Université Ibn Zohr
Faculté Polydisciplinaire de Ouarzazate
Filière master : Intelligence artificielle et applications

master IAA-2025

Naive Bayésienne Classification

Projet tutoré présenté par :
HANANE IQLI
HANAN BASSOU
MARYAME KHOUYA
ABDELHAQ AJGUERNOUN

Sous la direction de :

Prof. **KAMAL OMARI**
Prof. **MOHAMED LARAQUI**
Prof. **AISSAM HADRI**

Année universitaire : 2024-2025

Table des matières

Table des figures

Remerciement

Nous tenons tout d'abord à remercier Dieu le tout puissant et miséricordieux qui nous a donné la force et la patience d'accomplir ce modeste travail. En second lieu, nous tenons à remercier notre encadrons Prof. KAMAL OMARI , Prof. MOHAMED LARAQUI, Prof. AISSAM HADRI , pour ses précieux conseils et son aide durant toute la période d'élaboration de ce mémoire. Enfin, ça nous fait plaisir de délivrer un bouquet de remerciement pour toutes les personnes qui ont participé de près ou de loin à la réalisation et à la réussite de ce travail.

Introduction

L'apprentissage automatique repose sur plusieurs algorithmes statistiques et probabilistes permettant de modéliser des données et de faire des prédictions. Parmi eux, l'algorithme Naïve Bayes occupe une place importante en raison de sa simplicité et de son efficacité, notamment dans les tâches de classification. Il est basé sur le théorème de Bayes, un concept fondamental en probabilités. Ce document explore d'abord les bases du théorème de Bayes, avant d'approfondir l'algorithme Naïve Bayes et ses applications.

Chapitre 1

Naïve Bayes

1.1 Histoire de bayes

Thomas Bayes

Thomas Bayes (1702-1761) était un mathématicien et théologien britannique. Il est principalement connu pour avoir formulé un théorème fondamental en probabilité conditionnelle, qui porte aujourd'hui son nom. Ce théorème, formalisé après sa mort par Richard Price, est devenu un pilier des statistiques bayésiennes et trouve de nombreuses applications dans l'apprentissage automatique, la reconnaissance de formes et la prise de décision sous incertitude.

1.2 Théorème de Bayes

Le théorème de Bayes est une formule mathématique permettant de calculer une probabilité conditionnelle, c'est-à-dire la probabilité qu'un événement se produise en fonction d'une information préalable. Il s'exprime de la manière suivante :

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Dans le cas de la classification, cela devient :

$$P(C|X) = \frac{P(X|C)P(C)}{P(X)}$$

Où :

- $\mathbf{P(C|X)}$: C'est la probabilité que l'événement C se produise sachant que X s'est déjà produit. C'est ce qu'on cherche à calculer.
- $\mathbf{P(X|C)}$: C'est la probabilité que l'événement X se produise sachant que C est vrai. On l'appelle aussi la **vraisemblance**.
- $\mathbf{P(C)}$: C'est la **probabilité a priori** de C, c'est-à-dire la probabilité que C se produise avant d'observer X.
- $\mathbf{P(X)}$: C'est la **probabilité totale** de X, qui agit comme un facteur de normalisation. Elle peut être calculée comme :

$$P(X) = \sum_{C} P(X|C)P(C)$$

si C peut prendre plusieurs valeurs.

1.3 Hypothèse d'indépendance conditionnelle

Le modèle Naïve Bayes suppose que les caractéristiques $X = (x_1, x_2, \ldots, x_n)$ sont indépendantes les unes des autres conditionnellement à la classe C :

$$P(X|C) = P(x_1|C)P(x_2|C)\ldots P(x_n|C)$$

En injectant cette équation dans le théorème de Bayes :

$$P(C|X) = \frac{P(x_1|C)P(x_2|C)\ldots P(x_n|C)P(C)}{P(X)}$$

Puisque $P(X)$ est constant pour toutes les classes, la classification se fait en maximisant :

$$P(C|X) \propto P(C)\prod_{i=1}^{n} P(x_i|C)$$

On choisit la classe C qui maximise cette expression.

1.4 Cas des distributions spécifiques

La façon dont on calcule $P(x_i|C)$ dépend du type de données :

— **Naïve Bayes Gaussien (données continues) :** utilisé lorsque les variables indépendantes suivent une distribution normale.

— **Naïve Bayes Multinomial (texte, données discrètes) :** principalement utilisé pour la classification de textes où les caractéristiques sont des fréquences de mots.

— **Naïve Bayes Bernoulli :** adapté aux données binaires, souvent utilisé pour le filtrage de spam.

Chapitre 2

Types de Naïve Bayes

2.1 Gaussien

2.1.1 Définition

Le Naive Bayes Gaussien est un algorithme d'apprentissage automatique populaire, connu pour sa simplicité et son efficacité dans les tâches de classification. Parmi ses variantes, le Naive Bayes Gaussien est particulièrement utile pour les données continues. Il fait partie de la famille des classificateurs Naive Bayes et suppose que les caractéristiques suivent une distribution gaussienne. Cette hypothèse simplifie le calcul tout en produisant des résultats fiables.
Le Naive Bayes Gaussien est basé sur le théorème de Bayes, qui décrit la probabilité d'un événement en fonction de la connaissance préalable des conditions qui pourraient être liées à cet événement. Dans le contexte de l'apprentissage automatique, il s'agit d'un classificateur probabiliste qui suppose que les caractéristiques suivent une distribution gaussienne (normale). La partie "naïve" provient de l'hypothèse d'indépendance entre les caractéristiques, ce qui simplifie le calcul des probabilités.
Les mathématiques derrière gaussian

Le classificateur Naive Bayes repose sur l'hypothèse d'indépendance conditionnelle, ce qui signifie que, pour une classe donnée y, les caractéristiques (x1,x2,...,xn)sont supposées être indépendantes les unes des autres. Cela implique que la probabilité conditionnelle conjointe des caractéristiques données la classe se décompose en un produit de probabilités conditionnelles individuelles :

$$P(x_1, x_2, \ldots, x_n \mid y) = P(x_1 \mid y) \cdot P(x_2 \mid y) \cdot \cdots \cdot P(x_n \mid y)$$

Donc

$$P(x \mid y) = \prod_{i=1}^{n} P(x_i \mid y)$$

2.1.2 Modèle Gaussian Naive Bayes :

Dans le cas de Gaussian Naive Bayes, on suppose que chaque caractéristique xi, donnée la classe y, suit une distribution normale (gaussienne) :

$$P(x_i \mid y) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Où :

- x_i est la valeur de la caractéristique.
- μ est la moyenne des valeurs de la caractéristique pour une classe donnée y.
- σ est l'écart-type des valeurs de la caractéristique pour cette classe.
- π est une constante (environ 3,14159).
- e est la base du logarithme naturel.

Telque

$$\mu = E[X] = \frac{1}{N}\sum_{i=1}^{N} X_i$$

La variance 2 est donnée par :

$$\sigma^2 = V[X] = \frac{1}{N}\sum_{i=1}^{N}(X_i - \mu)^2$$

Application au calcul de la probabilité a posteriori :
La probabilité a posteriori $P(y|x)$ est ce que l'on cherche à maximiser lors de la classification, c'est-à-dire, étant donné un vecteur de caractéristiques $x = (x_1, x_2, \ldots, x_n)$, quelle est la classe y la plus probable.

D'après le Théorème de Bayes, on peut écrire la probabilité a posteriori comme :

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)}$$

Où :

- $P(y|x)$ est la probabilité a posteriori, c'est-à-dire la probabilité que la classe y soit vraie étant donné les observations x.
- $P(x|y)$ est la vraisemblance, c'est-à-dire la probabilité d'observer x étant donné la classe y.
- $P(y)$ est la probabilité a priori de la classe y.
- $P(x)$ est la probabilité marginale des caractéristiques x, telle que :

$$P(x) = \sum_{y} P(x|y)P(y)$$

- $P(x|y)$ est la vraisemblance, c'est-à-dire la probabilité d'observer les caractéristiques x sous la classe y.
- $P(y)$ est la probabilité a priori de la classe y, c'est-à-dire la probabilité d'appartenir à la classe y avant d'observer x.

Avec l'indépendance conditionnelle et l'hypothèse gaussienne, la probabilité a posteriori P(y/x) peut être réécrite de manière simplifiée :

$$P(y|x) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x)}$$

Remplaçons $P(x_i \mid y) dans la dernière formule$:

$$P(y|x) = \frac{P(y)\prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(x_i-\mu_i)^2}{2\sigma_i^2}\right)}{P(x)}$$

la formule de prédiction :

$$Y_{\text{pred}} = \arg\max_{y \in Y} \frac{P(x_1, x_2, \ldots, x_n \mid y)P(y)}{P(x_1, x_2, \ \ldots, \ x_n)}$$

Comprendre les Mathématiques Derrière le Naive Bayes Gaussien :

Pour comprendre le fonctionnement du Naive Bayes Gaussien, il est nécessaire d'examiner les mathématiques sous-jacentes.

- **Fonction de densité de probabilité** : Cette fonction est utilisée pour calculer la vraisemblance d'une caractéristique donnée une classe.
- **Probabilité conditionnelle** : L'algorithme calcule la probabilité de chaque classe donnée les caractéristiques d'entrée.
- **Théorème de Bayes dans Naive Bayes Gaussien** : Le théorème de Bayes est essentiel pour calculer la probabilité a posteriori d'une classe donnée les caractéristiques d'entrée.

2.2 Bernoulli

2.2.1 Définition

Bernoulli Naïve Bayes est une sous-catégorie de l'algorithme naïf bayésien. Il est généralement utilisé lorsque les données sont binaires et il modélise l'occurrence des caractéristiques à l'aide de la distribution de Bernoulli. Il est utilisé pour la classification de caractéristiques binaires telles que 'Oui' ou 'Non', '1' ou '0', 'Vrai' ou 'Faux', etc. Ici, il est à noter que les caractéristiques sont indépendantes les unes des autres.
Les mathématiques derrière Bernoulli
Naïve Bayes : Le cœur de Bernoulli Naïve Bayes est basé sur le théorème de Bayes qui aide à calculer la probabilité conditionnelle d'une classe donnée x = (x, x, ..., x). Maintenant, dans le modèle naïf de Bernoulli, nous supposons que chaque caractéristique est conditionnellement indépendante compte tenu de la classe y. Cela signifie que nous pouvons calculer la probabilité que

chaque caractéristique se produise comme suit :

$$p(x_i \mid y) = p(i \mid y)^{x_i} \cdot (1 - p(i \mid y))^{(1-x_i)}$$

p(xi|y) : est la probabilité conditionnelle que xi se produise à condition que y se soit produit

i : est l'événement

xi : contient la valeur binaire 0 ou 1.

Distribution de bernoulli :

$$P(x) = P[X = x] = \begin{cases} q = 1 - p & \text{si } x = 0 \\ p & \text{si } x = 1 \end{cases}$$

La distribution de Bernoulli est utilisée pour le calcul des probabilités discrètes. Il calcule soit le succès, soit l'échec. Ici, la variable aléatoire est soit 1, soit 0, dont la probabilité d'apparition est notée respectivement par p ou (1-p).

2.2.2 Propriétés de la distribution de Bernoulli

Espérance (moyenne)

L'espérance d'une variable aléatoire X suivant une distribution de Bernoulli est donnée par :

$$\mathbb{E}[X] = p$$

La valeur moyenne attendue est p.

Variance

La variance d'une variable aléatoire X suivant une distribution de Bernoulli est donnée par :

$$\mathrm{Var}(X) = p(1 - p)$$

La variance mesure la dispersion des valeurs autour de la moyenne.

Écart-type

L'écart-type d'une variable aléatoire X suivant une distribution de Bernoulli est donné par :

$$\sigma = \sqrt{p(1 - p)}$$

2.3 Implémentation

2.3.1 Phase d'entrainement.

Nous avons divisé le jeu de données en deux phases : 90 % pour la phase d'entrainement et 10 % pour la phase de teste.

2.3.2 Phase d'évaluation.

2.3.2.1 Métrique de performance :

	positive	**négative**
positive	Vrai positif (VP)	positif (FP)
négative	Faux négatif (FN)	Vrai négatif (VN)

les mesure de performance sont des indicateurs de la correspondance entre Valeurs prédites et valeurs obtenus à partire du modèle obtenu via la matrice de confusion qui vont nous permettre de juger de la qualité de nos prédictions **Precision** , **Rappel** , **F1-score** et **Accuracy** :

Precision : C'est la proportion des individues qui sont correctement identifiées par le modéle . l'équation de précision est représentée comme suit :

$$\text{Précision} = \frac{VP}{VP+FP}$$

Rappel : C'est lapetite proportion des individus par rapport à la quantité globale des individus applicables . l'equoition de rappel est représentée comme suit :

$$\text{Rappel} = \frac{VP}{VP+FN}$$

F1-score : C'est la moyanne entre la précision et le rappele :

$$\text{F1-score} = 2 * \frac{Prcision*Rappel}{Prcision+Rappel}$$

Accuracy : L'exactitude se réfère à la proximité d'une mesure ou d'une valeur à la valeur réelle ou attendue. Il s'agit de la distance entre la valeur mesurée et la valeur réelle. Plus la valeur mesurée est proche de la valeur réelle, plus l'exactitude est élevée.

$$\text{Accuracy} = \frac{VP+VN}{VP+VN+FP+FN}$$

2.4 Multinomial

2.4.1 Définition

Le classificateur Naïve Bayes Multinomial (MNB) est une extension de l'algorithme Naïve Bayes conçue pour le traitement de données discrètes, aussi utilisé pour la classification de documents où les données sont des fréquences de mots. Il suppose que les caractéristiques suivent une distribution multinomiale, ce qui est approprié pour les tâches de classification de texte où les caractéristiques sont des occurrences de mots.

Elle basé sur le **théorème de bayes** de sorte qu'elle utilise les probabilités pour prédire la catégorie d'un document texte en fonction des fréquences des mots (la probabilité qu'un document appartienne à une catégorie spécifique).

La distribution multinomiale est un concept clé de la méthode Bayes naïve multinomiale, notamment pour les tâches de classification de textes. Elle permet d'estimer la probabilité qu'un document appartienne à une classe en fonction de la distribution du nombre de mots ou de la fréquence des termes. De sort que :

• Chaque mot est traité comme un résultat.
• Le nombre de mots dans un document représente les données observées.
• Le modèle suppose que ces décomptes suivent une distribution multinomiale pour chaque classe

En général, elle modélise la probabilité d'observer une distribution particulière du nombre de mots dans un document.

2.4.2 Fonctionnement

Étape 1 : Calcul des probabilités de chaque mot par classe $P(w|c)$

Pour calculer la probabilité d'un mot dans une classe donnée, nous avons trouvé la moyenne de chaque mot pour une classe donnée. Pour la classe c et le mot w, la probabilité conditionnelle est donnée par :

$$P(w|c) = \frac{W}{W_c}$$

Où :

— W est le nombre total d'occurrences du mot w dans tous les documents de la classe c.

— W_c est le nombre total d'occurrences de tous les mots dans les documents de la classe c (c'est-à-dire la somme des fréquences de tous les mots dans cette classe).

Cependant, certains mots avaient un nombre d'occurrences égal à zéro, nous avons donc effectué un lissage de Laplace avec une faible valeur α, où α représente la valeur de lissage pour les mots non vus qui n'apparaissent pas dans les données d'entraînement.

$$P(w|c) = \frac{\text{count}(w, c) + \alpha}{T_c + \alpha V}$$

Où :

— $\text{count}(w, c)$: le nombre d'occurrences du mot w dans la classe c.

— T_c : le total de mots dans la classe c.

— V : le nombre de mots distincts dans le corpus.

On suppose généralement que $\alpha = 1$ car c'est la valeur par défaut du lissage de Laplace dans de nombreux algorithmes et implémentations. C'est un choix de compromis qui fonctionne bien dans la majorité des cas.

La formule devient donc :

$$P(w|c) = \frac{\text{count}(w, c) + 1}{T_c + V}$$

Étape 2 : Calcul des probabilités a priori de chaque classe

La probabilité à priori de chaque classe c est donnée par :

$$P(c) = \frac{\text{nombre de documents de la classe } c}{\text{nombre total de documents}}$$

Étape 3 : Calcul de la probabilité d'une classe c donnée un document D

$$P(c|D) = \frac{P(D|c) \cdot P(c)}{P(D)}$$

P(c|D) La probabilité d'un document D donné une classe c est :

$$P(D|c) = \prod_{i=1}^{f(w,d)} P(w|c)$$

Où :

— $f(w, d)$: la fréquence d'apparition du mot w dans le document d.

Afin d'éviter un sous-débordement numérique, nous utilisons la somme des logarithmes :

$$\log(P(c|D)) = \log(P(c)) + \sum_{w \in d} f(w, D) \cdot \log(P(w|c))$$

L'utilisation de la **transformation logarithmique** dans le contexte du classificateur Naïve Bayes est une étape essentielle pour :

— Éviter le sous-débordement numérique, lorsque vous multipliez de nombreuses petites probabilités.

— Améliorer la stabilité numérique.

— Faciliter les calculs, car le logarithme transforme les produits en sommes.

Étape 4 : Comparer et classer

Classement du document : Le document est attribué à la classe c qui maximise $P(c|D)$, ce qui correspond à la classe ayant le score le plus élevé.

$$f(D) = y_{\text{predict}}(D) = \arg\max_{c} P(c \mid D)$$

Chapitre 3

Application Et Implementation

3.1 Utilisation

Grâce à sa simplicité et à son efficacité, Naïve Bayes est largement utilisé dans de nombreux domaines :

Algorithme Naïve Bayes	Exemples d'Applications
Naïve Bayes Gaussien	**Exemples :** - **Diagnostic médical :** Estimer la probabilité d'une maladie en fonction des symptômes (par exemple, prédiction de maladies cardiaques en fonction de facteurs comme l'âge, la tension artérielle, etc.). - **Prédiction de la qualité d'un produit :** Estimer la qualité d'un vin en fonction de mesures chimiques (par exemple, pH, acidité).
Naïve Bayes Multinomial	**Exemples :** - **Filtrage de spam :** Identifier les e-mails indésirables en fonction des mots-clés. - **Analyse de sentiments :** Déterminer si un avis est positif ou négatif en analysant les mots utilisés dans le texte. - **Classification de textes :** Regrouper des documents par catégorie (par exemple, politique, sport, science).
Naïve Bayes Bernoulli	**Exemples :** - **Filtrage de spam :** Identifier les e-mails indésirables en fonction de la présence ou de l'absence de certains mots-clés. - **Reconnaissance d'écriture :** Classification des caractères manuscrits, par exemple pour la reconnaissance de chiffres ou de lettres dans les systèmes OCR.

TABLE 3.1 – Exemples d'Applications des Algorithmes Naïve Bayes

3.2 Implémentation du système

3.2.1 Bernoulli et gaussian

1- Import libraries

```
import numpy as np
import math
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from sklearn.metrics import accuracy_score, confusion_matrix, f1_score
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

2- Import dataset

```
#chargement de dataset NaiveBayes
df=pd.read_csv('NaiveBayes.csv')
df
```

3- Exploratory data analysis

```
#afficher la dimention de dataset
df. shape = (400, 3)
#afficher les cinqs premier lignes
df.head()
```

	Age	salary	Purchased
0	19	19000	0
1	35	20000	0
2	26	43000	0
3	27	57000	0
4	19	76000	0

FIGURE 3.1 – les cinqs premier lignes

- **Variables et caractéristiques**

Attribut	Description
Age	Âge d'un individus. [années]
salary	Le salaire des individus.
Purchased	: Une variable binaire (0 ou 1) indiquant si l'individu a effectué un achat (1) ou non (0).

TABLE 3.2 – les attributs de base données naive bayes

- **Descriptive Statique**

df.describe() :determine la discription statistique de base de donnée

	count	mean	std	min	25%	50%	75%	max
Age	400	37.655	10.482877	18	29.75	37	46	60
Salary	400	69742.50	34096.960282	15000	43000	70000	88000	150000
Purchased	400	0.357500	0.479864	0	0	0	1	1

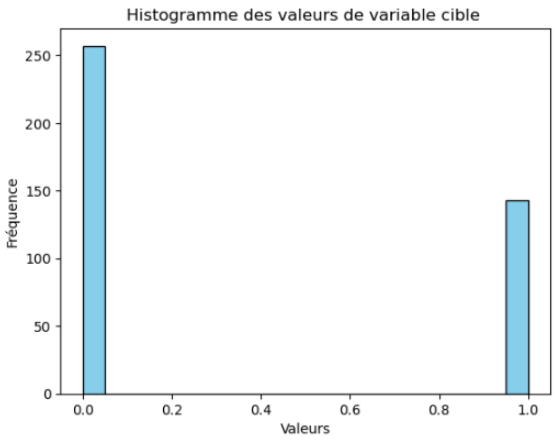
FIGURE 3.2 – description statistique de base de données

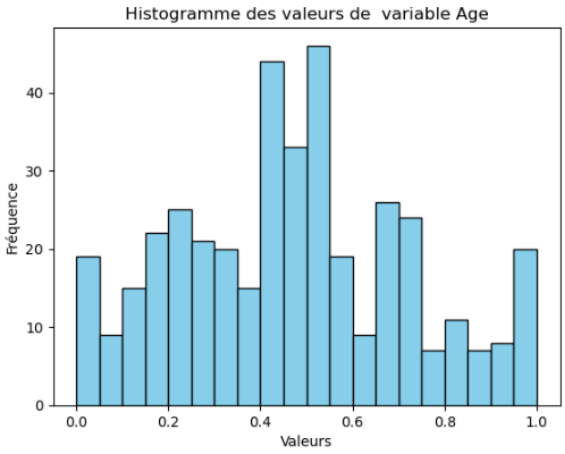
- **Analyse de corrélation**

	Age	salary	Purchased
Age	1.000000	0.155238	0.622454
salary	0.155238	1.000000	0.362083
Purchased	0.622454	0.362083	1.000000

FIGURE 3.3 – matrice de correlation de base données

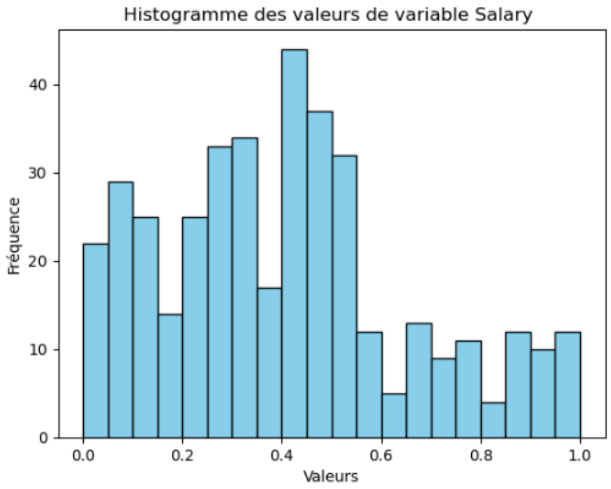
4- Visualisation de données



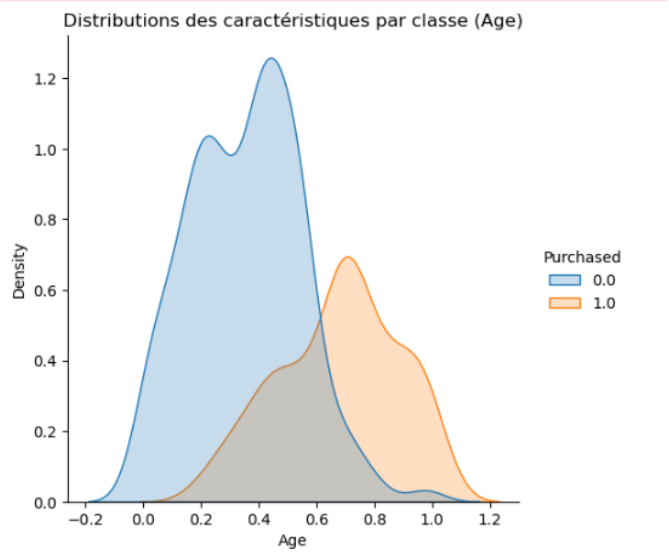


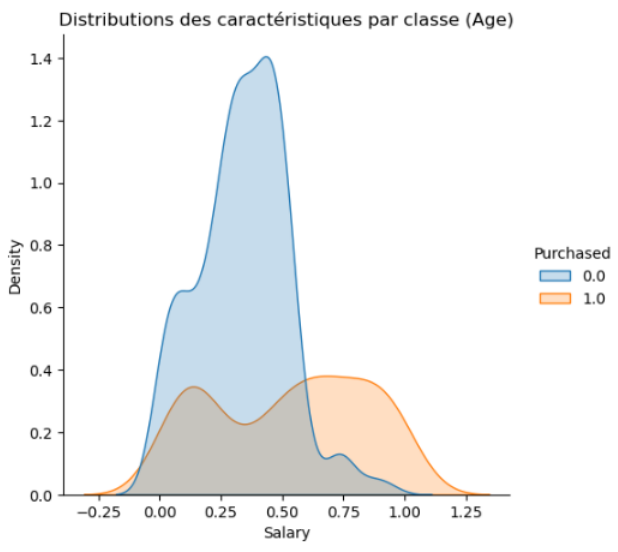
Cet histogramme indique que la variable cible est de nature binaire et qu'elle présente une distribution déséquilibrée. Ceci pourrait être majeur à prendre en évaluation dans le cadre de la modélisation ou de l'analyse des données

Cet histogramme démontre que la variable "Âge"est distribuée de manière continue et qu'elle ne présente pas une nature binaire, à la divergence du précédent histogramme. Certaines valeurs semblent être plus courantes (près de 0.4 et 0.6), ce qui pourrait signaler la présence de groupes d'âge plus nombreux dans l'échantillon. Si la variable "Age" a été standardisée (ajustée pour se situer entre 0 et 1), cette distribution peut servir à appréhender la variété des âges contenus dans les données.



Cet histogramme indique que la plupart des salaires se situent dans la première partie des valeurs normalisées. Cela pourrait indiquer que, dans ce groupe, une majorité d'individus reçoit un salaire proche de la moyenne inférieure, tandis que les salaires plus hauts sont relativement rares. Si la variable " Salaire" a été standardisée de 0 à 1, cela traduit une répartition habituelle des salaires où la plupart des individus gagnent moins, alors qu'une petite partie perçoit des revenus plus importants.





Le graphique pour Purchased = 1.0 semble indiquer une densité supérieure pour des âges plus élevés (proches de 1.0), ce qui indique que les personnes plus âgées ont une résolution plus grande à faire un achat. Le graphique pour Purchased = 0.0 révèle une présence plus importante de densité à des âges plus bas (proches de 0.0), signifiant que les personnes plus jeunes ont moins habitude à faire un achat.

Il semble que la courbe pour Purchased = 1.0 présente une densité plus importante lorsque les valeurs de "Salary " sont supérieures (proches de 1.0), ce qui indique que les personnes ayant un salaire plus élevé ont préférence à réaliser davantage d'achats. Le graphique pour Purchased = 0.0 indique une importance plus majeure pour des "Salary" inférieurs (proches de 0.0), indiquant que les personnes ayant un salaire élevé sont moins attirées à faire des achats.

5- Prétretement de données

1.detecté les valeurs aberrants

```
categoreis_data=[]
numeriques_data=[]
for i,c in enumerate(df.dtypes):
    if c==object:
        categoreis_data.append(df.iloc[:,i])
    else:
        numeriques_data.append(df.iloc[:,i])
categoreis_data=pd.DataFrame(categoreis_data).transpose()
numeriques_data=pd.DataFrame( numeriques_data).transpose()
numeriques_data = df.select_dtypes(include=[np.number]).columns
for col in numeriques_data:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    df[col] = np.where((df[col] < lower_bound) | (df[col] > upper_bound), df[col].median(),
print(df[numeriques_data].describe())
```

	count	mean	std	min	25%	50%	75%	max
Age	400	37.655	10.482877	18	29.75	37	46	60
Salary	400	69742.50	34096.960282	15000	43000	70000	88000	150000
Purchased	400	0.357500	0.479864	0	0	0	1	1

FIGURE 3.4 – description statistique de base de données

2.normalisation des données

```
from sklearn.preprocessing import MinMaxScaler

# Initialisation du MinMaxScaler
scaler = MinMaxScaler()
```

```
# Appliquer la normalisation sur les colonnes numériques
df[numeriques_data] = scaler.fit_transform(df[numeriques_data])

# Afficher les premières lignes pour vérifier la normalisation
print(df[numeriques_data].head(10))
```

	Age	Salary	Purchased
0	0.023810	0.029630	0.0
1	0.404762	0.037037	0.0
2	0.190476	0.207407	0.0
3	0.214286	0.311111	0.0
4	0.023810	0.451852	0.0
5	0.214286	0.318519	0.0
6	0.214286	0.511111	0.0
7	0.333333	1.000000	1.0
8	0.166667	0.133333	0.0
9	0.404762	0.370370	0.0

FIGURE 3.5 – Enter Caption

3.Gestion des déséquilibres par la methode SMOT

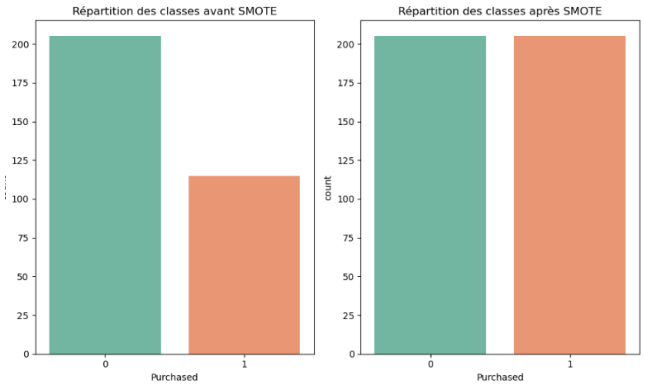
faire une énéquililbre entre les deux classe pour eviter que le modèle biais vers la classe majoritaire

```
import pandas as pd
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)

# Séparation des features (X) et de la cible (y)
X = df.drop('Purchased', axis=1)  # Supprimer la colonne cible des features
y = df['Purchased']  # La colonne cible

# Si nécessaire, convertir 'Survived' en une variable binaire (0 ou 1) si elle est continu
#  arrondir à 0 ou 1 si c'est une valeur continue
y = (y > 0.5).astype(int)

# Pré-traitement des données : encodage, imputation des valeurs manquantes, etc.
# Exemple d'encodage pour les colonnes catégorielles
X = pd.get_dummies(X, drop_first=True)

# Diviser les données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Vérification des valeurs uniques dans y_train
print(y_train.value_counts())

# Initialiser SMOTE
smote = SMOTE(sampling_strategy='auto', random_state=42)

# Appliquer SMOTE pour générer des données synthétiques
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)

# Vérifier la répartition des classes avant et après SMOTE
print(f"Répartition des classes avant SMOTE: \n{y_train.value_counts()}")
print(f"Répartition des classes après SMOTE: \n{y_resampled.value_counts()}")
```

4.Afichage de la gestion des déséquilibres

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(10, 6))
plt.subplot(1, 2, 1)
sns.countplot(x=y_train, palette='Set2')
plt.title('Répartition des classes avant SMOTE')
plt.subplot(1, 2, 2)
sns.countplot(x=y_resampled, palette='Set2')
plt.title('Répartition des classes après SMOTE')
plt.tight_layout()
plt.show()
```



6- Declare feature vector and target variable

. séparation de data

```
X = df.iloc[:, [0,1]].values
y = df.iloc[:, 2].values
```

7- Split data into separate training and test set

Nous avons déviser le jeu de données en deux phases :90% pour la phase d'entrainement et 10%pour la phase de teste

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=0)
```

X_train.shape =((360, 2), X_test.shape =(40, 2))

8- les modèles de naive bayes

a- Gaussien naive bayes

1.Conversion des données pour le modèle Naive Bayes

Le code convertit les données d'entraînement et de test dans un format compatible avec l'algorithme Naive Bayes.

```
train_data = [list(row) + [label] for row, label in zip(x_train.values, y_train.values)]
test_samples = x_test.values
```

2.Calcul de la probabilité gaussienne

•Le code calcule la probabilité gaussienne pour .une valeur donnée en utilisant la moyenne et l'écart-type. Fonction principale :

```
def proba_gaussienne(x, mean, sigma)
```

utilisée pour calculer la vraisemblance dans le modèle Naive Bayes Gaussien.

3. Calcul de la probabilité a priori

Le code calcule la probabilité a priori d'une classe. Fonction principale :

```
def proba_priori(y, data)
```

Calculer la probabilité qu'une classe apparaisse dans les données.

4. Calcul de la moyenne et de la variance

Le code calcule la moyenne et la variance d'une caractéristique pour une classe donnée. Fonction principale :

```
def calculate_mean_variance(class\_data)
```

Ces statistiques sont utilisées pour calculer la probabilité gaussienne.

5. Implémentation du classificateur Naive Bayes Gaussien

Le code implémente l'algorithme Naive Bayes Gaussien pour prédire la classe d'un échantillon de test. Fonction principale :

```
def gaussian_naive_bayes(train_data, test_sample)
```

Cette fonction calcule les probabilités a posteriori pour chaque classe et retourne la classe avec la probabilité la plus élevée.

6. Prédiction sur l'ensemble de test

Le code utilise le modèle Naive Bayes Gaussien pour prédire les classes des échantillons de test.
Fonction principale :

```
def [gaussian_naive_bayes(train_data, test_sample) for test_sample in test_samples]
```

Générer des prédictions pour l'ensemble de test.

```
''' Convertir les données d'entraînement et de test dans un format

compatible avec la fonction Naive Bayes'''

train_data = [list(row) + [label] for row, label in zip(x_train.values, y_train.values)]
test_samples = x_test.values

# La fonction pour calculer la probabilité de la distribution gaussienne
def proba_gaussienne(x, mean, sigma):
    expo = math.exp(-(math.pow(x - mean, 2) / (2 * math.pow(sigma, 2))))
    return (1 / (math.sqrt(2 * math.pi) * sigma)) * expo

# Fonction pour calculer la probabilité a priori P(y)
def proba_priori(y, data):
    return data.count(y) / len(data)

# La fonction pour calculer la moyenne et la variance d'une caractéristique d'une classe
def calculate_mean_variance(class_data):
    mean = sum(class_data) / len(class_data)
    variance = sum((x - mean) ** 2 for x in class_data) / len(class_data)  # Variance
    return mean, variance

# Fonction Naive Bayes Gaussien pour calculer la probabilité a posteriori
def gaussian_naive_bayes(train_data, test_sample):

    classes = list(set([row[-1] for row in train_data]))
    predict_posteriori = {}

''' Séparer les données par classe et calculer la moyenne et la variance

pour chaque caractéristique'''

    Dec = {}
    for c in classes:
        # Filtrer les échantillons pour la classe c
        class_data = [row[:-1] for row in train_data if row[-1] == c]

        means = []
```

```
        variances = []
        for i in range(len(class_data[0])):
            feature_data = [row[i] for row in class_data]
            mean, variance = calculate_mean_variance(feature_data)
            means.append(mean)
            variances.append(variance)

        Dec[c] = {'mean': means, 'variance': variances}

# Calcul des probabilités a priori P(y)
    prior_probs = {c: proba_priori(c, [row[-1] for row in train_data]) for c in classes}

# Calcul de la probabilité marginale P(x) =  P(x|y) P(y)
    proba_marginal= 0
    for c in classes:
        vraisemblance = 1
        for i, value in enumerate(test_sample):
            mean = Dec[c]['mean'][i]
            variance = Dec[c]['variance'][i]
            vraisemblance *= proba_gaussienne(value, mean, math.sqrt(variance))
        proba_marginal += vraisemblance * prior_probs[c]

# Calcul des probabilités a posteriori P(y|x) pour chaque classe
    for c in classes:
        vraisemblance = 1
        for i, value in enumerate(test_sample):
            mean = Dec[c]['mean'][i]
            variance = Dec[c]['variance'][i]
            vraisemblance *= proba_gaussienne(value, mean, math.sqrt(variance))
# Calcul de la probabilité a posteriori P(y|x)  P(x|y) * P(y)
        predict_posteriori[c] = vraisemblance * prior_probs[c] / proba_marginal

    return max(predict_posteriori, key=predict_posteriori.get)


# Prédictions sur l'ensemble de test
predict_posteriori = [gaussian_naive_bayes(train_data, test_sample)
for test_sample in test_samples]

# Calcul de la précision (accuracy)
accuracy = accuracy_score(y_test, predict_posteriori)
print(f"Précision du modèle : {accuracy * 100:.2f}%")

# Calcul de la matrice de confusion
conf_matrix = confusion_matrix(y_test, predict_posteriori)

# Affichage de la matrice de confusion
```

```
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix,
                              display_labels=['Non-Acheté', 'Acheté'])

disp.plot(cmap=plt.cm.Blues)
plt.title('Matrice de Confusion')
plt.show()

# Affichage du rapport de classification
print("Rapport de classification :")
print(classification_report(y_test, predict_posteriori))
```

b- Bernoulli naive bayes
Définition du code principale de bernoulliNB
premiere fonction utilisé dans le code

```
class BernoulliNaiveBayes:
    def __init__(self,alpha = 1):
        self.alpha =alpha
        return
```

Cette fonction initialise un objet de la classe BernoulliNaiveBayes en définissant un paramètre de lissage Laplace (alpha).
Le lissage de Laplace est utilisé pour éviter des probabilités nulles lorsque certaines valeurs n'apparaissent pas dans l'ensemble d'entraînement.
Entrée :
alpha (float) : paramètre de lissage (par défaut 1).
Sortie :
Aucune, mais stocke alpha dans l'objet.
deuxièmme fonction

```
def fit(self, X, y)
```

Cette fonction entraîne le modèle en estimant :
Les probabilités a priori des classes P(y).
Les probabilités conditionnelles pour chaque classe.
Entrées :
X (array de taille [n_samples, n_features]) : les données d'entraînement (binaire).
y (array de taille [n_samples]) : les étiquettes de classe associées.
Sortie :
Aucune, mais stocke :
log_class_prior : logarithme des probabilités des classes P(y).
log_class_conditional_positive : logarithme de $P(x \mid y)$.
log_class_conditional_negative : logarithme de $P(-x \mid y)$.

Explication du fonctionnement
Calcul des probabilités à priori P(y)

```
y_counts = np.unique(y, return_counts=True)[1]
self.n_classes = len(np.unique(y))
```

```
self.n_features = X.shape[1]
class_prior = y_counts / y_counts.sum()
self.log_class_prior = np.expand_dims(np.log(class_prior), axis=1)
```

Compte le nombre d'occurrences de chaque classe. Calcule P(y).
Prend le logarithme pour éviter les erreurs numériques.
Calcul des probabilités conditionnelles avec lissage de Laplace

```
prob_x_acondition_y = np.zeros([self.n_classes, self.n_features])
for i in range(self.n_classes):
    row_mask = (y == i)
    X_filtered = X[row_mask, :]
    numerator = (X_filtered.sum(axis=0) + self.alpha)
    denominator = (X_filtered.shape[0] + 2 * self.alpha)
    prob_x_acondition_y[i, :] = numerator / denominator
```

Pour chaque classe, filtre les lignes correspondantes.
Calcule $P(x \mid y)$ en appliquant le lissage de Laplace.

$$P(x_j = 1|y = i) = \frac{\sum X_j + \alpha}{\text{total échantillons} + 2\alpha}$$

Stockage des logarithmes des probabilités

```
self.log_class_conditional_positive = np.log(prob_x_acondition_y)
self.log_class_conditional_negative = np.log(1 - prob_x_acondition_y)
```

troisièmme fonction

```
def predict(self, X)
```

Prédit la classe des nouvelles données en calculant la vraisemblance conjointe $P(y \mid X)$et en choisissant la classe avec la probabilité maximale.
Entrée :
X (array de taille [n_samples, n_features]) : nouvelles données binaires.
Sortie :
preds (array de taille [n_samples]) : classes prédites.
Explication du fonctionnement :
Conversion des matrices creuses

```
if hasattr(X, "todense"):
    X = X.todense()
```

Convertit X en matrice dense si c'est une matrice creuse.
Calcul des probabilités logarithmiques

```
log_probs_positive = self.log_class_conditional_positive.dot(X.T)
log_probs_negative = self.log_class_conditional_negative.dot(1 - X.T)
log_likelihoods = log_probs_positive + log_probs_negative
log_joint_likelihoods = log_likelihoods + self.log_class_prior
```

Calcule la vraisemblance $P(X \mid y)$ pour chaque classe :
$\log P(x \mid y)$ pour les variables présentes.
$\log P(-x \mid y)$ pour les variables absentes.
Ajoute logP(y) pour obtenir la vraisemblance conjointe P(X,y).
Sélection de la classe la plus probable

```
preds = np.argmax(log_joint_likelihoods, axis=0)
preds = np.array(preds).squeeze()
return preds
```

9- l'évaluation des modèles bernoulli et gaussien

Rapport de classification :

	precision	recall	f1-score	support
0.0	0.86	1.00	0.93	32
1.0	1.00	0.38	0.55	8
accuracy			0.88	40
macro avg	0.93	0.69	0.74	40
weighted avg	0.89	0.88	0.85	40

FIGURE 3.6 – Valeurs du metriques de bernoulli

	precision	recall	f1-score	support
0	0.90	0.93	0.92	30
1	0.78	0.70	0.74	10
accuracy			0.88	40
macro avg	0.84	0.82	0.83	40
weighted avg	0.87	0.88	0.87	40

FIGURE 3.7 – Valeurs du metriques de Gaussian

Le modèle est très efficace pour la classe 0.0, affichant une précision de 86% , un rappel de 100% et un score F1 de 93% . Pour la classe 1.0, le modèle affiche une précision parfaite (100%), mais un rappel assez faible (38%), ce qui montre qu'il n'identifie pas suffisamment de vrais positifs pour cette catégorie. La précision générale est de 88, ce qui est satisfaisant, cependant le faible taux de rappel pour la classe 1.0 indique que le modèle pourrait être orienté vers la classe majoritaire (0.0).

Pour la classe 0, le modèle affiche de bonnes performances avec une précision de 90% , un rappel de 93 et un score F1 de 92% . Le modèle affiche une précision de 78% et un rappel de 70 % pour la classe 1, ce qui indique d'une performance moins optimale dans l'identification précise de cette classe. Le taux de précision global est de 88% , ce qui est satisfaisant, mais les résultats pour la classe 1 ne sont pas aussi bons que pour la classe 0.

matrice de confusion

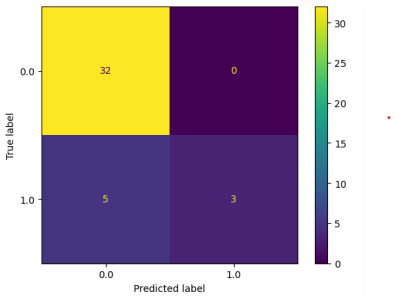


FIGURE 3.8 – matrice de confusion de bernoulli

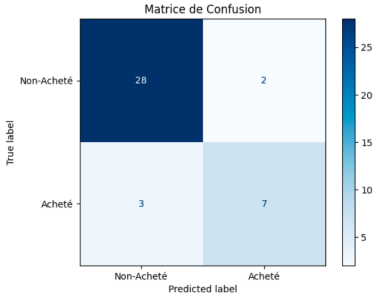


FIGURE 3.9 – matrice de confusion de Gaussian

Nous remarquons que le modèle naive bayes bernoulli est très efficace pour cette data par-ce-qu'elle

est déscrite pour cela naive bayes bernoulli est données des bonn resultats,soient au niveau de les metriques soit au niveau de matrice de confusion qui mentre que le modèle predicte bien ou non

3.2.2 Multinomial

La mise en œuvre de l'algorithme Naïve Bayes Multinomial un ensemble de données de Kaggle pour la classification de texte à l'aide de la bibliothèque scikit-learn de Python est décrite ci-dessous.

Ensemble de données On a extrait un morceau de 500 lignes dans un data-frame "Company Documents Dataset" contient une collection de plus de 2 000 documents d'entreprise, classés en quatre types principaux : factures, rapports d'inventaire, bons de commande et bons d'expédition. Chaque document est fourni au format PDF, accompagné d'un fichier CSV qui comprend le texte extrait de ces documents, leurs libellés respectifs et le nombre de mots de chaque document.

1- Charger et visualisation DataFrame

— Aprés d'installation des bibliothèques nécessaires l'ensemble de données peut être facilement ajouté sous forme de DataFrame pandas à l'aide de la fonction "read_csv".

```
import pandas as pd
df = pd.read_csv('company-document-text.csv', nrows=500)}
```

— Les 5 premières ligne data :

```
df.head()
```

	text	label	word_count
0	order id 10718 shipping details ship name k...	ShippingOrder	120
1	invoice order id 10707 customer id arout ord...	invoice	66
2	order id 10448 shipping details ship name r...	ShippingOrder	96
3	invoice order id 11068 customer id queen ord...	invoice	68
4	order id 10656 shipping details ship name g...	ShippingOrder	109

FIGURE 3.10 – Les 5 premières ligne data

— Affiche le type de données, le nombre de valeurs non nulles et la mémoire utilisée à l'aide de la méthode "info()" :

```
df.info()
```

```
les informations générale sur dataframe:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   text        500 non-null    object
 1   label       500 non-null    object
 2   word_count  500 non-null    int64
dtypes: int64(1), object(2)
memory usage: 11.8+ KB
```

FIGURE 3.11 – Les informations sur data

2- Prétraitement des données

— Travail juste sur deux colonnes.

```
df=df.drop('word_count',axis=1)
df.head()
```

	text	label
0	order id 10718 shipping details ship name k...	ShippingOrder
1	invoice order id 10707 customer id arout ord...	invoice
2	order id 10448 shipping details ship name r...	ShippingOrder
3	invoice order id 11068 customer id queen ord...	invoice
4	order id 10656 shipping details ship name g...	ShippingOrder

FIGURE 3.12 – data de travail finale

Remarque : Dans la classification de texte si vous voulez ou devez de réduire l'impact des mots très fréquents dans tous les documents peut utiliser Le **TF-IDF** qui est une mesure statistique permet de distinguer les mots importants dans un document en prenant en compte à la fois leur fréquence dans le document et leur rareté à travers l'ensemble du corpus.

— Convertir le texte en entités numériques à l'aide de "CountVectorizer".

Processus :

• Tokenisation des mots : Diviser le texte en unités appropriées.

• Créer un vocabulaire à partir de l'ensemble de données et compter la fréquence de chaque mot dans chaque document.

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(df['text'])

print("Matrice de caractéristiques numériques l'aide de CountVectorizer():")
print(X.toarray())
```

```
Matrice de caractéristiques numériques l'aide de CountVectorizer():
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 1]
 [0 0 0 ... 0 0 0]]
```

FIGURE 3.13 – Matrice numérique

— Encodage ordinal de la colonne "label".

• Affichage des valeurs unique de colonne label :

```
print(f'label: {df['label'].unique()}')
```

```
label: ['ShippingOrder' 'invoice' 'purchase Order' 'report']
```

FIGURE 3.14 – les valeurs unique de colonne "label"

• Encodage : "isin" Filtrer où la colonne 'label' contient ('ShippingOrder', 'invoice', 'purchase Order' ou 'report')

```
# Supprimer ou remplacer les valeurs inconnues dans 'label'
df = df[df['label'].isin(['ShippingOrder', 'invoice', 'purchase Order', 'report'])]

df['label']=df['label'].map({'ShippingOrder': 0,'invoice': 1,
                             'purchase Order':2,'report':3})

print("Colonne 'label' après encodage ordinal:")
print(df['label'])
```

```
Colonne 'label' après encodage ordinal:
0      0
1      1
2      0
3      1
4      0
      ..
495    0
496    0
497    2
498    3
499    0
Name: label, Length: 500, dtype: int64
```

FIGURE 3.15 – les valeurs numériques de colonne "label"

3- Entraînement et évaluation du classificateur

Nous divisons l'ensemble de données en un ensemble d'entraînement et un ensemble de test à l'aide de la fonction `train_test_split` de scikit-learn, en utilisant 20% des données pour les tests.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X,df['label'],
                                  test_size=0.2,random_state=42)

print("data_train:\n X_train :")
print(X_train.toarray().flatten())
print("y_train :")
print(y_train.head())

print("\ndata_test:\n X_test :")
print(X_test.toarray().flatten())
print("y_test :")
print(y_test.head())
```

```
data_train:
 X_train :
[0 0 0 ... 0 0 0]
y_train :
249    0
433    0
19     2
322    3
332    2
Name: label, dtype: int64
```

FIGURE 3.16 – Ensemble des données d'entrainment

```
data_test:
 X_test :
[0 0 0 ... 0 0 0]
y_test :
361    3
73     3
374    1
155    3
104    2
Name: label, dtype: int64
```

FIGURE 3.17 – Ensemble des données de teste

- Verifier les Données Déséquilibrées :

Remarque : l'algorithemes naive bayes multinomaial sensible aux données déséquilibrées pour qu'il puisse donner des résultats biaisés si une classe possède beaucoup plus de données que les autres(une classe majoré) mais si le contre la classe minoré peut gérer dans "naive bayes multinomaial" l'aide de lissage de Laplace.

```
plt.figure(figsize=(10, 6))
plt.subplot(1, 2, 1)
sns.countplot(x=y_train, palette='Set2')
plt.title('Répartition des classes avant SMOTE')
plt.tight_layout()
plt.show()
```

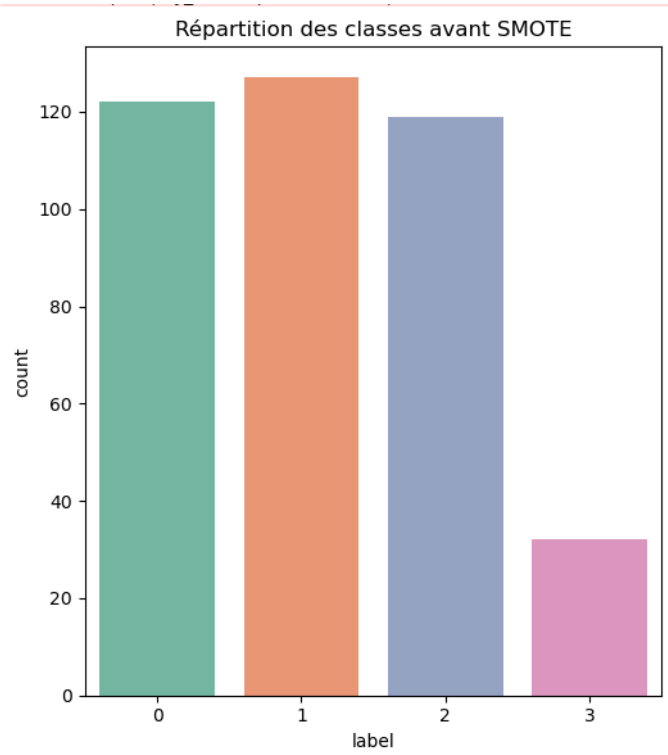


FIGURE 3.18 – les Données Déséquilibrée

–> On remarque l'absence d'une classe majoré.

• Définition de la class "MultinomialNB" qui contient le cour de notre méthode naive bayes multinomial

```
  class MultinomialNB:
''' def __init__() stoker les valeurs nécessaire dans les fonctions
principale de notre classe'''
    def __init__(stock):
        stock.classes = []
        stock.proba_w = {}
        stock.prop = {}

    def fit(stock,X, Y):
        #probabilité P(wc)
        stock.classes= np.unique(Y)
        count_w={c:np.zeros(X.shape[1]) for c in stock.classes}
        total={c: 0 for c in stock.classes}

        for c,d in zip(Y,X):
         ''' toarray().flatten() retourne un copie (1D) indépendente de
         tableau d '''
            count_w[c]+=d.toarray().flatten()
            total[c]+=np.sum(d)
```

```
        V=len(X[0].toarray()[0])
        #proba_w={}
        for c in stock.classes:
            stock.proba_w[c]=(count_w[c]+1)/(total[c]+V)

        #probabilité P(c)
        T_doc=X.shape[0]
        S_doc={c: 0 for c in stock.classes}
        for c in Y:
            S_doc[c]+=1
        #prop={}
        for c in stock.classes:
            stock.prop[c]=S_doc[c]/T_doc

    def predit(stock,X):
        predicions=[]
        for d in X:
           #probabilité P(c|D)
           prob_MNB={}
           for c in stock.classes:
              log_prob=np.log(stock.prop[c])
              for i,w in enumerate(d.toarray().flatten()):
                 if w > 0:
                    log_prob=log_prob+np.log(stock.proba_w[c][i])
              prob_MNB[c]=log_prob
            #comparer et classer
           predicions.append(max(prob_MNB,key=prob_MNB.get))

        return predicions
```

- Crée un objet "model" de class "MultinomialNB"

```
model= MultinomialNB()
```

- Ensuite, nous entraînons le classificateur à l'aide de la fonction `fit()`.

```
model.fit(X_train,y_train)
```

4- Résultats et évaluation

Après l'exécution de l'algorithme, Nous avons obtenu les résultats suivants :

```
y_pred=model.predit(X_test)
print("predicions:",y_pred)
```

```
predicions: [3, 3, 1, 3, 2, 3, 1, 2, 2, 2, 2, 2, 3, 1, 1, 0, 0, 2, 0, 0, 0, 1, 0, 1, 1, 1,
2, 2, 0, 2, 1, 0, 2, 0, 0, 2, 1, 1, 0, 1, 1, 0, 3, 2, 0, 2, 0, 1, 2, 2, 2, 0, 2, 2, 1, 1,
2, 1, 0, 0, 2, 3, 1, 1, 0, 2, 3, 0, 2, 0, 2, 2, 2, 2, 0, 0, 1, 1, 2, 1, 2, 3, 2, 1, 0, 0,
2, 2, 0, 0, 0, 2, 0, 1, 2, 0, 1, 1, 1, 1]
```

FIGURE 3.19 – les valeurs predicte par l'algorithme

- les métriques d'évaluation, telles que la précision et le taux de rappel, sont affichées.

```
from sklearn.metrics import classification_report
target_names = ['class 0', 'class 1', 'class 2', 'class 3']
print(classification_report(y_test, y_pred, target_names=target_names))
```

	precision	recall	f1-score	support
class 0	1.00	1.00	1.00	29
class 1	1.00	1.00	1.00	28
class 2	1.00	1.00	1.00	34
class 3	1.00	1.00	1.00	9
accuracy			1.00	100
macro avg	1.00	1.00	1.00	100
weighted avg	1.00	1.00	1.00	100

FIGURE 3.20 – les valeurs des metriques

- La matrice de confusion est également présentée pour évaluer la performance du modèle.

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
mar_con=confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(mar_con, annot=True, fmt="d", cmap="Blues",
            xticklabels=["ShippingOrder", "invoice", "purchase Order", "report"],
            yticklabels=["ShippingOrder", "invoice", "purchase Order", "report"])
plt.title("Matrice de confusion")
plt.xlabel("Prédictions")
plt.ylabel("Réel")
plt.show()
```

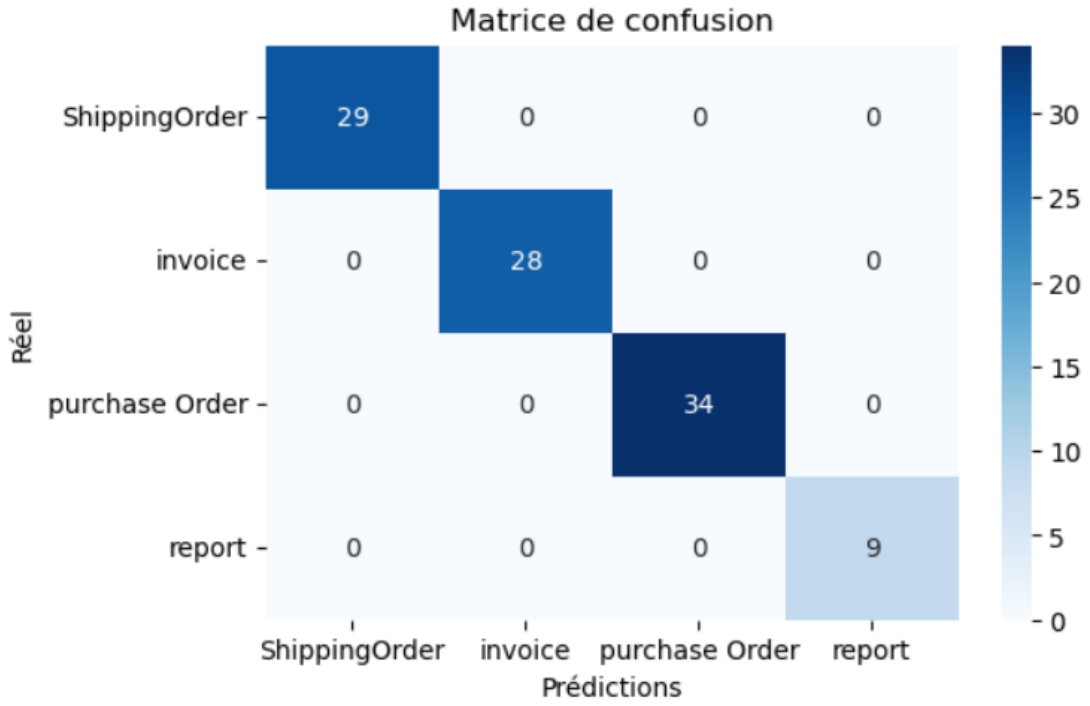


FIGURE 3.21 – matrice de confision

L'algorithme MNB gère rapidement de grands ensembles de données, ce qui le rend idéal pour les tâches de classification de texte. Il fournit des résultats clairs basés sur les probabilités et fonctionne bien même en présence de mots supplémentaires ou sans rapport, car il se concentre sur les modèles généraux du texte. Grâce à l'application du lissage de Laplace et à l'absence de données déséquilibrées dans le jeu de données (en particulier lorsqu'une classe possède beaucoup plus de données que les autres, ce qui pourrait entraîner des résultats biaisés), la précision du modèle est améliorée. Cela contribue à obtenir de bons résultats au niveau des métriques (precision, recall, f1-score, support et accuracy=1.00).

3.3 Schémas d'exécution

Schéma en général

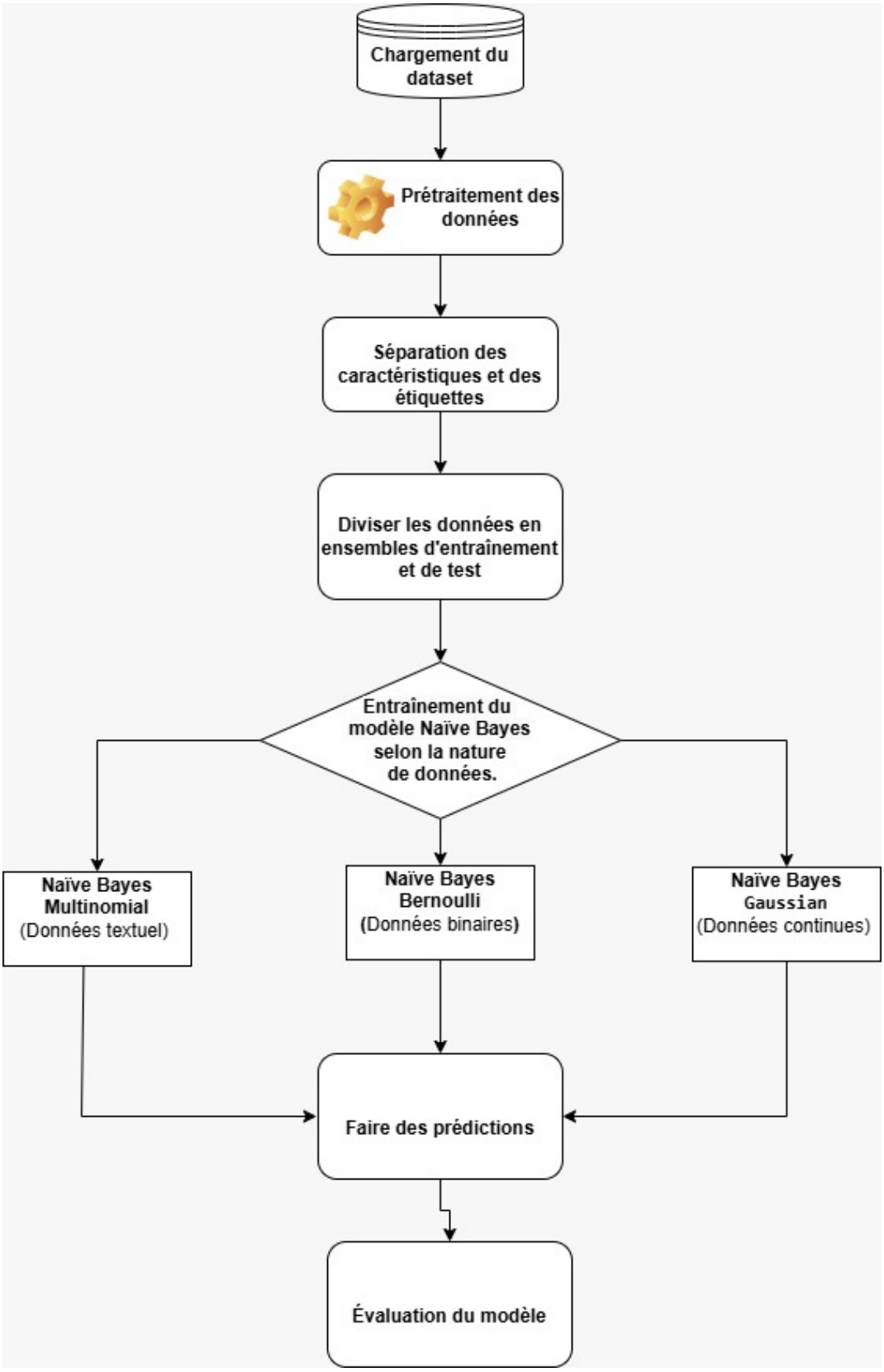


FIGURE 3.22 – Schéma général

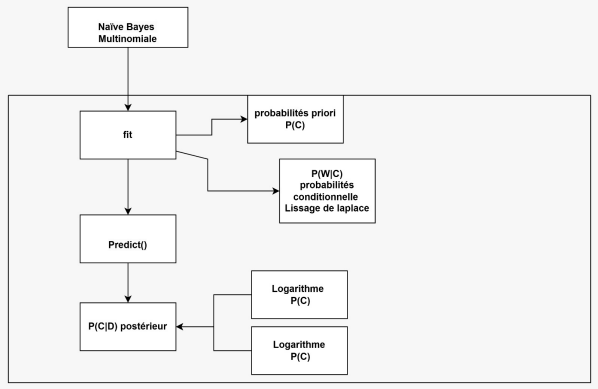


FIGURE 3.23 – Multinomial

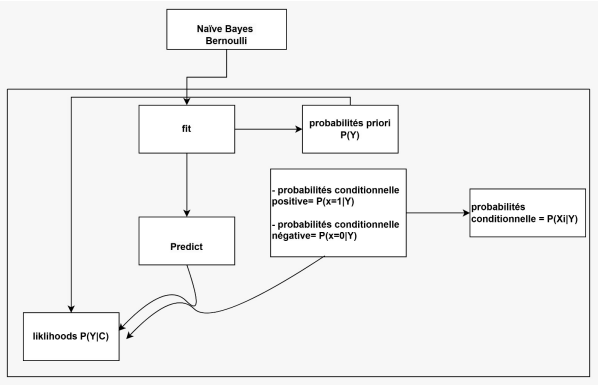


FIGURE 3.24 – Bernoulli

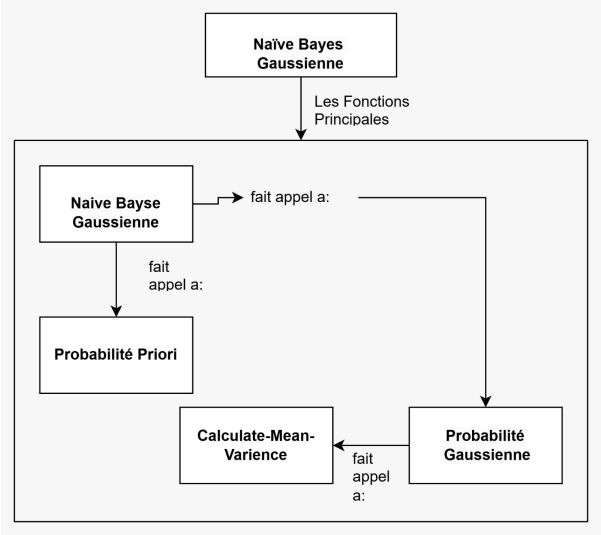


FIGURE 3.25 – Gaussien

Conclusion

Naïve Bayes est un algorithme puissant et rapide, bien adapté aux grandes bases de données textuelles ou aux tâches de classification simple. Malgré son hypothèse forte d'indépendance conditionnelle, il donne souvent de bons résultats et constitue un excellent point de départ pour les problèmes de classification probabiliste. Cependant, ses performances peuvent être limitées dans des situations où les variables explicatives sont fortement corrélées. Son utilisation reste néanmoins très répandue dans des domaines variés, en raison de sa facilité de mise en œuvre et de sa robustesse.

Résumé :

• Le théorème de Bayes : est utilisé pour mettre à jour les probabilités en fonction de nouvelles preuves.

• L'indépendance des fonctionnalités : simplifie les calculs en supposant que les mots apparaissent indépendamment.

• Le lissage de Laplace : empêche les probabilités nulles pour les mots non vus dans les données d'entraînement.

• Probabilités : calculées à l'aide de données d'entraînement avec lissage pour gérer les comptages nuls

• La décision de classification : est prise en comparant les probabilités postérieures

Bibliographie

Sources des bases de données :
• Gaussien/ Bernoulli
https://www.kaggle.com/datasets/helloitsdaksh/datasheet
• Multinomial
https://www.kaggle.com/datasets/ayoubcherguelaine/company-documents-dataset
https://www.researchgate.net/profile/Anees-Ahmed-5/publication/334451164_Multinomial_Naive_Bayes_Classification_Model_for_Sentiment_Analysis/links/5e227e8d92851cafc38c813c/Multinomial-Naive-Bayes-Classification-Model-for-Sentiment-Anal
pdf
https://www.upgrad.com/blog/multinomial-naive-bayes-explained/
https://medium.com/@kashishdafe0410/gaussian-naive-bayes-understanding-the-basics-and-ap
https://www.geeksforgeeks.org/bernoulli-naive-bayes/