L3S2 MIASHS - Introduction à l'apprentissage automatique

Application d'algorithmes de classification supervisée pour prédire la satisfaction de passagers d'une compagnie aérienne

Dossier de validation

Estelle Long-Merle | No étudiant : 12106173

Université Grenoble Alpes
Licence 3 MIASHS

Table des matières

1 Description du jeu de données et présentation de la problématique

Le jeu de données utilisé est "airline passenger satisfaction" qui décrit la satisfaction des passagers à bord d'une compagnie aérienne anonyme. Il s'agit probablement de la compagnie aérienne United Airlines au vue de la classe Economy Plus. Il vise à comprendre les facteurs qui influencent la satisfaction des passagers lors de leurs voyages en avion.

Le jeu de données est disponible sur Kaggle :
https://www.kaggle.com/datasets/teejmahal20/airline-passenger-satisfaction

Le code utilisé pour le projet est disponible au lien github suivant :
https://github.com/estellelm38/airline_satisfaction

2 Pré-traitement des données

Je cherche à comprendre quels facteurs influencent la satisfaction des passagers aériens et quelle est leur influence respective. L'ensemble de données comprend un total de 129 880 observations et 22 variables, parmi lesquelles figurent des variables catégorielles, quantitatives et ordinales. La target est le niveau de satisfaction du client de la compagnie aérienne dont les modalités sont : satisfied / neutral or dissatisfied. Le jeu de données est équilibré.

Ainsi, l'objectif est d'obtenir la meilleure accuracy et la meilleure précision sur la target.
Les variables catégorielles comprennent des informations telles que le genre, le type de client, le type de voyage et la classe de vol. Les variables quantitatives comprennent l'âge, la distance de vol, ainsi que les retards au départ et à l'arrivée. Les variables ordinales sont des notes attribuées par les passagers sur les différents services à bord, tels que le wifi à bord, la commodité des horaires de départ/arrivée, la réservation en ligne, etc. (table 2)

Je commence par observer les distributions des différentes variables à l'aide d'histogrammes de countplots et de diagrammes pour mieux visualiser le jeu de données, les graphiques obtenus sont disponibles en annexe. (table 3)

2.1 Encodage

Les modèles étant basés sur des calculs mathématiques, il est nécessaire de représenter les données catégorielles et ordinales numériquement à l'aide de techniques d'encodage afin qu'elles soient comprises par les modèles.
Je cherche à encoder des features, Je vais donc utiliser les deux encodeurs proposés par scikit learn dans ce cas de figure, à savoir one hot encoder et ordinal encoder, afin d'encoder respectivement les variables catégorielles et ordinales.

J'utilise le one hot encoder pour l'encodage des variables catégorielles afin d'éviter de créer des relations d'ordre entre ces variables et pénaliser les modèles sensibles. Je vais ici décomposer les variables en sous variables et créer autant de colonnes que j'ai de catégories (celles-ci sont représentées de façon binaires). J'obtiens une sparse matrix très large qui est automatiquement compressée. Après encodage, 6 nouvelles colonnes ont été générées (4 pour les features et 2 pour la target). (figure 1)

Ensuite, pour l'encodage des variables ordinales j'utilise l'ordinal encoder. Il permet d'associer à chaque classe une valeur numérique en traitant une par une les différentes colonnes. Cela a permis de représenter les différentes catégories des variables ordinales par des entiers, tout en préservant l'ordre des catégories. (figure 2)

2.2 Mise à l'échelle

Il faut mettre toutes les features à la même échelle car par exemple l'âge est de l'ordre de la dizaine et la distance est de l'ordre du millier et certains classifieurs fonctionnent selon des distances. (figure 3)

Le MinMax scaling permet de mettre à l'échelle chaque feature pour que ses valeurs soient comprises entre 0 et 1. La formule suivante est appliquée à toutes les valeurs de chaque feature : :

$$\frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$$

avec X la valeur d'entrée, X_{min} la valeur minimum et X_{max} la valeur maximum de la feature. Ce processus permet de comparer les différentes features sans perdre d'information car les distances entre leurs valeurs restent proportionnelles entre elles. (figure 4)

J'ai également testé standard scaling et robust scaling cependant ceux-ci ont donné de moins bon résultats ce qui peut être expliqué pa la grande variance des données. Pour standard scaling afin de donner un écart type de 1, celui-ci a produit plusieurs valeurs négatives qui ont par la suite complexifié les calculs de certains modèles d'apprentissage. De plus, les valeurs aberrantes sont rares mais importants pour le jeu de données comme on le verra juste après. Ainsi il est moins intéressant d'utiliser le robust scaling qui perd de l'information en utilisant un calcul de médiane plutôt qu'une moyenne au profit d'une robustesse face aux valeurs aberrantes. Ici, je privilégie au contraire une méthode sensible à ces valeurs aberrantes.

2.3 Valeurs aberrantes

Concernant les valeurs aberrantes, j'ai choisi de les garder dans le jeu de données. En effet, en ce qui concerne les délais de départ/arrivée j'ai jugé qu'une valeur aberrante aurait une influence sur la satisfaction du passager et donc il est important de ne pas la supprimer.

Ensuite je ne supprime pas les valeurs aberrantes des distances de vol car il s'agit de valeurs plausibles (la distance maximum dans le dataset : 4983 km, est inférieure à la distance d'un vol Russie-USA : 8881 km)

2.4 Sélection des features

Je dois sélectionner les variables les plus utiles pour mes modèles afin d'améliorer leurs performances.
J'ai tracé la matrice des corrélation (figure 5) puis affiché les features qui ont une corrélation $|r| > 0.8$. (figure 6)
J'ai ensuite supprimé une feature pour chaque paire de features corrélées sur le critère de garder une bonne compréhension, par exemple en gardant seulement 'satisfied' et en supprimant 'unsatisfied'.
J'ai supprimé la feature délai de départ en minutes car il 'agissait de la seule qui comportait des valeurs manquantes et ce en grande quantité (393) (figure 7). Ainsi j'ai préféré la supprimer et garder la feature délai de départ à l'arrivée pour éviter d'avoir à imputer les valeurs manquantes.
J'ai ensuite tracé deux graphiques : l'un représentant l'importance de chaque feature (figure 8) et l'autre l'importance des features cumulée (figure 9) qui dépend du modèle de forêt aléatoires. Néanmoins, j'ai obtenu de bons résultats en gardant toutes les features.

3 Apprentissage supervisé

Pour la phase d'apprentissage supervisé, j'ai testé les 4 classifieurs suivants : Le K-Nearest Neighbors (KNN), le Naive Bayes, des forêts aléatoires et le SVC ainsi que la classification par un réseau de neurones (MLP).

J'ai décidé de représenter la qualité de l'apprentissage pour chaque modèle à l'aide de graphiques avec une courbe ROC (Receiver Operating Characteristic) c'est une mesure de la performance pour les classificateurs binaires qui donne le taux de vrais positifs en fonction du taux de faux positifs. L'air sous la courbe indique la qualité globale de l'apprentissage et est indiquée sur chaque graphique.

3.1 K plus proches voisins

L'algorithme des k plus proches voisins (KNN) est une méthode d'apprentissage supervisé utilisée pour la classification et la régression. Son fonctionnement repose sur le principe suivant : les instances similaires se regroupent dans l'espace des caractéristiques.

Dans mon projet, j'ai utilisé l'estimateur KNeighborsClassifier pour implémenter KNN. La première étape a été de diviser l'ensemble de données en un ensemble d'entraînement (80% des données) et un ensemble de test (20% des données), une pratique courante en apprentissage automatique.

L'hyperparamètre principal de KNN est le nombre de voisins (k), qui détermine le nombre d'instances les plus proches à considérer lors de la prédiction. De plus, il est nécessaire de choisir une méthode de calcul de la distance, comme la distance euclidienne, la distance de Manhattan ou la distance de Minkowski. Après avoir testé les distances euclidienne et de Manhattan, j'ai décidé d'utiliser uniquement la distance euclidienne car elle donnait de meilleurs résultats et le temps de calcul était largement inférieur à celui pour un modèle KNN basé sur la distance de Manhattan.

Pour évaluer la performance de mon modèle, j'ai utilisé la validation croisée. J'ai opté pour une validation K-fold, qui consiste à diviser l'ensemble de données en k sous-ensembles et à évaluer le modèle k fois en utilisant chaque sous-ensemble comme ensemble de test une fois et les autres comme ensemble d'entraînement. Cela m'a permis d'obtenir une estimation plus fiable des performances du modèle en calculant la moyenne des mesures de performance sur chaque pli.

En répétant ce processus pour différentes valeurs de k (de k=1 à k=10), j'ai pu déterminer l'hyperparamètre optimal (k=7 dans ce cas) qui maximise les performances du modèle. Cette approche m'a également permis d'identifier les splits de données les plus appropriés pour l'entraînement et la validation du modèle.

J'ai obtenu une accuracy moyenne de 0.93 (figure 10) et une précision moyenne de 0.97 (figure 11) lors de la validation K-fold pour mon modèle KNN et j'ai représenté l'efficacité de ce modèle à l'aide d'un graphique ROC. (figure 12)

Bien que le classifieur KNN soit relativement simple à comprendre, il présente des inconvénients tels que sa sensibilité au bruit et sa lenteur en raison du calcul de toutes les distances pour chaque instance. Cependant, les résultats obtenus fournissent une base solide pour l'analyse et peuvent être utilisés comme point de départ pour des études plus approfondies.

3.2 Classifieur bayésien naïf

Les classifieurs bayésiens naïfs sont des modèles probabilistes utilisés pour estimer la probabilité d'appartenance à une classe donnée étant donné un vecteur de distribution. Ils reposent sur le théorème de Bayes pour calculer cette probabilité en se basant sur les distributions conditionnelles des caractéristiques (features) et la probabilité a priori des classes. Ces classifieurs probabilistes font l'hypothèse que les caractéristiques sont indépendantes

les unes des autres, ce qui simplifie le calcul des probabilités conditionnelles. Ils n'ont pas d'hyperparamètres à régler et estiment les paramètres des distributions en découpant le jeu de données en fonction des classes cibles, puis en calculant les moyennes et les écarts-types (ou d'autres paramètres selon la distribution choisie) pour chaque classe.

Dans ce projet, j'ai utilisé les estimateurs Gaussian Naive Bayes et Categorical Naive Bayes

pour traiter à la fois les variables quantitatives et catégorielles. Après avoir divisé l'ensemble de données en ensembles d'entraînement et de test, puis avoir effectué une validation croisée, j'ai obtenu une accuracy moyenne d'environ 0.68 pour les variables quantitatives et d'environ 0.89 pour les variables catégorielles. Sur l'ensemble du jeu de données, j'ai obtenu une accuracy moyenne de 0.82 et une précision moyenne de 0.82. (figure 13) J'ai représenté l'efficacité de ce modèle à l'aide d'un graphique ROC. (figure 14)

Les avantages des classifieurs bayésiens naïfs résident dans leur capacité à prendre en compte des connaissances externes pour estimer les distributions, ainsi que dans leur rapidité de calcul. Cependant, ils présentent des limites en raison de leurs suppositions de distribution et d'indépendance conditionnelle. Bien que cet algorithme soit rapide et efficace avec un grand nombre de données, il ne prend pas en compte les liens entre certaines caractéristiques et est connu pour être un mauvais estimateur de probabilité, ce qui limite son utilisation dans certaines situations.

3.3 Forêts aléatoires

Les forêts aléatoires sont des modèles d'apprentissage automatique qui combinent plusieurs arbres de décision pour améliorer la précision de la prédiction et réduire le surapprentissage. Au départ, plusieurs arbres de décision sont entraînés sur des sous-ensembles aléatoires du jeu de données d'entraînement. Chaque arbre est formé sur un échantillon de données tiré avec remplacement (bootstrap) et sur un sous-ensemble aléatoire des caractéristiques (features).
Les hyperparamètres à tester comprennent le nombre d'arbres de décision à inclure dans la forêt, la profondeur maximale des arbres pour éviter le surapprentissage, et le nombre minimum d'échantillons requis pour diviser un nœud en deux.

Chaque arbre de décision est construit en choisissant à chaque étape le meilleur split (avec le critère de Gini), parmi un sous-ensemble aléatoire des caractéristiques. Cela continue jusqu'à ce que les feuilles des arbres soient "pures" ou que la profondeur maximale de l'arbre soit atteinte.

Pour effectuer une prédiction, chaque arbre de la forêt donne une réponse basée sur les caractéristiques d'entrée. Dans le cas d'une classification, le résultat final est déterminé par un vote majoritaire parmi tous les arbres.

En combinant plusieurs arbres formés sur des sous-ensembles de données différents, les forêts aléatoires réduisent le surapprentissage par rapport à un seul arbre de décision et augmentent la précision. Cela les rend plus robustes et moins sensibles aux variations du jeu de données d'entraînement.

Les forêts aléatoires fournissent également une mesure de l'importance de chaque caractéristique dans la prédiction. Cette importance est calculée en mesurant la réduction moyenne

de l'impureté (par exemple, l'indice de Gini) que chaque caractéristique apporte lorsqu'elle est utilisée pour diviser les arbres de la forêt.

J'ai utilisé l'estimateur RandomForestClassifier et effectué une KFold validation. J'ai obtenu une accuracy moyenne de 0.96 (figure 15) sur l'ensemble des données avec 100 estimateurs (arbres). J'ai représenté l'efficacité de ce modèle à l'aide d'un graphique ROC. (figure 16)

Les forêts aléatoires sont robustes, peuvent gérer des données manquantes et des caractéristiques catégorielles sans prétraitement supplémentaire, et sont moins sensibles au surapprentissage par rapport aux arbres de décision simples.

Dans l'ensemble, les résultats montrent que le classifieur le plus intéressant pour le jeu de données en apprentissage supervisé semble être celui des forêts aléatoires.

3.4 SVM à noyau

La machine à vecteurs de support avec noyau (SVM, pour Support Vector Machine) appliquée à la classification (SVC, pour Support Vector Classifier) vise à séparer les données en classes en les représentant comme des vecteurs dans un espace de caractéristiques, puis à tracer une frontière de décision de manière à maximiser la distance entre les classes et la frontière.

Les hyperparamètres à tester comprennent le paramètre de régularisation C, le choix du noyau et le paramètre

Si les données ne sont pas linéairement séparables, le SVM peut utiliser une technique appelée "kernel trick" pour projeter les données dans un espace de dimension supérieure où elles peuvent être séparées linéairement.

On ne prend en compte que les valeurs situées près de la frontière entre les classes, appelées vecteurs de support. On mesure la distance jusqu'à chacun des vecteurs de support et on évalue leur importance dans l'apprentissage. Ensuite, on trace la frontière de décision. Pour effectuer une prédiction sur un nouveau point, on projette ces données dans l'espace de caractéristiques et on les classe en fonction du côté de la frontière où ils se trouvent.

J'ai obtenu une accuracy moyenne d'environ 0.95 (figure 17) et une précision moyenne de 0.95 (figure 18) sur l'ensemble de test pour une validation K-fold.

Le SVM est capable de trouver des frontières de décision complexes et non linéaires dans des espaces de grande dimension. Cette méthode est intéressante dans ce cas car elle permet d'atténuer le déséquilibre du jeu de données en pondérant différemment les erreurs de classification selon les classes. Ainsi, même si une classe est sous-représentée, le SVM peut toujours apprendre efficacement à distinguer entre les classes. De plus, le SVM est ro-

buste face au surapprentissage et peut généraliser rapidement et facilement à de nouveaux exemples une fois entraîné.

3.5 Réseau de neurones (MLP)

Le perceptron multicouche (MLP) est un type de réseau de neurones artificiels qui se compose de plusieurs couches de neurones, comprenant généralement une couche d'entrée, une ou plusieurs couches cachées et une couche de sortie. Chaque neurone dans le réseau est connecté à tous les neurones des couches adjacentes, formant un réseau dense.

Lorsque des données sont présentées au réseau, elles sont propagées à travers le réseau par des calculs pondérés et des transformations non linéaires effectuées par les neurones. Cette propagation avant (feedforward) aboutit à des prédictions ou des classifications basées sur les valeurs d'entrée. Pendant l'entraînement, les poids du réseau sont ajustés par rétropropagation du gradient, un algorithme d'optimisation, pour minimiser une fonction de perte. Cela permet au modèle d'apprendre à partir des données en adaptant les poids pour mieux correspondre aux étiquettes de sortie désirées.

Les performances et la capacité de généralisation du modèle sont influencées par différents hyperparamètres. Parmi eux, on trouve le nombre de couches cachées, le nombre de neurones par couche, la fonction d'activation utilisée dans chaque couche, ainsi que les paramètres d'optimisation tels que le taux d'apprentissage et le batch size. (figure 19)

Après avoir optimisé le réseau de neurones en sélectionnant les hyperparamètres appropriés, j'ai obtenu une accuracy de 0.93 (figure 20) et une précision de 0.99 (figure 21). Ces résultats soulignent l'efficacité du modèle MLP dans la classification des données, avec une précision élevée dans la prédiction des classes. J'ai représenté l'efficacité du réseau de neurones à l'aide d'un graphique représentant la courbe loss. (figure 22)

4 PCA

La réduction de dimension est une technique visant à réduire la complexité d'un ensemble de données en le projetant dans un espace de plus petite dimension, c'est-à-dire en réduisant le nombre de variables. Son objectif principal est d'accélérer le processus d'apprentissage des machines tout en luttant contre le risque d'overfitting, qui est souvent associé à un surplus de dimensions.

Le principe de la PCA (Analyse en Composantes Principales) consiste à projeter les données sur des axes, appelés composantes principales, tout en cherchant à minimiser la distance entre les points originaux et leurs projections. De cette manière, la PCA réduit la dimension du dataset tout en préservant autant que possible la variance des données, c'est-à-dire l'information qu'elles contiennent.

Dans cette approche, j'ai d'abord calculé le pourcentage de variance expliquée par chaque composante principale. Ensuite, j'ai tracé un graphique pour visualiser la relation entre la variance expliquée et le nombre de composantes. Mon objectif était de trouver un équilibre entre la réduction de dimension et la préservation de la variance. À cet effet, j'ai identifié sur le graphique (figure 23) un point optimal où un pourcentage de variance de 90% était atteint avec seulement 10 composantes principales.

Suite aux apprentissages avec KNN (figure 24) (figure 25) (figure 26), classifieur bayésien naïf (figure 27) (figure 28) et forêts aléatoires (figure 29) (figure 30), j'ai constaté que l'accuracy moyenne pour les trois modèles testés avec PCA était presque identique à celle obtenue sans réduction de dimension. Cependant, ce qui est remarquable, c'est la nette réduction du temps de calcul lors de l'utilisation de la PCA. En effet, le processus d'apprentissage supervisé s'est déroulé de manière beaucoup plus rapide avec la PCA.

Pour conclure, la réduction de dimension par PCA s'est avérée être une stratégie efficace pour améliorer les performances des modèles d'apprentissage tout en réduisant le temps de calcul.

5 Apprentissage non-supervisé

5.1 CAH

Pour mon approche en apprentissage non supervisé, j'ai choisi d'utiliser l'algorithme de Classification Ascendante Hiérarchique (CAH). Cette méthode consiste à regrouper des individus en clusters en se basant sur leurs similarités.
Je commence par échantillonner aléatoirement mon jeu de données pour travailler avec moins de valeurs et avoir une représentation graphique facilement compréhensible. Ensuite, je génère un dendrogramme (figure 31) qui illustre tous les individus regroupés en différents clusters, en montrant les étapes de regroupement. J'utilise ici le "complete link", une méthode qui cherche à regrouper les caractéristiques en clusters en se basant sur les valeurs de distance les plus éloignées entre les clusters car les tests avec le single link et l'average link n'étaient pas concluants.
En analysant le dendrogramme, je choisis un seuil de distance approprié. Dans ce cas, j'opte pour un seuil de 36, car cela nous permet de former 6 clusters distincts et permet une meilleure interprétabilité des clusters obtenus.
Les prochaines étapes consistent à analyser les clusters en étudiant la répartition des individus puis évaluer le clustering.

6 Résumé et perspectives

En comparant les performances des modèles de classification appliqués à l'ensemble de données, les résultats démontrent que le modèle de forêts aléatoires offre la meilleure pré-

Table 1 – Scores des différents classifieurs testés

Classifieur	Accuracy	Precision
KNN	0.93	0.97
Bayésien Naïf	0.82	0.82
Forêts aléatoires	**0.96**	
SVC	0.95	0.95
MLP	0.93	**0.99**
Avec PCA		
KNN	0.92	0.96
Bayésien Naïf	0.83	0.84
Forêts aléatoires	0.93	

cision globale, mesurée par l'accuracy tandis que le perceptron multicouche (MLP) produit la meilleure précision, en particulier dans la prédiction des classes positives, comme l'illustre le score de précision.

Lorsque je compare mes résultats de classification avec ceux disponibles sur Kaggle, les performances de mes modèles surpassent celles des autres. Cela peut-être dû à deux facteurs principaux : un prétraitement de données rigoureux et une meilleure optimisation des hyperparamètres pour chaque modèle utilisé.

Le prétraitement des données joue un rôle important dans les résultats de classification. J'ai investi du temps pour encoder et mettre à l'échelle les données, et sélectionné avec soins les features importantes.

De plus, j'ai accordé beaucoup d'attention à l'optimisation des hyperparamètres de chaque modèle. En les ajustant, j'ai maximisé les performances de chaque modèle de classification.

Ma prochaine étape consiste à tester de nouveaux modèles de classification, d'optimiser les hyperparamètres de mon modèle SVM à noyau afin d'obtenir de meilleurs résultats et de tester le SVM à noyau et les réseaux de neurones (MLP) en utilisant le jeu de données de dimension réduite.

Parallèlement, j'envisage d'explorer une autre technique de réduction de dimension en utilisant t-SNE.

Ensuite, je prévois de poursuivre mon exploration des techniques d'apprentissage non su-

pervisé en achevant l'étape de classification ascendante hiérarchique (CAH) et en testant deux autres algorithmes de clustering : le partitionnement en k-moyennes et DBSCAN.

7 Annexe

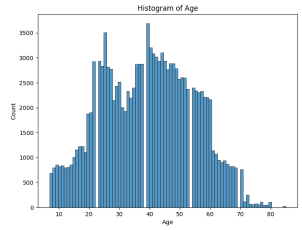
Table 2 – features du dataset airline passenger satisfaction triées par type

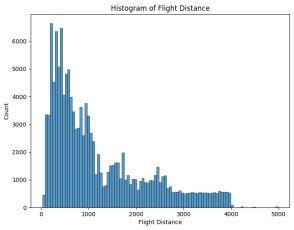
Quantitatives	Catégorielles	Ordinales
Age	Gender (Male/Female)	Inflight wifi service
Flight Distance	Customer Type (Loyal/Disloyal)	Departure/Arrival time convenient
Departure Delay in Minutes	Type of Travel (Personal/Business)	Ease of Online booking
Arrival Delay in Minutes	Class (Eco plus/Business/Eco)	Gate location
		Food and drink
		Online boarding
		Seat comfort
		Inflight entertainment
		On-board service
		Leg room service
		Baggage handling
		Checkin service
		Inflight service
		Cleanliness

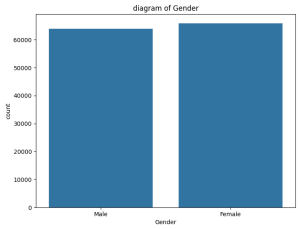
Gender_Male	Customer Type_Loyal Customer	Customer Type_disloyal Customer	Type of Travel_Business travel	Type of Travel_Personal Travel	Class_Business	Class_Eco	Class_Eco Plus	satisfaction_neutral or dissatisfied
True	True	False	False	True	False	False	True	True
True	False	True	True	False	True	False	False	True

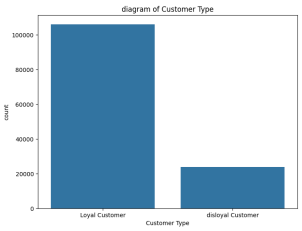
Figure 1 – variables catégorielles une fois encodée avec le one-hot encode

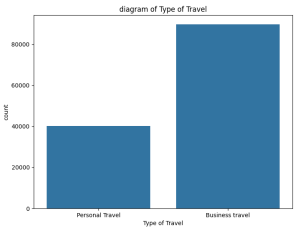
Table 3 – répartition des individus par features dans le dataset

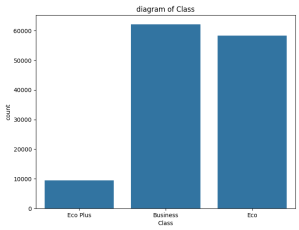


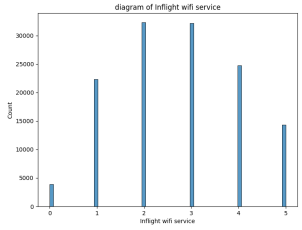


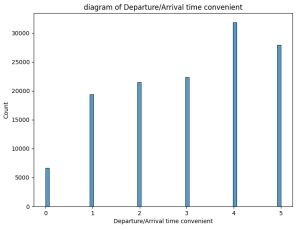


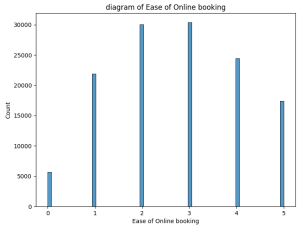


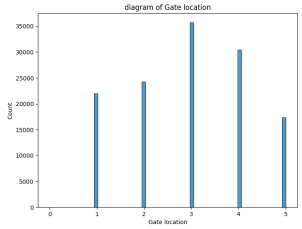


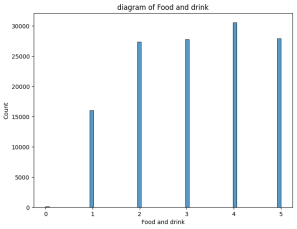


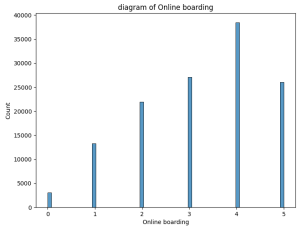


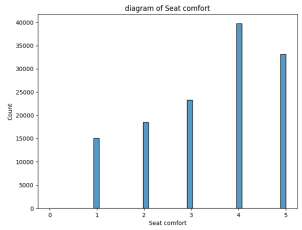


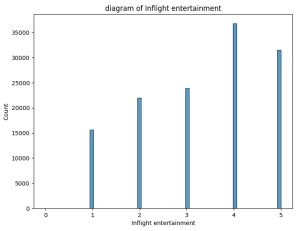


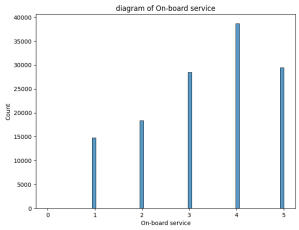


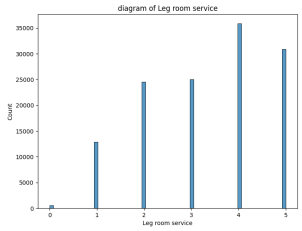


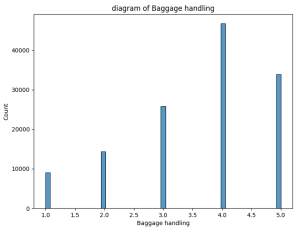


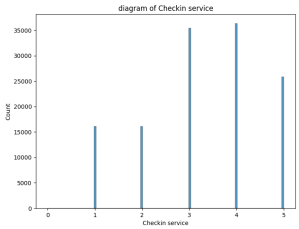


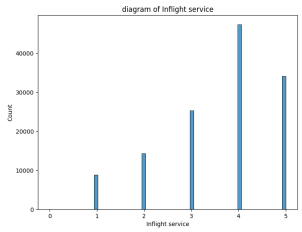


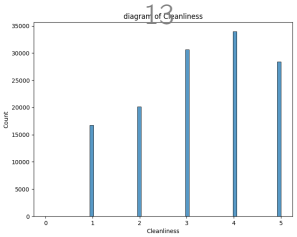


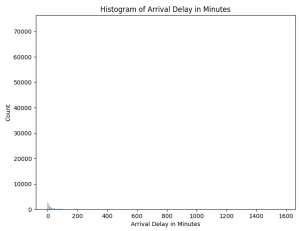












```
===== Encoded classes for Inflight wifi service: [array([0, 1, 2, 3, 4, 5])]
===== Encoded classes for Departure/Arrival time convenient: [array([0, 1, 2, 3, 4, 5])]
===== Encoded classes for Ease of Online booking: [array([0, 1, 2, 3, 4, 5])]
===== Encoded classes for Gate location: [array([0, 1, 2, 3, 4, 5])]
===== Encoded classes for Food and drink: [array([0, 1, 2, 3, 4, 5])]
===== Encoded classes for Online boarding: [array([0, 1, 2, 3, 4, 5])]
===== Encoded classes for Seat comfort: [array([0, 1, 2, 3, 4, 5])]
```

Figure 2 – variables ordinales une fois encodée avec l'ordinal encoder

	id	Age	Flight Distance
count	129880.000000	129880.000000	129880.000000
mean	64940.500000	39.427957	1190.316392
std	37493.270818	15.119360	997.452477
min	1.000000	7.000000	31.000000
25%	32470.750000	27.000000	414.000000
50%	64940.500000	40.000000	844.000000
75%	97410.250000	51.000000	1744.000000
max	129880.000000	85.000000	4983.000000

Figure 3 – description des statistiques univariées pour l'âge et la distance de vol

	Age	Flight Distance	Departure Delay in Minutes	Arrival Delay in Minutes
0	0.076923	0.086632	0.015704	0.011364

Figure 4 – variables quantitatives une fois misent à l'échelle

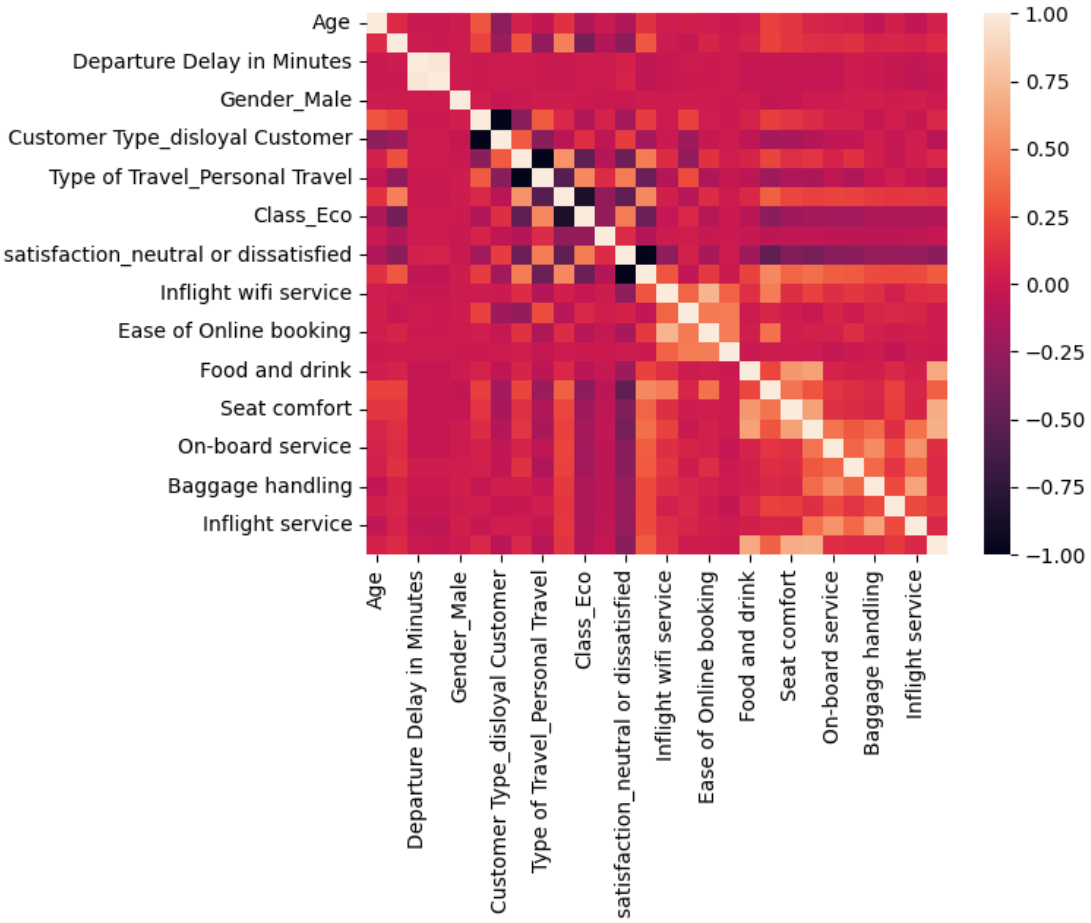


Figure 5 – heatmap des corrélations entre les features

Feature 1	Feature 2	Correlation
Departure Delay in Minutes	Arrival Delay in Minutes	0.965506
Customer Type_Loyal Customer	Customer Type_disloyal Customer	-1.000000
Type of Travel_Business travel	Type of Travel_Personal Travel	-1.000000
Class_Business	Class_Eco	-0.864761
satisfaction_neutral or dissatisfied	satisfaction_satisfied	-1.000000

Figure 6 – paires de features corrélées à |r|>0.8

```
print(nan_counts)
```

```
Age                                0
Flight Distance                    0
Departure Delay in Minutes         0
Arrival Delay in Minutes         393
```

Figure 7 – nombres de valeurs manquantes pour les 4 premières features

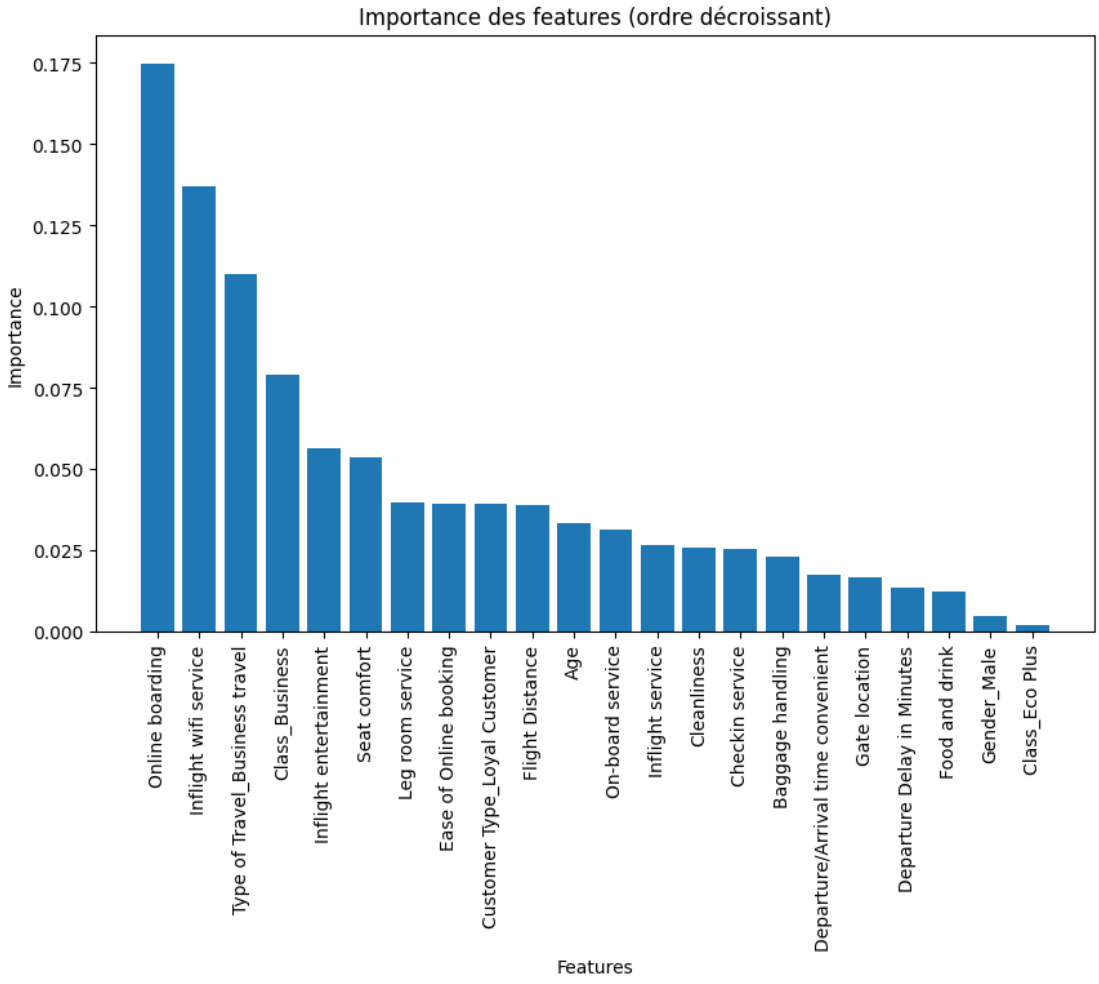


Figure 8 – Importance de chaque feature calculée par le classifieur forêts aléatoires

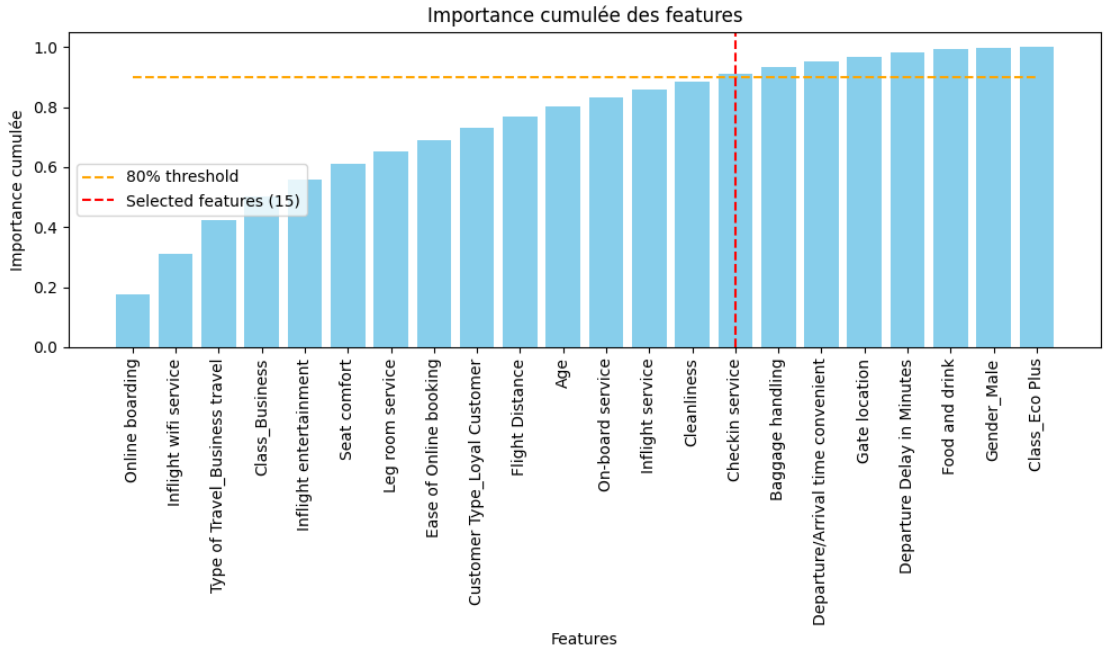


Figure 9 – Importances cumulées des features selon le classifieur forêts aléatoires avec un seuil à 80%

```
K-fold validation score [0.93655682 0.92931937 0.92931937 0.93193717 0.92931937 0.92546966
 0.92793348 0.9316292  0.93701879 0.93224515 0.93393902 0.92854943
 0.92731752 0.93085925 0.93147521 0.92331383 0.93101324 0.9316292
 0.92947336 0.93671081]
Average k-fold validation score : 0.9307514628888207
Average error on fold: 0.0029198857082782894
Standard error on fold: 0.0029198857082782894
```

Figure 10 – Accuracy moyenne et erreur moyenne / standard pour une validation Kfold sur le modèle KNN

```
K-fold validation score [0.97488862 0.96681688 0.97086149 0.97843552 0.97191953 0.96887967
 0.96697921 0.97412731 0.96654427 0.96917386 0.97058824 0.97405272
 0.97092547 0.96876302 0.96968472 0.97399431 0.97055111 0.96586345
 0.97174907 0.97054076]
Average k-fold validation score : 0.9707669605708272
Average error on fold: 0.0029198857082782894
Standard error on fold: 0.0029198857082782894
```

Figure 11 – Précision moyenne pour une validation Kfold sur le modèle KNN

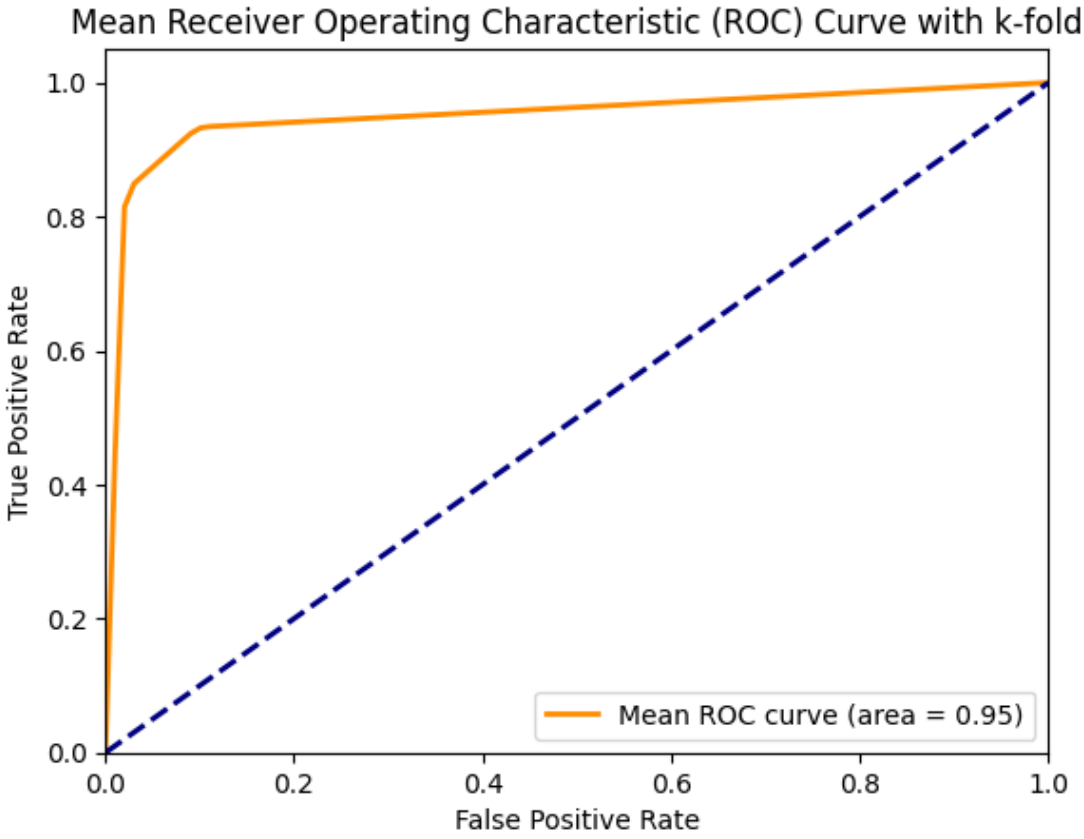


Figure 12 – Graphique ROC pour KNN avec PCA évalué par Kfold

```
===== Accuracy using full dataset
 0.8247998152140438
===== Precision using full dataset
 0.8155798391782721
```

Figure 13 – Accuracy et precision pour le modèle de classification bayésien naïf

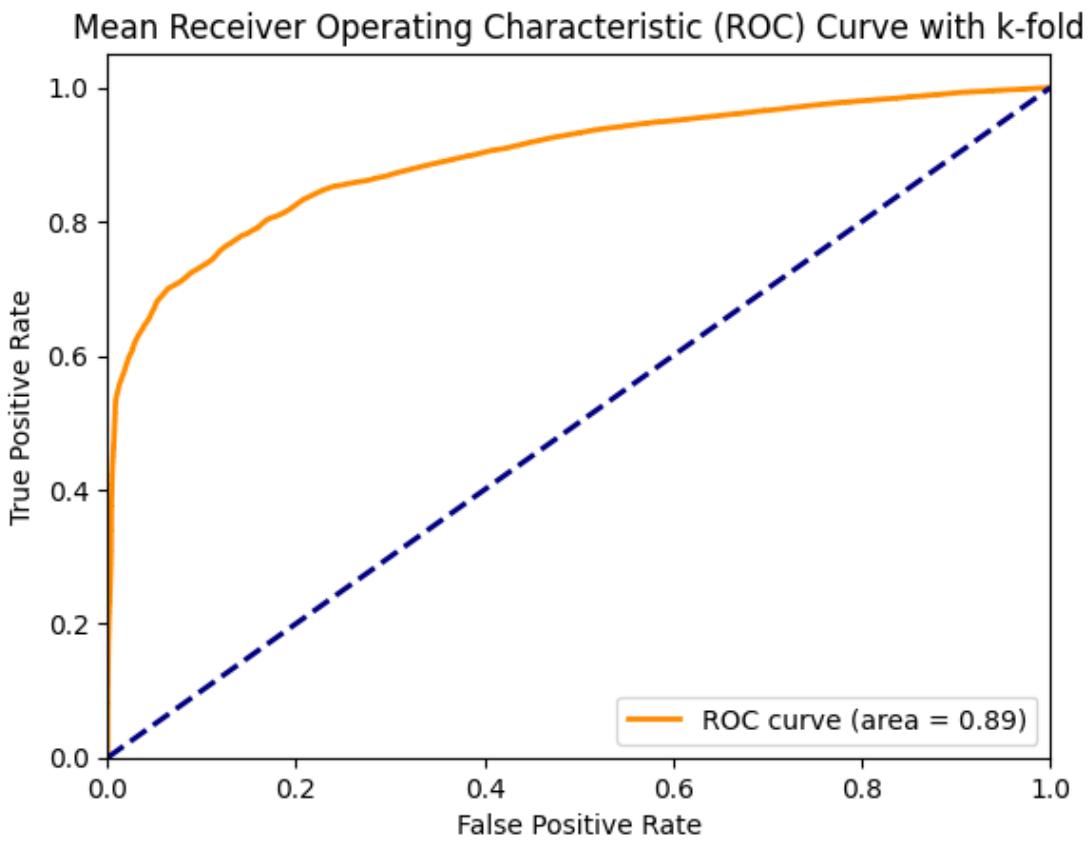


Figure 14 – Graphique ROC pour classifieur bayésien naïf avec PCA évalué par Kfold

```
K-fold validation score [0.96134894 0.9644287  0.96535263 0.96273483 0.96550662 0.96027102
 0.96427472 0.95965507 0.96612257 0.96196489 0.96335079 0.96427472
 0.9644287  0.96612257 0.96766246 0.96412073 0.96304281 0.96227287
 0.9658146  0.9618109 ]
Average k-fold validation score : 0.9637280566676931
Average error on fold: 0.002243306138876943
Standard error on fold: 0.002243306138876943
```

Figure 15 – Accuracy et erreur moyenne pour une validation Kfold sur le modèle de forêts aléatoires

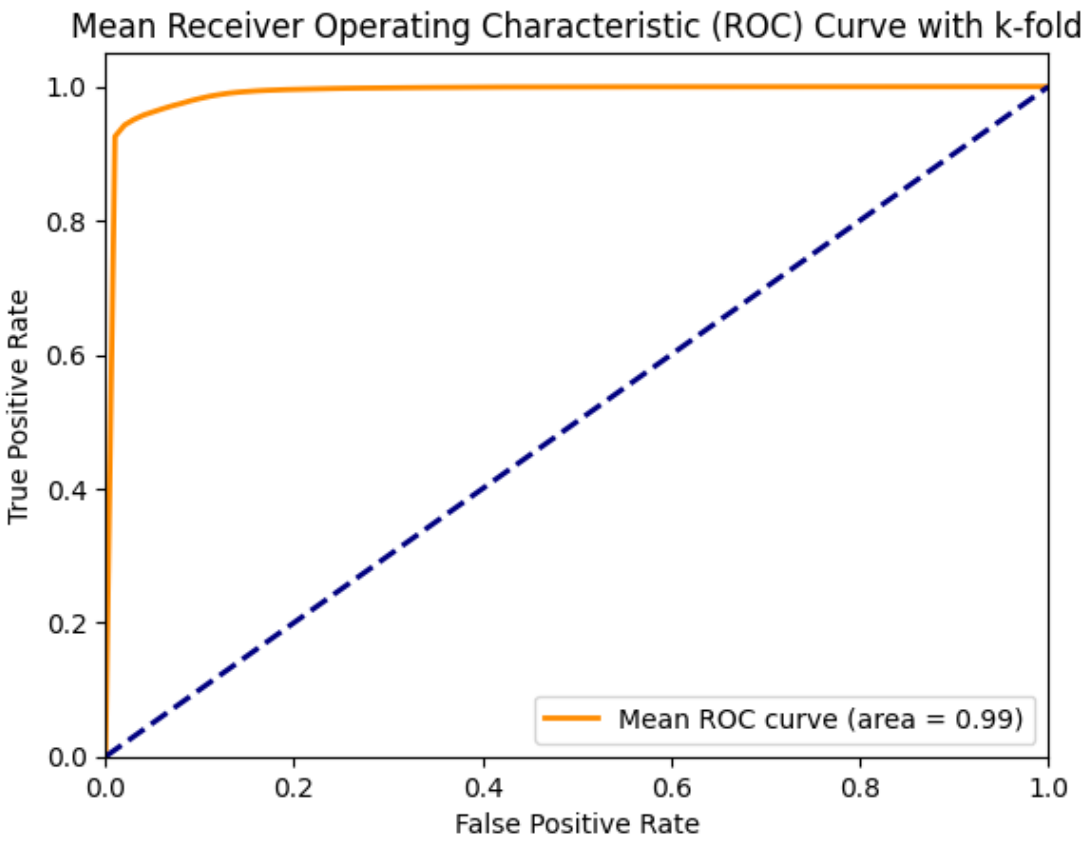


Figure 16 – Graphique ROC pour forêts aléatoires avec PCA évaluées par Kfold

```
K-fold validation score [0.95595935 0.95041577 0.95688328 0.94687404 0.94979982 0.95241762
 0.95411149 0.94949184 0.95210964 0.95180166 0.95056976 0.95118571
 0.95087773 0.95041577 0.95796119 0.95210964 0.95241762 0.95457345
 0.94764398 0.95287958]
Average k-fold validation score : 0.9520249461040962
Average error on fold: 0.002371857198506534
Standard error on fold: 0.002371857198506534
```

Figure 17 – Accuracy et erreur moyenne pour une validation Kfold sur le modèle SVC

```
K-fold validation score [0.94814815 0.95412844 0.96129974 0.95035198 0.95840739 0.95182013
 0.95405599 0.95947426 0.9609459  0.95600559 0.95250181 0.95602837
 0.95853037 0.94874514 0.94940916 0.94177584 0.94850852 0.95902229
 0.94898336 0.95920617]
Average k-fold validation score : 0.9538674307582526
Average error on fold: 0.0032205417714994008
Standard error on fold: 0.0032205417714994008
```

Figure 18 – Precision pour une validation Kfold sur le modèle SVC

```
alpha_values = [0.0001, 0.001, 0.01, 0.1, 1]
iter_nb = [1000, 1250, 1500,1750,2000]
batch_size_values = [8,16,32,64,128,256]
learning_rate_init_values = [0.001, 0.005, 0.01, 0.05, 1]
hidden_layer_sizes_values = [25,50,100,150,200,500]
```

Figure 19 – Hyperparamètres testés pour optimiser le MLP

```
Best accuracy : 0.925
 Best alpha : 0.01
 Best number of iter : 1250
 Best batch_size : 16
 Best learning rate init : 0.01
 Best hidden layer sizes : 150
```

Figure 20 – Hyperparamètres optimaux pour obtenir la meilleure accuracy pour mon MLP

```
========= Precision=0.9861111111111112
 Alpha=0.01
 max_iter=1500
 batch_size=32
 learning_rate_init=0.01
 hidden_layer_sizes=200
```

Figure 21 – Hyperparamètres optimaux pour obtenir la meilleure précision pour mon MLP

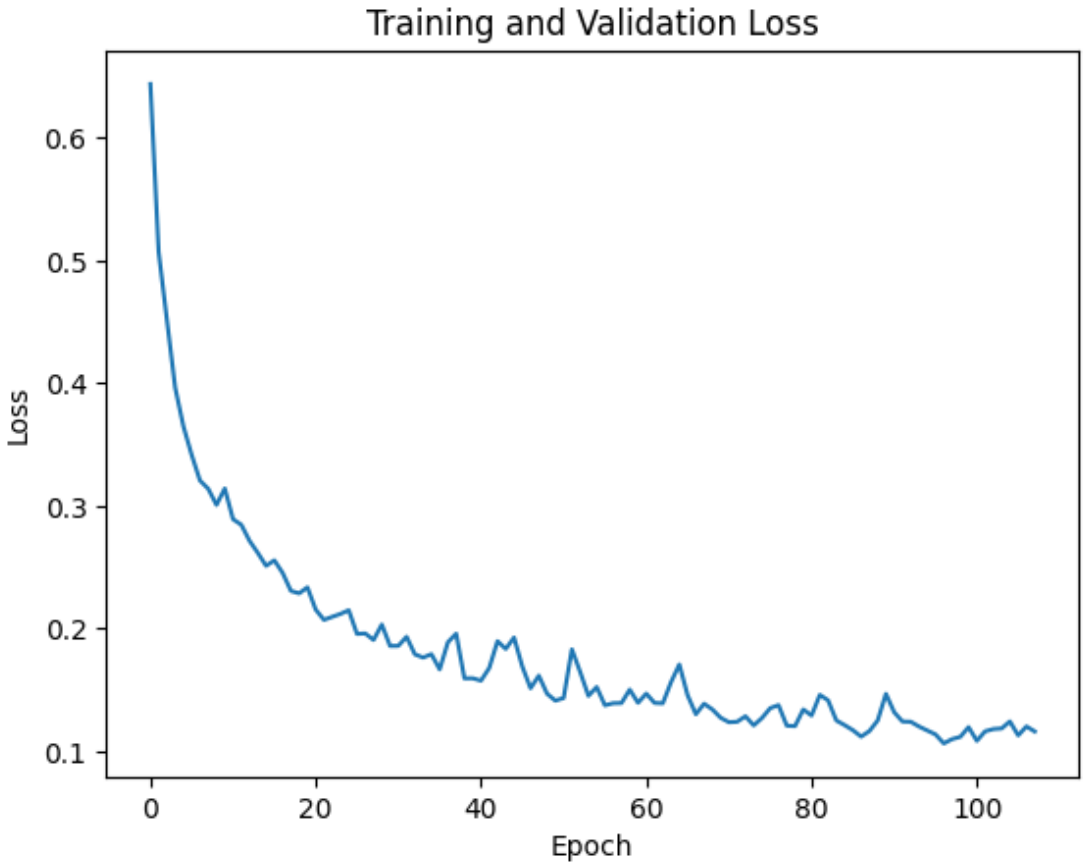


Figure 22 – Courbe de perte lors de l'entraînement et du test du MLP

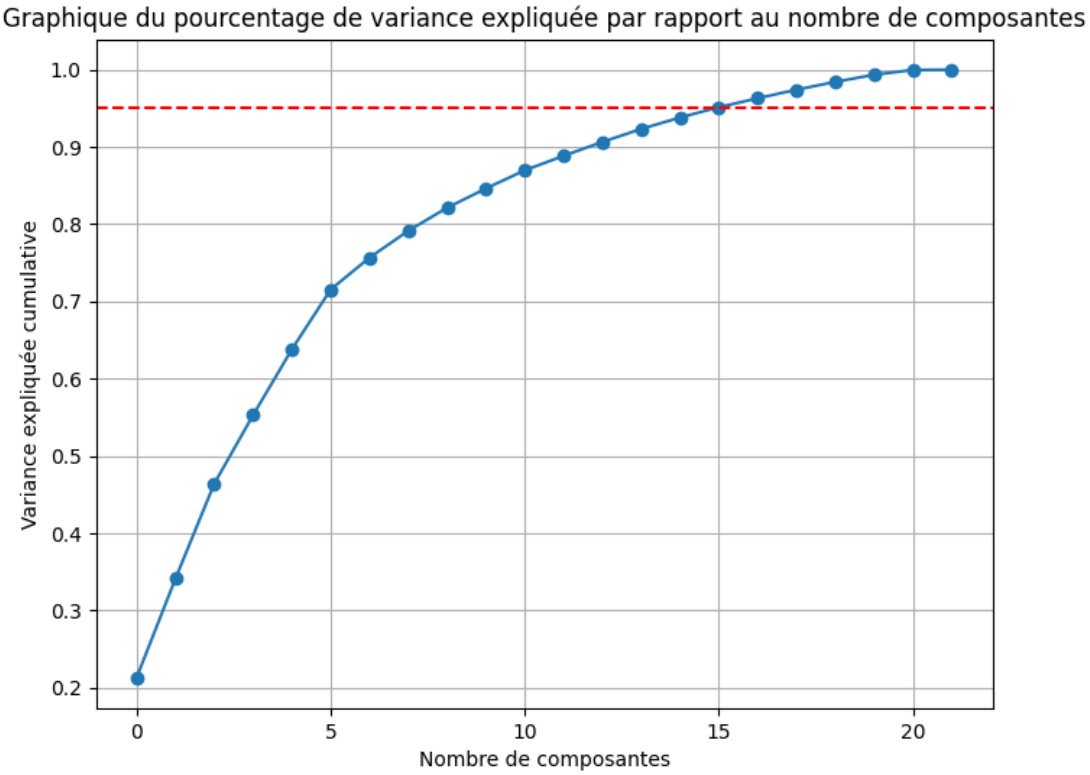


Figure 23 – Diagramme du coude pour la PCA avec un seuil 0.95

```
K-fold validation score [0.92161996 0.9171543  0.92254389 0.91684632 0.92285186 0.92315984
 0.92408377 0.92577764 0.92593163 0.92331383 0.92254389 0.92485371
 0.92346782 0.92146597 0.92531568 0.91915614 0.92346782 0.92131198
 0.92993532 0.92115799]
Average k-fold validation score : 0.922797967354481
Average error on fold: 0.0035836289319042917
Standard error on fold: 0.0035836289319042917
```

Figure 24 – Accuracy moyenne et erreur moyenne / standard pour une validation Kfold sur le modèle KNN avec PCA

```
K-fold validation score [0.96351575 0.95668317 0.9589852  0.96101203 0.95222405 0.96109937
 0.95791667 0.96052079 0.95967078 0.96427105 0.9545644  0.95159338
 0.96822121 0.95628866 0.95639654 0.95565686 0.96689824 0.96163575
 0.96346073 0.96213992]
Average k-fold validation score : 0.9596377280079661
Average error on fold: 0.0035836289319042917
Standard error on fold: 0.0035836289319042917
```

Figure 25 – Précision moyenne pour une validation Kfold sur le modèle KNN avec PCA

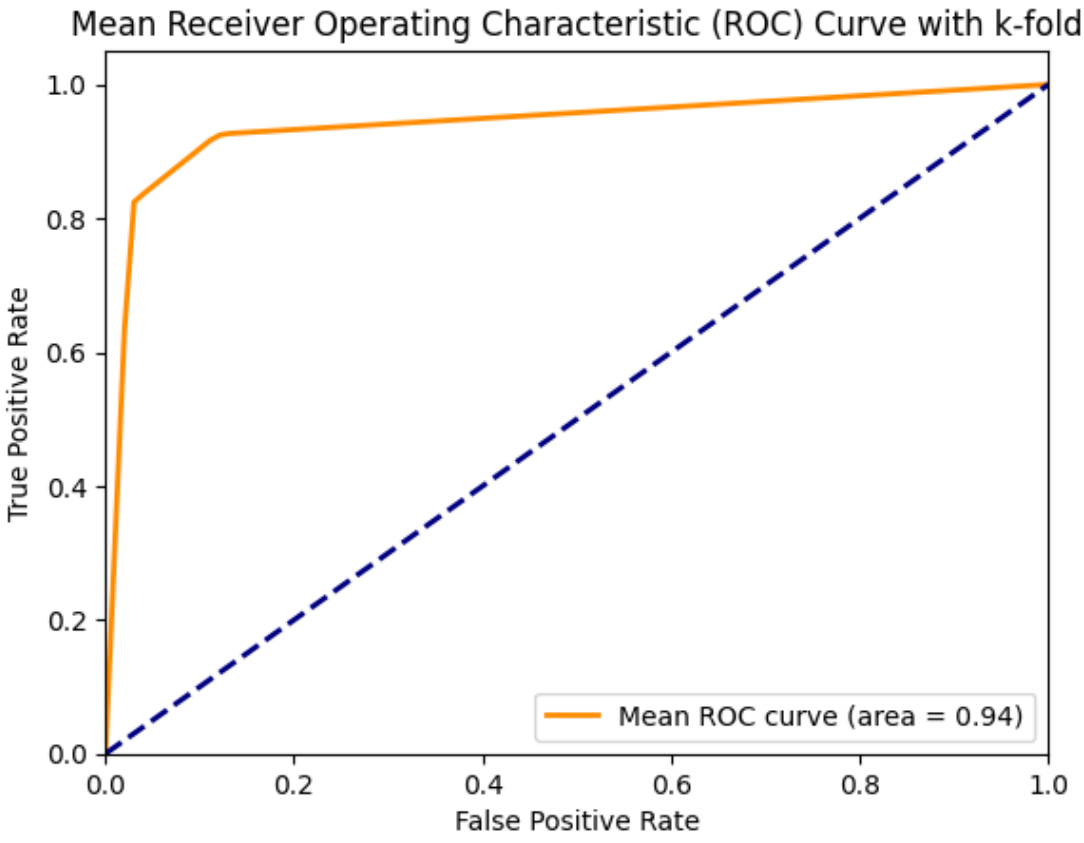


Figure 26 – Graphique ROC pour KNN avec PCA évalué par Kfold

```
===== Accuracy using full dataset
 0.8299738219895288
===== Precision using full dataset
 0.8367619084972447
```

Figure 27 – Accuracy et precision pour le modèle de classification bayésien naïf

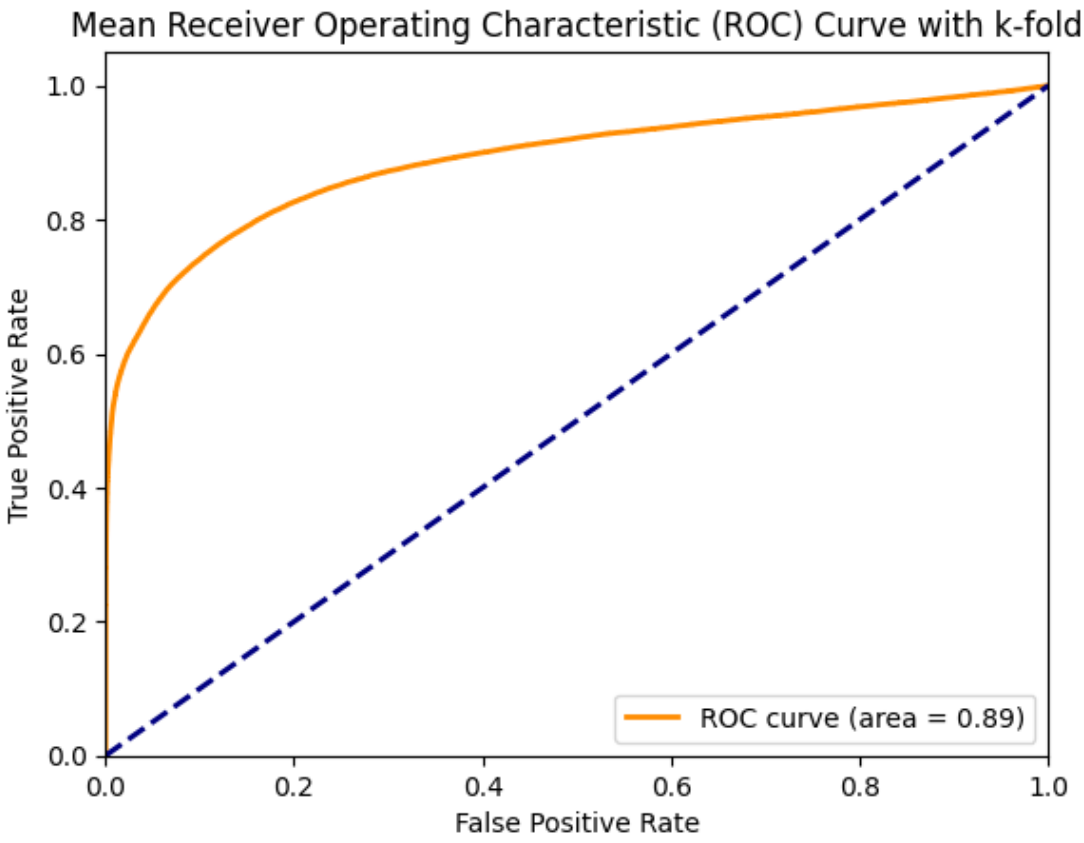


Figure 28 – Graphique ROC pour classifieur bayésien naïf avec PCA évalué par Kfold

```
Accuracy on training set: 1.000
Accuracy on test set: 0.929
```

Figure 29 – Accuracy et erreur moyenne pour une validation Kfold sur le modèle de forêts aléatoires avec PCA

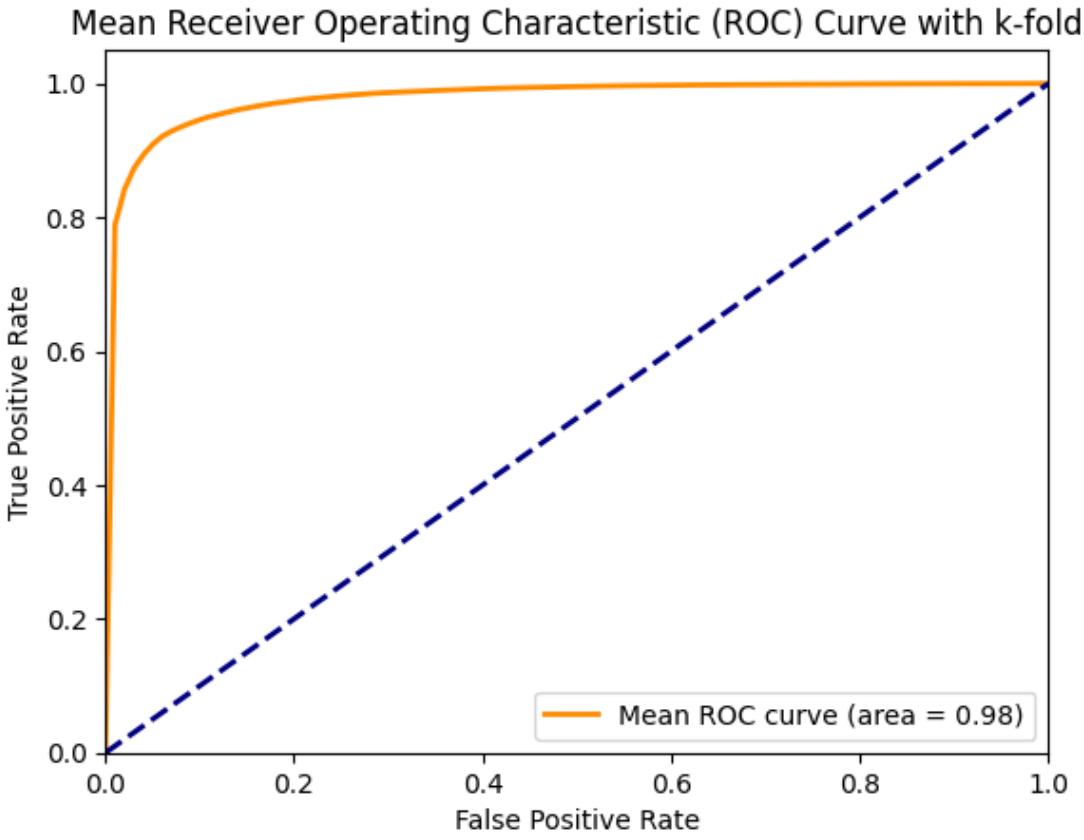


Figure 30 – Graphique ROC pour forêts aléatoires avec PCA évaluées par Kfold

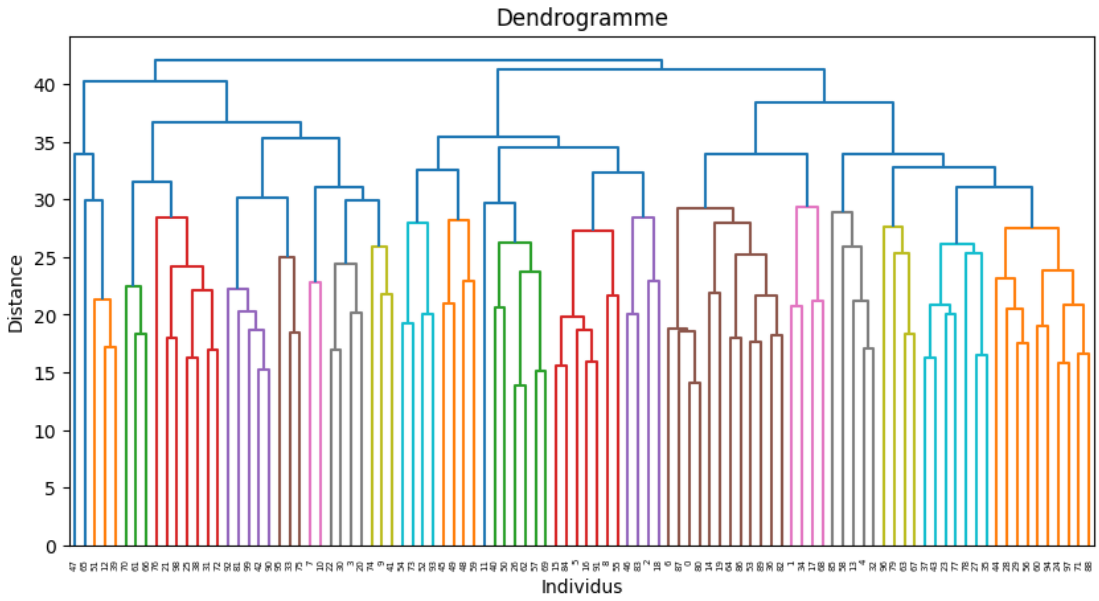


Figure 31 – Dendrogramme du CAH

Table des figures

Liste des tableaux