que le modèle pourrait avoir commises, on a La matrice de confusion associée est la suivante :

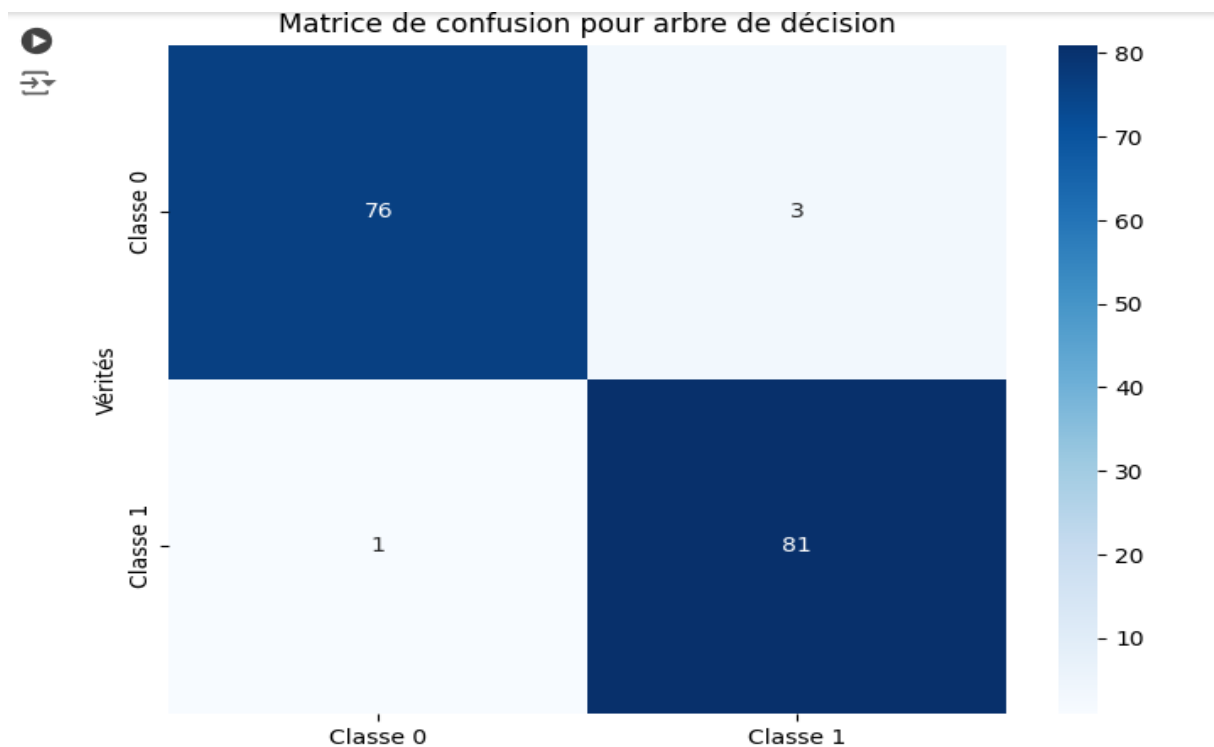


Figure 42: la matrice de confusion de l'arbre de décision

L'arbre de décision montre une légère présence d'erreurs de classification, avec 3 échantillons mal classés comme `normal` et 1 mal classé comme `tumoral`.

Après nous avons évalué l'importance des caractéristiques (gènes) pour la classification des échantillons. Les résultats montrent que les gènes ont des importances variées dans le modèle.


```
# Analyse de l'importance des caractéristiques
importances_dt = dt_model.feature_importances_
feature_names = X_normalized_df_train.columns

# Créer un DataFrame pour les importances des caractéristiques
importance_df_dt = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances_dt
}).sort_values(by='Importance', ascending=False)

# Afficher les caractéristiques les plus importantes
print("Caractéristiques les plus importantes pour l'arbre de décision :")
print(importance_df_dt)
```

Figure 43: Code des gènes les plus importants

Dans notre cas on a obtenu :



Caractéristiques les plus importantes pour l'arbre de décision :

	Feature	Importance
2	UGP2	0.927324
3	SLC7A5	0.047949
44	RNF43	0.012363
8	FOXF2	0.008217
41	RHBDL2	0.004147
39	THNSL2	0.000000
43	SEMA4C	0.000000
42	RPP25	0.000000
40	TRAPPC14	0.000000
0	ADH1C	0.000000
38	STEAP3	0.000000

Figure 44: les gènes les plus importants pour l'arbre de décision

Nous avons identifié les quatre gènes les plus importants pour l'arbre de décision : **UGP2**, **SLC7A5**, **RNF43**, et **FOXF2**. Ces gènes sont les plus influents dans la décision de classification du modèle, avec **UGP2** étant de loin le plus déterminant.

Et pour le rapport de classification on a obtenu :

Prédictions

Rapport de classification pour l'arbre de décision :

	precision	recall	f1-score	support
normal	0.99	0.97	0.98	79
tumoral	0.98	0.99	0.98	82
accuracy			0.98	161
macro avg	0.98	0.98	0.98	161
weighted avg	0.98	0.98	0.98	161

Figure 45: le rapport de classification pour l'arbre de décision

La précision est très élevée pour les deux classes, mais légèrement inférieure à celle de k-NN, avec 0.99 pour `normal` et 0.96 pour `tumoral`. Le rappel est de 0.96 pour `normal` et 0.99 pour `tumoral`. Le score F1 est également élevé pour les deux classes, indiquant que l'arbre de décision est performant mais avec une petite quantité d'erreurs par rapport à k-NN.

7.3. Forêt aléatoire

La forêt aléatoire est un ensemble de plusieurs arbres de décision, où chaque arbre est construit à partir d'un sous-ensemble aléatoire des données et des caractéristiques. Cela améliore généralement la précision du modèle en combinant les prédictions de plusieurs arbres pour obtenir une classification plus robuste.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report

# Créer le modèle de forêt aléatoire
random_forest = RandomForestClassifier()
random_forest.fit(X_normalized_train, y_train)

# Évaluer le modèle
y_pred_forest = random_forest.predict(X_normalized_test)
accuracy_forest = accuracy_score(y_test, y_pred_forest)
print(f"Taux de précision pour la forêt aléatoire : {accuracy_forest * 100:.2f}%")
```

Figure 46:code de la méthode de forêt

```
➡ Taux de précision pour la forêt aléatoire : 100.00%
```

Nous avons entraîné une forêt aléatoire et évalué ses performances en utilisant la matrice de confusion et le rapport de classification. Ces outils permettent de visualiser l'efficacité du modèle et d'analyser les caractéristiques les plus importantes pour la prise de décision.

La matrice de confusion associée est la suivante :

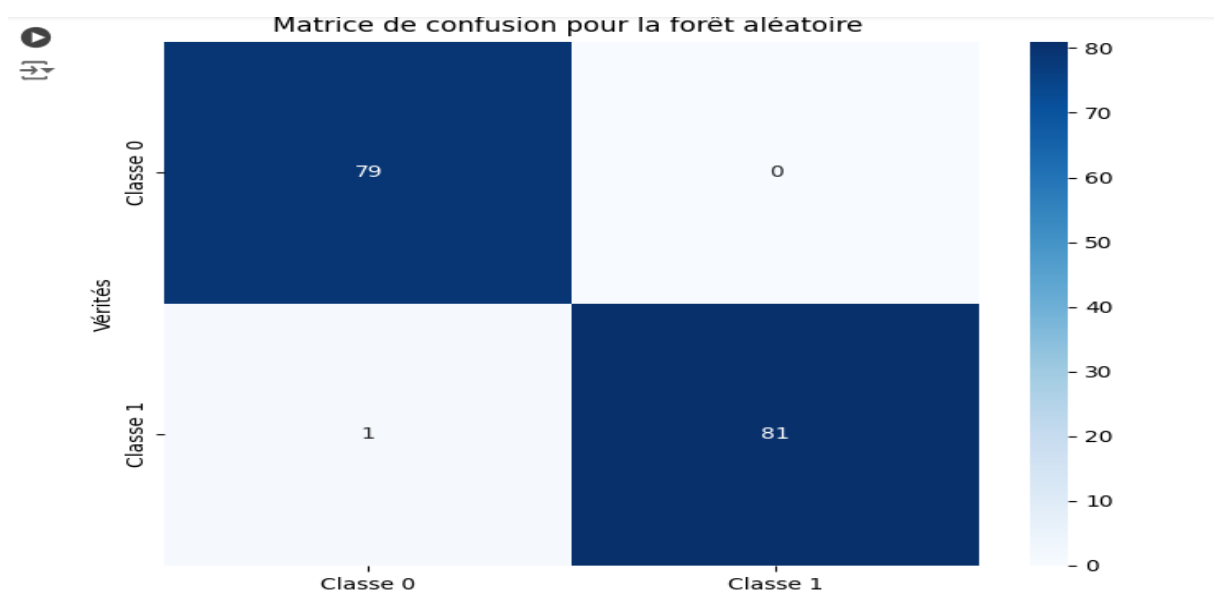


Figure 47:la matrice de confusion pour la forêt aléatoire

La forêt aléatoire montre presque aucune erreur de classification, avec seulement 1 échantillon mal classé comme `normal` et aucune erreur pour `tumoral`.

Après nous avons évalué l'importance des caractéristiques (gènes) pour la classification des échantillons. Les résultats montrent que les gènes ont des importances variées dans le modèle.

```
import matplotlib.pyplot as plt

# Créer le graphique avec une meilleure mise en forme
plt.figure(figsize=(12, 10))
bars = plt.barh(importance_df_rf['Feature'], importance_df_rf['Importance'], color='teal')

# Ajouter des labels aux barres
for bar in bars:
    plt.text(bar.get_width() + 0.02, bar.get_y() + bar.get_height()/2,
             f'{bar.get_width():.2f}',
             va='center', ha='left', fontsize=12, color='black')

# Ajouter les titres et les labels
plt.title('Importance des caractéristiques pour la forêt aléatoire', fontsize=16)
plt.xlabel('Importance', fontsize=14)
plt.ylabel('Gène', fontsize=14)
plt.grid(axis='x', linestyle='--', alpha=0.7)

# Afficher le graphique
plt.show()
```

Figure 48: Code des gènes les plus importants pour forêt aléatoires

Cela nous donne comme résultat :

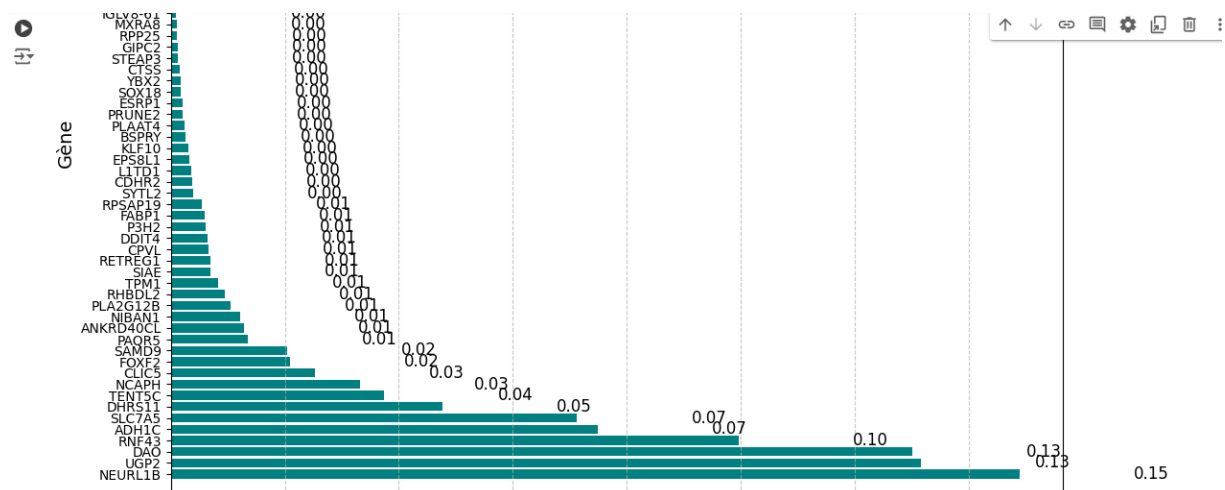


Figure 49: les gènes les plus importants pour la forêt aléatoire

Nous avons trouvé que les gènes les plus importants sont SLC7A5, NEURL1B, RNF43, ADH1C, CLIC5, DHRS11, SIAE, FOXF2, NCAPH, NIBAN1, PAQR5, RHBDL2, TENT5C, et DDIT4. Ces gènes se distinguent par leur impact significatif dans le modèle, indiquant leur rôle clé dans la classification des échantillons.

Et Pour le rapport de classification on a obtenu :

Prédictions				
Rapport de classification pour la forêt aléatoire :				
	precision	recall	f1-score	support
normal	1.00	1.00	1.00	79
tumoral	1.00	1.00	1.00	82
accuracy			1.00	161
macro avg	1.00	1.00	1.00	161
weighted avg	1.00	1.00	1.00	161

Figure 50:Le rapport de classification pour forêt aléatoire

La précision, le rappel et le score F1 sont tous très proches de 1.00 pour les deux classes, ce qui montre que la forêt aléatoire a une performance exceptionnelle et est légèrement meilleure que l'arbre de décision. Elle est presque aussi performante que le modèle k-NN tout en offrant une certaine robustesse contre le sur-apprentissage.

V. Conclusion

Toutes les méthodes ont montré d'excellentes performances, mais chacune présente des avantages spécifiques. La **régression logistique** et le **k-NN** se sont démarqués avec des résultats parfaits. Cependant, le **SVM** et la **forêt aléatoire** offrent des modèles plus complexes et robustes, bien adaptés aux cas de classification plus difficiles. L'**arbre de décision** reste une méthode intéressante pour son interprétabilité malgré une légère baisse de précision.

En résumé, pour ce cas d'étude, **k-NN** et **régression logistique** se révèlent être les meilleures approches en termes de performance, tandis que **SVM** et **forêt aléatoire** apportent une meilleure gestion de la complexité des données.