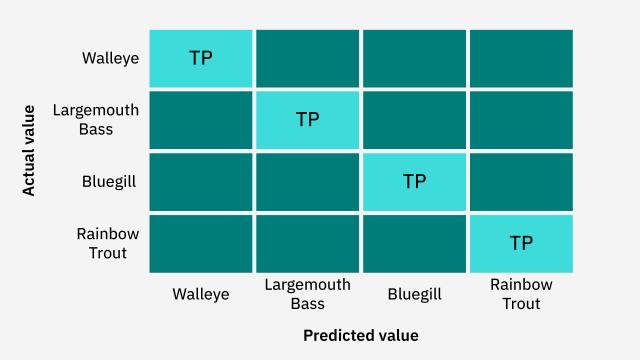
Comparativa clasificadores NaiveBayes



Adrián Yared Armas de la Nuez

**Contenido**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

[**1. Enunciado 2**](#_2c35568x6wg8)

[**2. Actividad 3**](#_95u6rtumiz9g)

[**2.1 Iris Dataset 3**](#_t8cwbo6n4466)

[**2.1.1 Imports 3**](#_aig92oe7k2m0)

[**2.1.2 Función para trazar la matriz de confusión 3**](#_e9uqlcl56mu)

[**2.1.3 Dataset cargado 4**](#_pas1hurje558)

[**2.1.4 Train y test 4**](#_8reip0vub9b8)

[**2.1.5 Clasificadores 5**](#_29303gf1t7q5)

[**2.1.6 Almacenar resultados 5**](#_ga4hc12h1ktp)

[**2.1.6 Resumen de los resultados 6**](#_qkk1h1nmp614)

[**2.1.7 Resultados 6**](#_4e6lthmhfqbt)

[**2.1.7.1 GaussianNB 6**](#_k0wslrxpur8l)

[**2.1.7.2 MultinomialNB 7**](#_jizj69qspz94)

[**2.1.7.3 ComplementNB 7**](#_68uys4ykqycb)

[**2.1.7.4 BernoulliNB 8**](#_k3caotxyjd2z)

[**2.1.7.5 CategoricalNB 8**](#_5qk7z4on6wrd)

[**2.1.7.6 Resumen de los resultados 9**](#_mnmqpphi6nnk)

[**2.1.7.7 Justificación 9**](#_vlkj2c1sqocj)

[**2.1 Penguin Dataset 9**](#_239e7990fnt4)

[**2.1.1 Imports 9**](#_tfjdymd0jpnf)

[**2.1.2 Función para trazar la matriz de confusión 9**](#_9jsl9xscfek)

[**2.1.3 Dataset carga 10**](#_b95lyaecjik0)

[**2.1.4 Funciones categóricas del código para CategoricalNB 11**](#_o2f5ukhg4b9b)

[**2.1.5 Train y test 11**](#_yf7ss1yxn5bo)

[**2.1.6 Clasificadores 11**](#_p99q3zjbpsax)

[**2.1.7 Almacenar resultados 12**](#_hjrf44ko19pp)

[**2.1.8 Resumen de los resultados 13**](#_sfxejay1r1ik)

[**2.1.9 Resultados 13**](#_ltwws2uakltj)

[**2.1.9.1 GaussianNB 13**](#_nxen8d6gtcmu)

[**2.1.9.2 MultinomialNB 14**](#_dts65cldanuy)

[**2.1.9.3 ComplementNB 14**](#_v92efq94dbnc)

[**2.1.9.4 BernoulliNB 15**](#_ei42un7h6kxn)

[**2.1.9.5 Resumen del resultado 15**](#_81drq4lprumy)

[**2.1.9.6 Justificación 15**](#_nr87hqkv0v9)

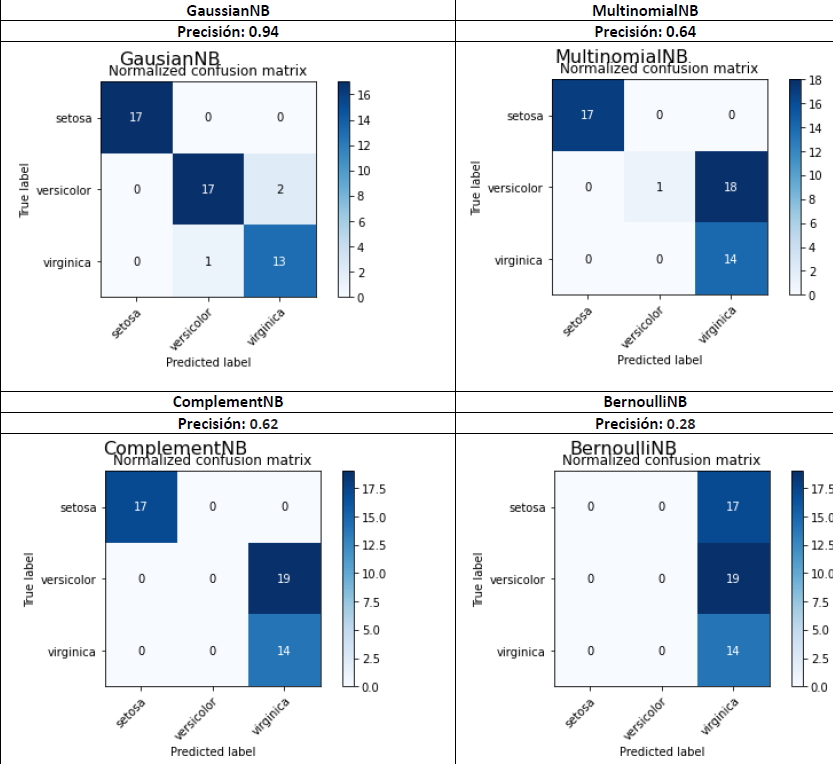
[**3. Github y Colab 16**](#_lmdngf8dlin3)

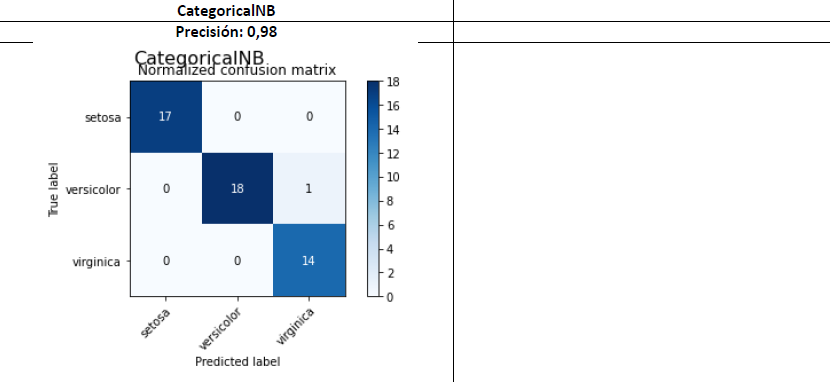
## **1. Enunciado**

Utilizando como referencia el cuaderno Ejemplo\_3\_2\_Iris\_NaiveBayes - GaussianNB.ipynb, realizar otros tantos cuadernos, o desarrollar la solución como consideres oportuno, con los diferentes clasificadores NaiveBayes, de forma que para un mismo problema podamos comparar las precisiones obtenidas.

Realizar esta comparativa para el Dataset Iris y el Dataset Penguin (por separado)

A modo de ejemplo, la comparativa se podría representar de la siguiente manera:





## **2. Actividad**

### **2.1 Iris Dataset**

#### **2.1.1 Imports**

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import GaussianNB, MultinomialNB, CategoricalNB, ComplementNB, BernoulliNB

from sklearn.metrics import confusion\_matrix, accuracy\_score

from sklearn.utils.multiclass import unique\_labels

#### **2.1.2 Función para trazar la matriz de confusión**

# Function to plot confusion matrix

def plot\_confusion\_matrix(y\_true, y\_pred, classes,

normalize=False,

title=None,

cmap=plt.cm.Blues,

titleSup=None):

"""

Prints and plots the confusion matrix. Normalization can be applied by setting `normalize=True`.

"""

cm = confusion\_matrix(y\_true, y\_pred)

if normalize:

cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

fig, ax = plt.subplots()

im = ax.imshow(cm, interpolation='nearest', cmap=cmap)

ax.figure.colorbar(im, ax=ax)

ax.set(xticks=np.arange(cm.shape[1]),

yticks=np.arange(cm.shape[0]),

xticklabels=classes, yticklabels=classes,

title=title,

ylabel='True label',

xlabel='Predicted label')

plt.setp(ax.get\_xticklabels(), rotation=45, ha="right",

rotation\_mode="anchor")

fmt = '.2f' if normalize else 'd'

thresh = cm.max() / 2.

for i in range(cm.shape[0]):

for j in range(cm.shape[1]):

ax.text(j, i, format(cm[i, j], fmt),

ha="center", va="center",

color="white" if cm[i, j] > thresh else "black")

fig.tight\_layout()

fig.suptitle(titleSup, fontsize=16, y=1, ha='center')

plt.show()

#### **2.1.3 Dataset cargado**

# Load Iris dataset

iris = sns.load\_dataset('iris')

X\_iris = iris.drop('species', axis=1)

y\_iris = iris['species']

#### **2.1.4 Train y test**

# Split into training and test sets

Xtrain, Xtest, ytrain, ytest = train\_test\_split(X\_iris, y\_iris, test\_size=0.33, random\_state=1)

#### **2.1.5 Clasificadores**

# Define classifiers

classifiers = {

'GaussianNB': GaussianNB(),

'MultinomialNB': MultinomialNB(),

'ComplementNB': ComplementNB(),

'BernoulliNB': BernoulliNB(),

'CategoricalNB': CategoricalNB()

}

#### **2.1.6 Almacenar resultados**

# Store results

results = {}

clases\_iris = np.array(['setosa', 'versicolor', 'virginica'])

for name, model in classifiers.items():

try:

# Train the model

model.fit(Xtrain, ytrain)

# Predict

y\_pred = model.predict(Xtest)

# Evaluate

acc = accuracy\_score(ytest, y\_pred)

results[name] = {

'accuracy': acc,

'y\_pred': y\_pred

}

print(f"{name} - Accuracy: {acc:.2f}")

# Convert categorical labels for the confusion matrix

ytest\_num = ytest.replace(['setosa', 'versicolor', 'virginica'], [0, 1, 2]).to\_numpy()

y\_pred\_num = pd.Series(y\_pred).replace(['setosa', 'versicolor', 'virginica'], [0, 1, 2]).to\_numpy()

# Plot normalized confusion matrix

plot\_confusion\_matrix(

ytest\_num, y\_pred\_num,

classes=clases\_iris,

normalize=True,

title=f'{name} - Normalized Confusion Matrix',

titleSup=name

)

except Exception as e:

print(f"{name} encountered an error: {e}")

#### **2.1.6 Resumen de los resultados**

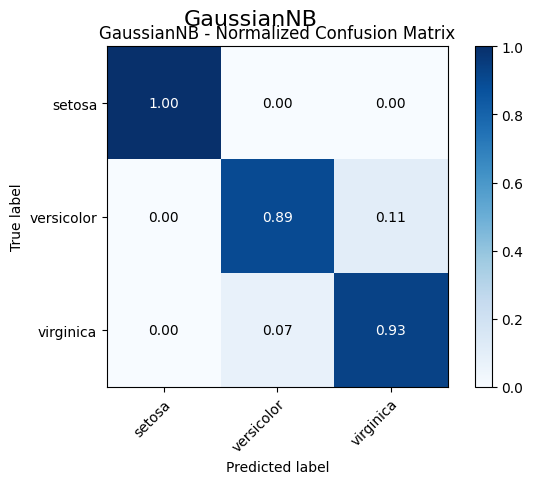
# Summary of results

for name, result in results.items():

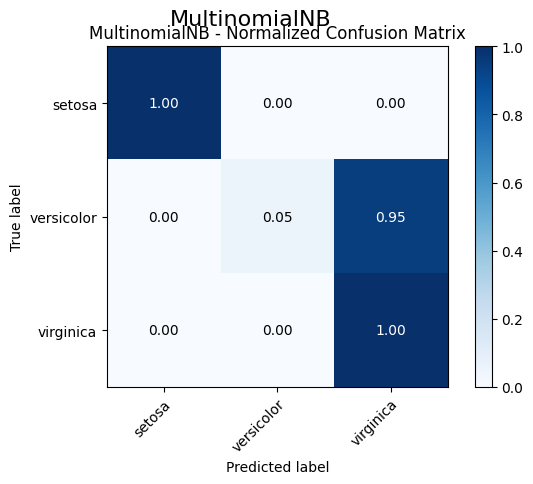
print(f"{name}: Accuracy = {result['accuracy']:.2f}")

#### **2.1.7 Resultados**

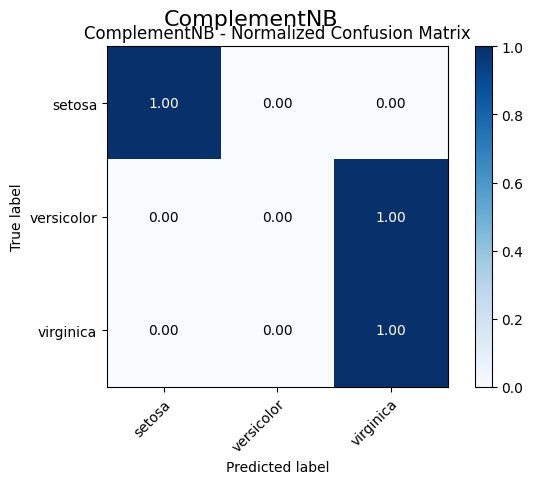
##### **2.1.7.1 GaussianNB**



##### **2.1.7.2 MultinomialNB**

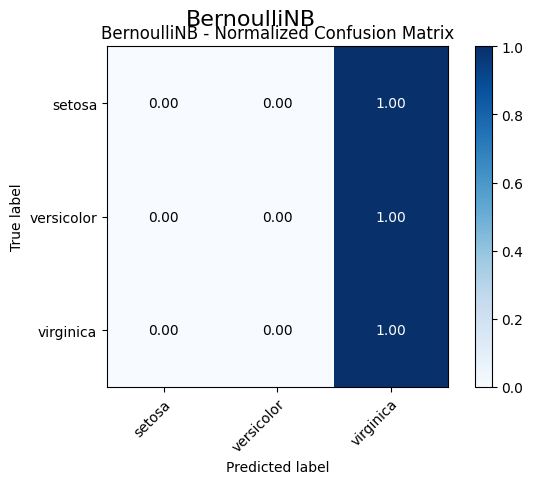


##### **2.1.7.3 ComplementNB**

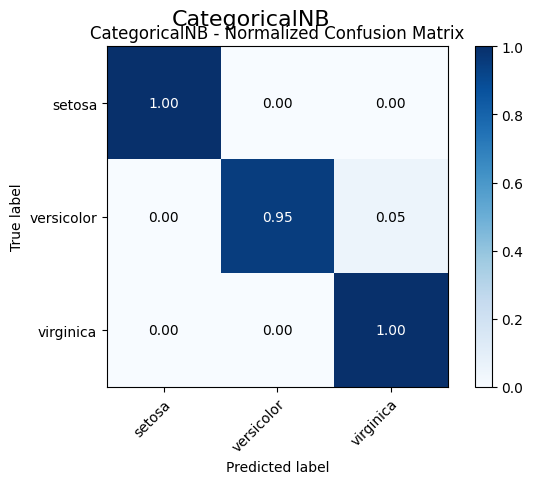
****

##### 

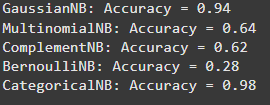
##### **2.1.7.4 BernoulliNB**

****

##### **2.1.7.5 CategoricalNB**

****

##### **2.1.7.6 Resumen de los resultados**

****

##### **2.1.7.7 Justificación**

El resultado es lógico, ya que muestra una variabilidad esperada en el rendimiento de los diferentes modelos. El GaussianNB tiene una alta precisión de 0.94, lo cual es razonable si los datos son aproximadamente gaussianos. El MultinomialNB y ComplementNB tienen resultados más bajos, lo que sugiere que los datos pueden no ajustarse bien a sus supuestos de distribución. El BernoulliNB muestra una precisión muy baja, indicando que su suposición de variables binarias no es adecuada. Finalmente, el CategoricalNB tiene una precisión muy alta, lo que sugiere que sus supuestos de datos categóricos son más adecuados para este caso.

### **2.1 Penguin Dataset**

#### **2.1.1 Imports**

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import GaussianNB, MultinomialNB, ComplementNB, BernoulliNB, CategoricalNB

from sklearn.metrics import confusion\_matrix, accuracy\_score

from sklearn.utils.multiclass import unique\_labels

from sklearn.preprocessing import OrdinalEncoder

#### **2.1.2 Función para trazar la matriz de confusión**

# Function to plot confusion matrix

def plot\_confusion\_matrix(y\_true, y\_pred, classes,

normalize=False,

title=None,

cmap=plt.cm.Blues,

titleSup=None):

"""

Prints and plots the confusion matrix. Normalization can be applied by setting `normalize=True`.

"""

cm = confusion\_matrix(y\_true, y\_pred)

if normalize:

cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

fig, ax = plt.subplots()

im = ax.imshow(cm, interpolation='nearest', cmap=cmap)

ax.figure.colorbar(im, ax=ax)

ax.set(xticks=np.arange(cm.shape[1]),

yticks=np.arange(cm.shape[0]),

xticklabels=classes, yticklabels=classes,

title=title,

ylabel='True label',

xlabel='Predicted label')

plt.setp(ax.get\_xticklabels(), rotation=45, ha="right",

rotation\_mode="anchor")

fmt = '.2f' if normalize else 'd'

thresh = cm.max() / 2.

for i in range(cm.shape[0]):

for j in range(cm.shape[1]):

ax.text(j, i, format(cm[i, j], fmt),

ha="center", va="center",

color="white" if cm[i, j] > thresh else "black")

fig.tight\_layout()

fig.suptitle(titleSup, fontsize=16, y=1, ha='center')

plt.show()

#### **2.1.3 Dataset carga**

# Load Penguins dataset

penguins = sns.load\_dataset('penguins').dropna()

X\_penguins = penguins.drop('species', axis=1)

y\_penguins = penguins['species']

#### **2.1.4 Funciones categóricas del código para CategoricalNB**

### **# Encode categorical features for CategoricalNB**

### **categorical\_features = X\_penguins.select\_dtypes(include=['object']).columns**

### **encoder = OrdinalEncoder()**

### **X\_penguins\_categorical = X\_penguins.copy()**

### **X\_penguins\_categorical[categorical\_features] = encoder.fit\_transform(X\_penguins\_categorical[categorical\_features])**

### 

### **# Encode categorical features for other models**

### **X\_penguins\_encoded = pd.get\_dummies(X\_penguins, drop\_first=True)**

### 

#### **2.1.5 Train y test**

# Split into training and test sets

Xtrain\_cat, Xtest\_cat, ytrain\_cat, ytest\_cat = train\_test\_split(X\_penguins\_categorical, y\_penguins, test\_size=0.33, random\_state=1)

Xtrain, Xtest, ytrain, ytest = train\_test\_split(X\_penguins\_encoded, y\_penguins, test\_size=0.33, random\_state=1)

#### **2.1.6 Clasificadores**

# Define classifiers

classifiers = {

'GaussianNB': GaussianNB(),

'MultinomialNB': MultinomialNB(),

'ComplementNB': ComplementNB(),

'BernoulliNB': BernoulliNB(),

'CategoricalNB': CategoricalNB()

}

#### **2.1.7 Almacenar resultados**

# Store results

results = {}

clases\_penguins = np.array(penguins['species'].unique())

for name, model in classifiers.items():

try:

if name == 'CategoricalNB':

# Train and evaluate CategoricalNB

model.fit(Xtrain\_cat, ytrain\_cat)

y\_pred = model.predict(Xtest\_cat)

acc = accuracy\_score(ytest\_cat, y\_pred)

else:

# Train and evaluate other models

model.fit(Xtrain, ytrain)

y\_pred = model.predict(Xtest)

acc = accuracy\_score(ytest, y\_pred)

results[name] = {

'accuracy': acc,

'y\_pred': y\_pred

}

print(f"{name} - Accuracy: {acc:.2f}")

# Convert categorical labels for the confusion matrix

ytest\_num = ytest.replace(clases\_penguins, range(len(clases\_penguins))).to\_numpy()

if name == 'CategoricalNB':

ytest\_num = ytest\_cat.replace(clases\_penguins, range(len(clases\_penguins))).to\_numpy()

y\_pred\_num = pd.Series(y\_pred).replace(clases\_penguins, range(len(clases\_penguins))).to\_numpy()

# Plot normalized confusion matrix

plot\_confusion\_matrix(

ytest\_num, y\_pred\_num,

classes=clases\_penguins,

normalize=True,

title=f'{name} - Normalized Confusion Matrix',

titleSup=name

)

except Exception as e:

print(f"{name} encountered an error: {e}")

#### **2.1.8 Resumen de los resultados**

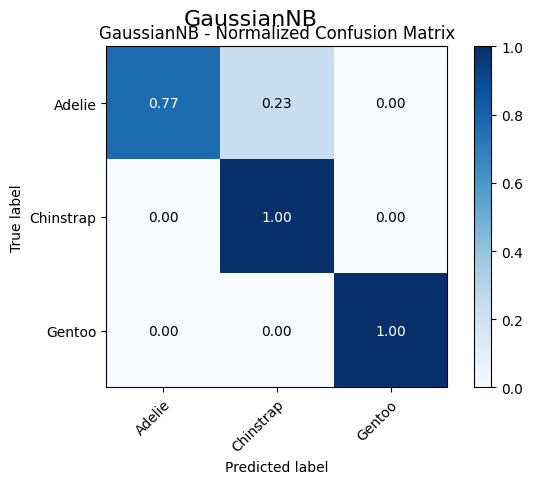
# Summary of results

for name, result in results.items():

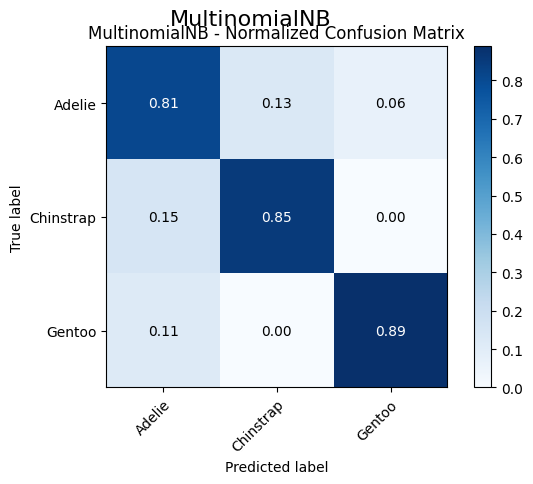
print(f"{name}: Accuracy = {result['accuracy']:.2f}")

#### **2.1.9 Resultados**

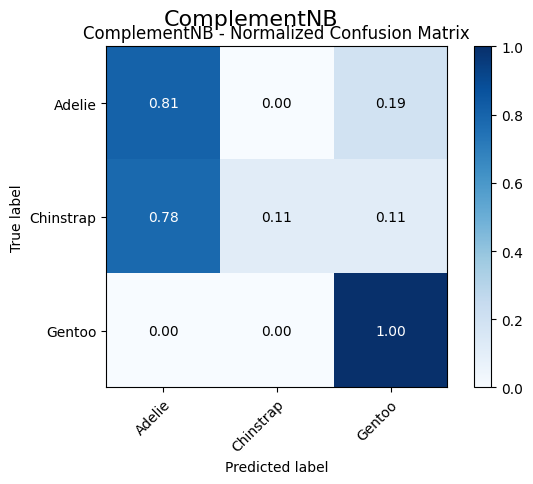
##### **2.1.9.1 GaussianNB**



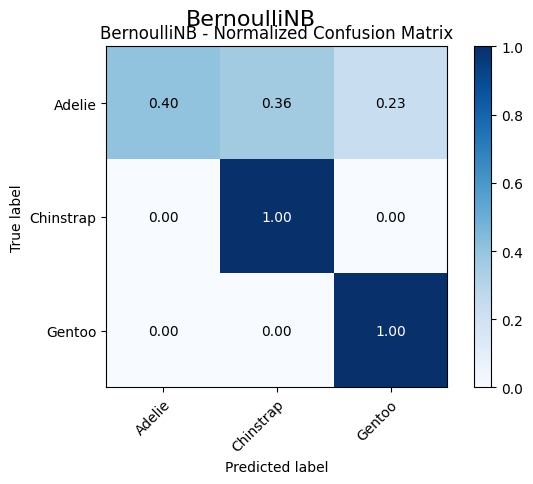
##### **2.1.9.2 MultinomialNB**



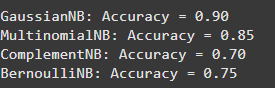
##### **2.1.9.3 ComplementNB**



##### **2.1.9.4 BernoulliNB**



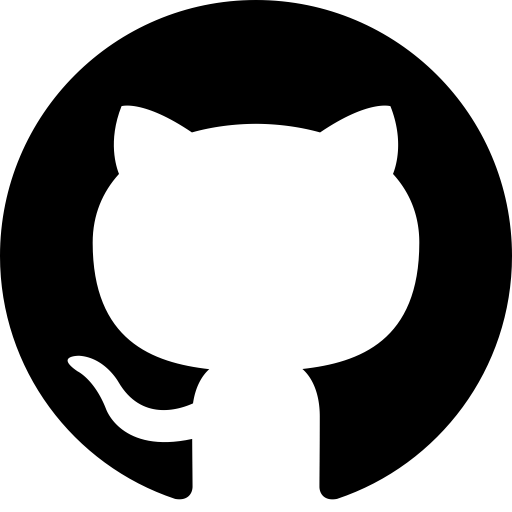
##### **2.1.9.5 Resumen del resultado**



##### **2.1.9.6 Justificación**

El modelo GaussianNB muestra la mejor precisión (0.90), lo que sugiere que la distribución de las características en los datos es aproximadamente gaussiana, favoreciendo este modelo. El MultinomialNB también tiene una precisión alta (0.85), lo que es esperado dado que es eficaz para variables categóricas o de frecuencia. Sin embargo, el ComplementNB (0.70) y el BernoulliNB (0.75) presentan resultados inferiores, lo cual podría indicar que las características de los datos no se ajustan tan bien a sus supuestos, especialmente en términos de independencia entre variables o en el manejo de datos binarios en el caso de Bernoulli.

## **3. Github y Colab**

[](https://github.com/AdrianYArmas/IaBigData/tree/main/SNS/3%20%20-%20Algoritmos%20y%20herramientas%20para%20el%20aprendizaje%20supervisado%20/3.2%20Comparativa%20clasificadores%20NaiveBayes) [](https://colab.research.google.com/drive/1gXhRBMJb-xtVsGL8iQ_y4qfzwr0a3G_h?usp=sharing)