UNIVERSITE ABDELMALEK ESSAADI

Master IA et science de données

Module : Machine Learning

Projet :

Sujet 1 : Cancer de l'ovaire ou de la prostate : Gaussien vs XGBoost

Réalisé PAR :

SABBAHI MOHAMED AMINE

BOUFARHI AYMAN

KHATTABI IDRISS

ENCADRE PAR :

M'hamed AIT KBIR

Année Universitaire : 2023/2024

Table des matières:

1. Description des données :

Le jeu de données Arcene, constitue une ressource cruciale dans le domaine de l'apprentissage automatique appliqué à la classification binaire du cancer de l'ovaire ou de la prostate, ce jeu de données offre un aperçu détaillé des caractéristiques moléculaires capturées à travers la spectrométrie de masse. Les données incluent des mesures d'intensités de spectres de masse, qui sont essentielles pour la distinction entre les échantillons sains et ceux associés au cancer. Cette granularité moléculaire permet de construire des modèles de classification sophistiqués et précis.

Le jeu de données Arcene, résulte de la fusion de trois ensembles de données de spectrométrie de masse, fournissant une abondance de données d'entraînement et de test. Il se compose de caractéristiques originales détaillant l'abondance des protéines dans le sérum humain en fonction de la masse, visant à différencier les patients atteints de cancer des patients en bonne santé. Des caractéristiques distrayantes, appelées "sondes," ont été ajoutées sans pouvoir prédictif, et l'ordre des caractéristiques a été aléatoirement mélangé.

Réparties en ensembles d'entraînement, de validation et de test, les données comprennent environ 10 000 variables, dont 7 000 sont réelles et 3 000 sont des sondes. Le jeu de données est un défi de sélection de caractéristiques NIPS 2003.

Le tableau ci-dessous illustre la distribution des exemples positifs et négatifs dans chaque ensemble :

ARCENE	**Positifs**	**Négatifs**	**Total**
Entraînement	44	56	100
Validation	44	56	100
Test	310	390	700
Total	398	502	900

Le format des données est structuré en plusieurs fichiers, notamment des fichiers de paramètres, d'identités de caractéristiques, d'ensembles d'entraînement, de validation et de test, ainsi que des fichiers d'étiquettes indiquant les vérités de classes :

- dataname.param : Ce fichier contient les paramètres et les statistiques détaillées sur les données. Il offre un aperçu global des propriétés du jeu de données.

• dataname.feat : Ce fichier identifie les caractéristiques, cependant, ces informations sont délibérément retirées pour éviter tout biais potentiel dans le processus de sélection des caractéristiques. Cela garantit une évaluation impartiale des modèles.

• dataname_train.data : Il s'agit du fichier décrivant l'ensemble d'entraînement. Les données sont présentées sous forme d'une matrice régulière délimitée par des virgules, avec les motifs disposés en lignes et les caractéristiques en colonnes. Ces données constituent la base sur laquelle les algorithmes d'apprentissage automatique seront formés.

• dataname_valid.data : Similaire au fichier d'entraînement, ce fichier contient l'ensemble de validation. Il est utilisé pour évaluer les performances des modèles pendant la phase de développement.

• dataname_test.data : Ce fichier décrit l'ensemble de test, une partie cruciale du processus d'évaluation. Les modèles formés sur l'ensemble d'entraînement sont testés sur ces données pour évaluer leur généralisation aux nouvelles observations.

• dataname_train.labels : Les étiquettes associées aux exemples de l'ensemble d'entraînement. Elles représentent les valeurs de vérité des classes auxquelles appartiennent les exemples.

• dataname_valid.labels : Les étiquettes de l'ensemble de validation. Ces informations étaient retirées pendant le défi, mais elles sont maintenant fournies pour une analyse approfondie.

• dataname_test.labels : Les étiquettes de l'ensemble de test. Bien que retirées, cela permet l'utilisation des données comme référence pour les performances des modèles.

2. Synthèse des Travaux de Recherche

La dataset "arcene" occupe une place significative dans la recherche en bioinformatique et en apprentissage automatique, en particulier en raison de son utilisation dans le cadre du défi KDD Cup 2003. Ce jeu de données, provenant d'une étude sur la classification du cancer basée sur des données de spectrométrie de masse, a été le centre de nombreux travaux de recherche visant à développer des modèles d'apprentissage automatique capables de distinguer entre tissus sains et tissus cancéreux.

Les chercheurs ont exploré diverses approches d'apprentissage automatique, notamment des algorithmes de classification tels que les machines à vecteurs de support (SVM), les réseaux de neurones, et les méthodes d'ensemble comme les forêts aléatoires. L'objectif commun était d'optimiser la précision de la classification afin d'améliorer la détection précoce du cancer.

Les travaux de recherche ont également porté sur l'extraction et la sélection de caractéristiques pertinentes à partir des données de spectrométrie de masse, cherchant à identifier les signatures moléculaires distinctives associées à différents types de tissus. Ces efforts ont contribué à une

compréhension plus approfondie des patterns biologiques impliqués dans le développement du cancer.

Par ailleurs, la dataset arcene a été utilisée comme point de comparaison pour évaluer la performance de nouvelles méthodes et techniques émergentes dans le domaine de l'apprentissage automatique. Les résultats de ces travaux ont souvent été publiés dans des revues scientifiques spécialisées, contribuant ainsi à l'avancement des connaissances dans la compréhension et la classification des données liées au cancer à partir de la spectrométrie de masse.

En résumé, la dataset arcene a joué un rôle crucial en fournissant un banc d'essai standardisé pour évaluer les algorithmes de classification dans le contexte de la détection du cancer. Les travaux de recherche associés ont contribué à l'amélioration des méthodes diagnostiques et ont ouvert la voie à de nouvelles avenues pour la recherche en bio-informatique et en oncologie.

3. Prétraitement des données :

Le prétraitement des données est une étape cruciale dans le processus d'analyse des données et d'apprentissage automatique. Il vise à préparer, nettoyer et transformer les données brutes pour les rendre adaptées à l'application de modèles d'apprentissage automatique. Voici quelques prétraitements on fait pour Arcene datasets :

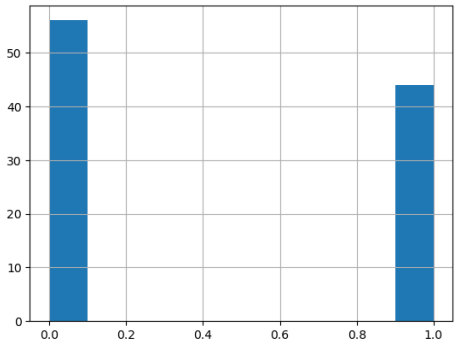
1.3 Préparation pour l'apprentissage :

Dans la phase de préparation de donnes d'entrainement on a converti deux fichiers d'Arcene de données textuelles (caractéristiques et étiquettes) en fichiers CSV pour faciliter leur utilisation, dans les tâches d'apprentissage automatique, en les organisant en colonnes distinctes avec des entêtes appropriées, les colonnes sont nommées de feature_1 jusqu'à feature_1000, et la colonne d'étiquettes nommé labels avec le structure suivant :

- Lecture des données du fichier 'arcene/ARCENE/arcene_train.data'.
- Création d'un fichier CSV 'csv_arcene_train.csv' avec des entêtes de colonnes 'feature_1' à 'feature_10000' pour les 10 000 caractéristiques.
- Traitement des étiquettes correspondantes du fichier 'arcene/ARCENE/arcene_train.labels'.
- Stockage des données et des étiquettes respectivement dans les fichiers 'csv_arcene_valid.csv' et 'csv_arcene_valid_labels.csv'.

Après on fait la même chose pour traiter le jeu de donnes de test, on a choisi arcene_valid.data comme notre test data :

- Lecture des données du fichier 'arcene/ARCENE/arcene_train.data'.
- Création d'un fichier CSV 'csv_arcene_train.csv' avec des entêtes de colonnes 'feature_1' à 'feature_10000' pour les 10 000 caractéristiques.
- Traitement des étiquettes correspondantes du fichier 'arcene/ARCENE/arcene_train.labels'.



Histogramme des valeurs présentes dans la colonne labels
Les cas tester Positive (+1) : 44
Les cas tester Négative (0) : 56

- Stockage des données et des étiquettes respectivement dans les fichiers 'csv_arcene_valid.csv' et 'csv_arcene_valid_labels.csv'.

Après on a remplacent toutes les occurrences de -1 par 0 dans les variables des classes d'appartenance test_labels et train_labels.

On utilise cette méthode pour simplifier la manipulation de dataset. Cette opération est souvent utilisée dans des problèmes de classification où les étiquettes peuvent être codées différemment.

Et finalement on a fusionné les caractéristiques (features) d'entraînement avec leurs étiquettes et de même on a aussi fusionné les caractéristiques (features) d'entraînement avec leurs étiquettes.

2.3 Minimisation des données :

La minimisation des données, ou "feature selection" en anglais, est une pratique fréquemment utilisée en machine learning pour sélectionner les caractéristiques (features) les plus importantes ou les plus pertinentes pour un modèle.

Notre ensemble de données contient 10 000 attributs, ce qui est trop et qui contiennent des valeurs qui ne donnent pas de résultats importants. Notre ensemble de données nécessite donc une sélection de fonctionnalités et pour y parvenir, nous suggérons 2 techniques :

1.2 Elimination des colonnes qui n'ont que des 0 ou qui ont plus de 65 zéro :

On cette partie identifie les colonnes dans train_dataset contenant 65 zéros ou plus et les supprime de train_dataset et test_dataset, probablement car ces colonnes sont considérées comme ayant une proportion significative de zéros, ce qui peut potentiellement ne pas être informatif pour un modèle d'apprentissage automatique, et nous avons réduit notre jeu de données à 2944 colonnes.

2.2 La minimisation par le calcul de corrélation :

Est une méthode courante pour réduire le nombre de caractéristiques dans un ensemble de données. La corrélation mesure la relation et l'indépendance entre deux variables.

Dans cette partie on a identifié les paires de caractéristiques fortement corrélées au-dessus d'un seuil de 0.9 et supprime une colonne de chaque paire de caractéristiques corrélées, garantissant ainsi une moindre redondance d'informations dans les ensembles de données.

Après le processus de minimisation, nous restent avec un ensemble de données filtré comportant 100 lignes et 2220 caractéristiques.

4. Implémentation des techniques d'apprentissage automatique :

1.4 Implémentation de modèle Gaussien de native bayes pour la classification :

Le modèle naïf bayésien gaussien (Gaussian Naive Bayes en anglais) est une variante du classificateur naïf bayésien classique. Ce modèle est souvent utilisé pour des tâches de classification où les caractéristiques sont continues et peuvent être modélisées par une distribution gaussienne, bien que

son hypothèse de caractéristiques indépendantes (d'où le terme "naïf") puisse ne pas toujours correspondre à la réalité dans des situations plus complexes.

Et on fait cette implémentation par ecrire les fonctions suivantes :

calcule_priori : cette fonction Calcule la probabilité a priori de chaque classe en comptant le nombre d'occurrences de chaque classe et en le divisant par la taille totale de l'ensemble de données.

calcule_vraisemblance(df, feat_name, feat_val, Y, label): Cette fonction est responsable du calcul de la vraisemblance d'une valeur de caractéristique pour une classe donnée dans le contexte du classifieur Naïve Bayes gaussien.

La probabilité est calculée en utilisant la formule de la densité de probabilité de la distribution gaussienne pour la valeur feat_val étant donné la classe label

$$f(x) = \frac{1}{\sigma\sqrt{2\,\pi}}\,\mathrm{e}^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Avec :
x : La valeur pour laquelle on souhaite calculer la densité de probabilité.
μ : La moyenne de la distribution gaussienne.
σ : L'écart-type de la distribution gaussienne.

naive_bayes_gaussian() : Cette fonction utilise les fonctions calcule_priori() et calcule_vraisemblance() pour estimer les probabilités a priori et les probabilités conditionnelles respectivement, afin de prédire les classes pour les exemples donnés.

Alors cette fonction Calcule de P(X=x1|Y=y)P(X=x2|Y=y)...P(X=xn|Y=y) * P(Y=y) pour tout les calsse y et retourner le maximum.

2.4 Implémentation de modèle XGBoost pour la classification :

XGBoost (Extreme Gradient Boosting) est une méthode d'apprentissage ensembliste qui utilise des arbres de décision comme apprenants de base. Plus précisément, il construit un ensemble d'arbres de décision de manière séquentielle, où chaque nouvel arbre tente de corriger les erreurs commises par les précédents.

XGBoost est particulièrement efficace avec les données numériques continues. Il gère bien les caractéristiques numériques et peut identifier des schémas complexes à l'intérieur de ces types de variables.

XGBoost est basé sur le « gradient boosting », qui construit des modèles de manière séquentielle, chacun corrigeant les erreurs commises par les précédents. Il minimise une fonction de perte (loss function) prédéfinie en ajoutant des apprenants faibles, chaque nouvel apprenant se concentrant sur les erreurs de l'ensemble existant.

Nous avons réalisé cette implémentation en divisant le code en deux parties, chaque partie ayant la fonction requise pour l'algorithme :

1.2 La partie de l'arbre de décision :

calculate_entropy(y) : cette fonction calcule l'entropie d'un ensemble de valeurs cibles y suivant la formule de l'entropy: $H(X) = -\sum_{i=1}^{n} p(x_i) \cdot \log_2(p(x_i))$

split_dataset() : Cette fonction divise un ensemble de données en deux parties en fonction d'une caractéristique spécifique et d'un seuil donné.

find_best_split() : Cette fonction cherche la meilleure caractéristique et le meilleur seuil pour diviser les données en minimisant l'entropie.

grow_tree() : Cette fonction est responsable de la construction récursive d'un arbre de décision.

predict_tree() et predict_single() : Ces deux fonctions, predict_tree() et predict_single(), sont utilisées pour effectuer des prédictions à l'aide de l'arbre de décision construit précédemment.

2.2 La partie de de XGBoost :

calculate_gradient() : Cette fonction calculate_gradient() est utilisée dans le contexte de l'algorithme de boosting pour calculer les gradients des prédictions et cette fonction degsin notre fonction de loss.

fit_tree() : Cette fonction fit_tree(X, residuals, max_depth) utilise la fonction grow_tree() pour ajuster un arbre à partir des résidus fournis jusqu'à une profondeur maximale spécifiée.

fit(X, y, n_estimators=7, learning_rate=0.1, max_depth=2) : Cette fonction met en œuvre un processus de boosting où chaque nouvel arbre est entraîné pour corriger les erreurs résiduelles laissées par le modèle précédent. Ensuite, le modèle global est mis à jour en ajoutant progressivement les prédictions de chaque nouvel arbre, pondérées par le taux d'apprentissage.

Avec :
n_estimators: Le nombre d'estimateurs (arbres) à entraîner.
learning_rate: Le taux d'apprentissage qui contrôle la contribution de chaque arbre à la mise à jour du modèle global.
max_depth: La profondeur maximale des arbres utilisés comme estimateurs.

predict(X, trees, learning_rate): Cette fonction predict(X, trees, learning_rate) est utilisée pour faire des prédictions à l'aide du modèle de boosting entraîné. Cette fonction combine les prédictions de chaque arbre entraîné dans le modèle de boosting, en les pondérant par le taux d'apprentissage, pour obtenir la prédiction finale du modèle global sur les données X.
Avec :
X: Les données pour lesquelles vous voulez faire des prédictions.
trees: La liste des arbres entraînés dans le modèle de boosting.
learning_rate: Le taux d'apprentissage utilisé pour ajuster l'influence de chaque arbre sur les prédictions globales.

XGBoost_classifier((X_train, y_train, X_test, n_estimators=7, learning_rate=0.1, max_depth=2) : Cette fonction XGBoost_classifier() utilise les fonctions précédemment définies pour former un modèle de type XGBoost simplifié et pour prédire sur un ensemble de données de test.

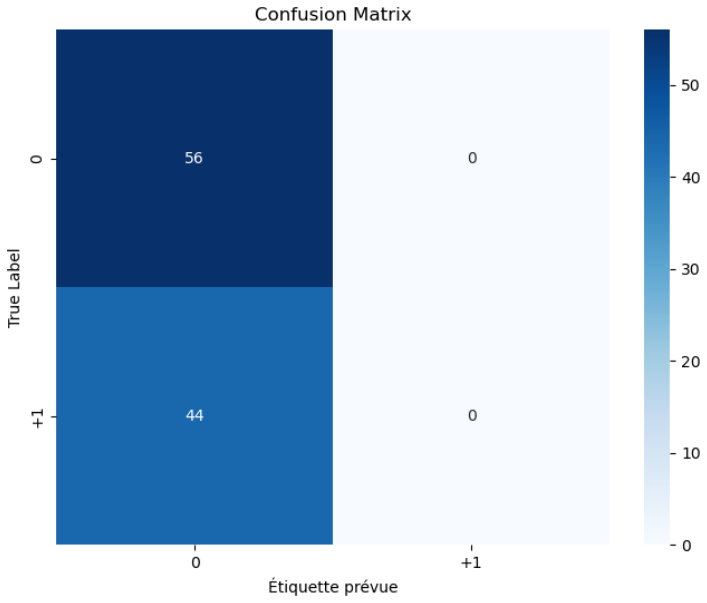
n_estimators: Le nombre d'arbres à entraîner dans le modèle de boosting.
learning_rate: Le taux d'apprentissage qui contrôle la contribution de chaque arbre à la mise à jour du modèle global.
max_depth: La profondeur maximale des arbres utilisés comme estimateurs.

5. Présentation des résultats les différentes métriques :

1.5 Résultat du modèle Gaussien de native bayes :

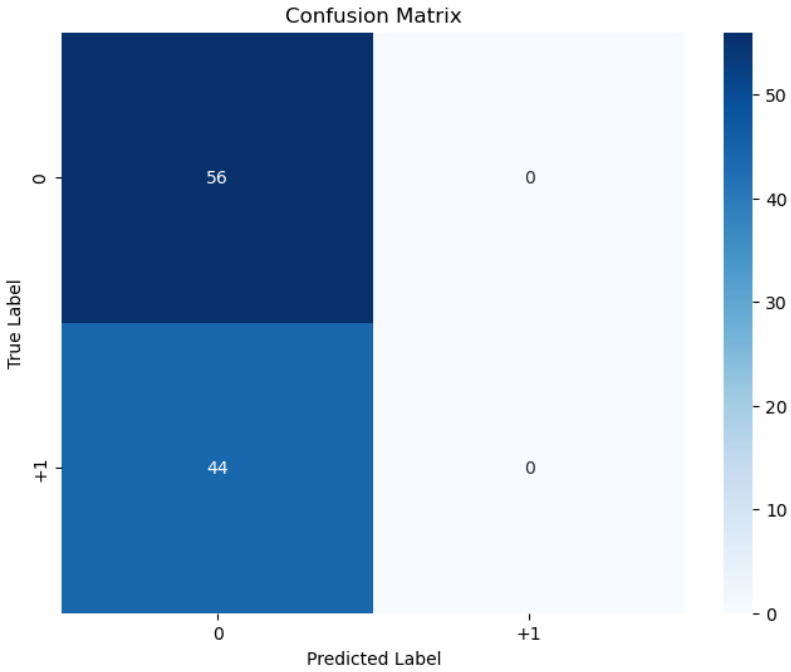
	precision	recall	f1-score	support
0	0.56	1.00	0.72	56
1	0.00	0.00	0.00	44
accuracy			0.56	100
macro avg	0.28	0.50	0.36	100
weighted avg	0.31	0.56	0.40	100

Ce modèle a une précision de 56 % pour la classe 0, mais il ne parvient pas à prédire correctement la classe 1, avec une précision de 0 %. En ce qui concerne le F1-score, une mesure combinée de la précision et du rappel, le modèle présente un score de 0,72 pour la classe 0 et un score de 0 pour la classe 1. Il identifie parfaitement tous les exemples de la classe 0 (rappel de 100 %), mais ne capture aucun exemple de la classe 1 (rappel de 0 %). L'exactitude globale du modèle est de 56 %.



2.5 Résultat du modèle XGBoost :

	precision	recall	f1-score	support
0	0.56	1.00	0.72	56
1	0.00	0.00	0.00	44
accuracy			0.56	100
macro avg	0.28	0.50	0.36	100
weighted avg	0.31	0.56	0.40	100

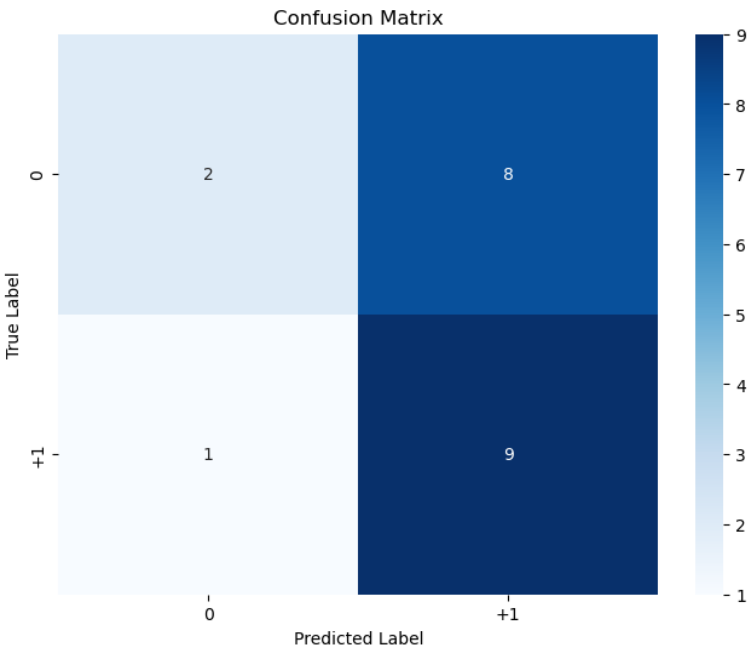


Ce modèle montre une précision de 56 % pour la catégorie 0, mais il ne parvient pas à prédire la catégorie 1 (précision de 0 %). Il identifie tous les éléments de la catégorie 0 (rappel de 100 %), mais n'en identifie aucun pour la catégorie 1 (rappel de 0 %). L'exactitude globale du modèle est de 56 %.

Pour évaluer bien notre model on fait l'entrainement sur le train_dataset uniqement est on separer le dataset avec train_test_split de sklearn, 80% pour l'entarinement et 20% pour le test.

	precision	recall	f1-score	support
0	0.67	0.20	0.31	10
1	0.53	0.90	0.67	10
accuracy			0.55	20
macro avg	0.60	0.55	0.49	20
weighted avg	0.60	0.55	0.49	20

Pour la classe 0, le modèle affiche une précision de 67 %. Cela signifie que parmi toutes les prédictions faites pour cette classe, 67 % sont correctes. Le rappel pour la classe 0 est de 20 %. Cela indique que le modèle a réussi à identifier correctement seulement 20 % des exemples réels de la classe 0. Le score F1 pour la classe 0 est de 0,31, une mesure combinée de précision et de rappel. Pour la classe 1, le modèle affiche une précision de 53 %. Cela signifie que parmi toutes les prédictions faites pour cette classe, 53 % sont correctes. Le rappel pour la classe 1 est de 90 %. Cela indique que le modèle a réussi à identifier correctement 90 % des exemples réels de la classe 1. Le score F1 pour la classe 1 est de 0,67, une mesure combinée de précision et de rappel. L'exactitude globale du modèle est de 55 %,.



6. Comparaison des résultats trouvés avec modules des librairies sklearn :

1.6 Gaussiane VS GaussianNB() de la bibliothèque Sklearn de Python :

Le premier modèle qu'on implémenter a une précision de 0.56 pour la classe 0, indiquant qu'il identifie correctement environ 56% des éléments de cette classe. Cependant, pour la classe 1, sa performance est très basse avec une recall de 0.00, ce qui signifie qu'il ne parvient pas à identifier correctement les éléments de cette classe. L'accuracy globale du modèle est de 0.56.

Pour le modèle de SKlearn, la précision pour la classe 0 est de 0.70, et il a une recall assez élevée de 0.91 pour cette classe. Pour la classe 1, sa performance est également meilleure avec une précision de 0.81 et une recall de 0.50. L'accuracy du modèle 2 est de 0.73.

En comparant les deux modèles, le modèle de sklearn surpasse clairement le modèle 1 en termes de performances. Il a des scores de précision, de recall et de f1-score plus élevés pour les deux classes, ce qui se traduit par une meilleure accuracy globale.

2.6 XGBoost VS XGBoost de la bibliothèque XGBoost de Python :

Pour le premier modèle XGBoost que nous avez implémenter, la précision pour la classe 0 est de 0.67 et pour la classe 1 est de 0.53. Cependant, le modèle présente une meilleure recall pour la classe 1 (0.90) par rapport à la classe 0 (0.20). L'accuracy globale de ce modèle est de 0.55.

En revanche, le modèle XGBoost de la bibliothèque Python affiche de meilleures performances. Il présente des scores de précision, de recall et de f1-score supérieurs pour les deux classes. Avec une précision de 0.70 pour la classe 0 et de 0.73 pour la classe 1, ainsi qu'une recall plus équilibrée pour les deux classes (0.84 pour la classe 0 et 0.55 pour la classe 1), ce modèle atteint une accuracy de 0.71, ce qui est considérablement plus élevé que le premier modèle XGBoost que vous avez mentionné.Cette différence peut être due à divers facteurs tels que les hyperparamètres utilisés, la taille ou la qualité des données d'entraînement, ou même des techniques de prétraitement différens.

3.6 Comparaison entre les deux modèles Gaussiane vs XGBoost implémenté :

Pour le modèle Gaussiane :

Il a une précision de 0.56 pour la classe 0, ce qui signifie qu'il identifie correctement environ 56% des éléments de cette classe. Cependant, il n'identifie aucun élément de la classe 1, ce qui donne un f1-score de 0.00 pour cette classe. L'accuracy globale du modèle est de 0.56.

Pour le modèle XGBoost :

Sa précision pour la classe 0 est de 0.67, mais sa recall est relativement faible à 0.20 pour cette classe. Pour la classe 1, il a une précision de 0.53 et une recall plus élevée à 0.90. L'accuracy du modèle est de 0.55.

En général on peut dire que XGBoost est le meilleur modèle pour ce type de jeu de données.

7. Conclusion :

Dans ce projet, le prétraitement des données a été approfondi, en passant par la transformation des données brutes, la sélection de caractéristiques pertinentes et l'implémentation des modèles gaussien de native bayes et XGBoost.

Le prétraitement des données a inclus des étapes telles que la conversion des fichiers sources en formats adaptés pour l'apprentissage automatique, la minimisation des données en éliminant les colonnes peu informatives et la corrélation entre les caractéristiques, ainsi que l'application de modèles spécifiques comme le modèle Gaussien de Naïve Bayes et XGBoost pour la classification.

Les résultats obtenus ont été évalués à travers des métriques telles que la précision, le rappel et le F1-score pour chaque modèle, permettant de comprendre les performances relatives de chacun. Ces évaluations ont été menées à la fois sur des données d'entraînement et de test, fournissant ainsi une vue complète des performances des modèles.

Comparativement, l'évaluation des modèles implémentés par rapport aux modèles de référence de la bibliothèque Sklearn a mis en évidence des différences notables, montrant que les modèles de la bibliothèque Sklearn surpassent les performances des modèles implémentés, en termes de précision, de rappel et de F1-score, conduisant à une meilleure exactitude globale.

Enfin, la comparaison entre les modèles implémentés eux-mêmes, en l'occurrence le modèle Gaussien et XGBoost, a révélé que XGBoost a montré de meilleures performances générales, avec des scores de précision plus élevés pour les deux classes, malgré des valeurs de rappel variables.

En résumé, bien que l'implémentation des modèles ait été instructive, les modèles de référence de la bibliothèque Sklearn ont démontré des performances supérieures. Cela met en lumière l'importance de l'exploration de modèles existants et la compréhension approfondie des données pour obtenir des résultats optimaux en apprentissage automatique.

8. Biographie :

[1] Nand Sharma ; Prathamesh Verlekar; Rehab Ashary; Sui Zhiquan, Regularization and feature selection for large dimensional data.

[2] https://www.geeksforgeeks.org/xgboost/

[3] https://simonwenkel.com/2019/03/09/revisiting-ML-datasets-arcene.html

[4] XGBoost classification: https://www.youtube.com/watch?v=8b1JEDvenQU&t=22s

[5] Xgboost Classification In-depth Maths Intuition: https://www.youtube.com/watch?v=gPciUPwWJQQ&t=599s

[6] XGBoost in Python from Start to Finish : https://www.youtube.com/watch?v=GrJP9FLV3FE&t=2421s

[7] Gaussian Naive Bayes Classifier Algorithm : www.youtube.com/watch?v=kufuBE6TJew&t=201s

[8] Gaussian naive Bayes, Clearly Explained : https://www.youtube.com/watch?v=H3EjCKtlVog

[9] https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/

[10] www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/