Actividad 3.2 - Comparativa clasificadores NaiveBayes

Utilizando como referencia el cuaderno Ejemplo_3_2_Iris_NaiveBayes – GaussianNB.ipynb (enlace al cuaderno: Cuaderno), 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 penguins (por separado)

Enlace al github: Github

Contenido

Categorical Naive Bayes con dataset de Iris

Comprobamos el dataset de Iris:

```
# importación de datos
import seaborn as sns
iris = sns.load_dataset('iris')
iris
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

Quitamos el tipo de especie del dataframe y hacemos que Y sea el tipo de especie (de modo que el modelo tendrá que adivinar de qué tipo de especie se trata):

```
# separamos datos de entrada y salida
X_iris = iris.drop('species', axis=1)
y_iris = iris['species']
# separamos train y test
from sklearn.model_selection import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(X_iris, y_iris, test_size=0.33, random_state=1)
```

Seleccionamos el modelo que vamos a utilizar, en este caso, el GaussianNB y cargamos sobre este nuestro dataset.

```
# 1. elegimos clasificador
from sklearn.naive_bayes import CategoricalNB
# 2. instanciamos el modelo
model = CategoricalNB()
# 3. Entrenamiento con los datos
# Importante: No es necesario realizar una conversión de variable categóricas a nuemrícas en ytrain porque
#             el modelo está diseñado/pensado para trabajar directamente con variables categóricas.
model.fit(Xtrain, ytrain)
```

```
CategoricalNB
CategoricalNB()
```

Guardamos las especies en una lista:

```
model.classes_
especies = model.classes_.tolist() #guardamos las especies en una lista
print(especies)
```

```
✓ 0.0s
['setosa', 'versicolor', 'virginica']
```

Hacemos la predicción con datos nuevos:

```
# 4. Predicción con nuevos datos
y_model = model.predict(Xtest)
```

Lo evaluamos

```
# 5 evaluación
from sklearn.metrics import accuracy_score
accuracy_score(ytest, y_model) # Precisión del modelo
```

Por último, usamos la siguiente función para ver la matriz de confusión del modelo:

```
from sklearn.metrics import confusion_matrix
from sklearn.utils.multiclass import unique_labels
import matplotlib.pyplot as plt

# y_true : dataframe   -> Los valores de las clases que son ciertos (test)
# y_pred : ndarray     -> Los valores calcuylados de las clases después de
realizar la predicción
# class  : ndarray     -> Los nombres de las clases/valores objetivos

def plot_confusion_matrix(y_true, y_pred, classes,
              normalize=False,
              title=None,
              cmap=plt.cm.Blues,
```

```
            titleSup=None):
"""
This function prints and plots the confusion matrix.
Normalization can be applied by setting `normalize=True`.
"""
if not title:
  if normalize:
    title = 'Normalized confusion matrix'
  else:
    title = 'Confusion matrix, without normalization'

# Compute confusion matrix
cm = confusion_matrix(y_true, y_pred)
# Only use the labels that appear in the data
classes = classes[unique_labels(y_true, y_pred)]
if normalize:
  cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
  print("Normalized confusion matrix")
else:
  print('Confusion matrix, without normalization')

print(cm)

fig, ax = plt.subplots()
im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
ax.figure.colorbar(im, ax=ax)
# We want to show all ticks...
ax.set(xticks=np.arange(cm.shape[1]),
    yticks=np.arange(cm.shape[0]),
    # ... and label them with the respective list entries
    xticklabels=classes, yticklabels=classes,
    title=title,
    ylabel='True label',
    xlabel='Predicted label')

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
     rotation_mode="anchor")

# Loop over data dimensions and create text annotations.
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')
return ax
```

Convertimos todos los “setosa”, “versicolor” y “virginica” en 0, 1 y 2, respectivamente en la Y de test.

```
ytest_df= ytest.to_frame()
ytest_df['species'].replace(especies,[0, 1, 2], inplace=True)
ytest_df
```

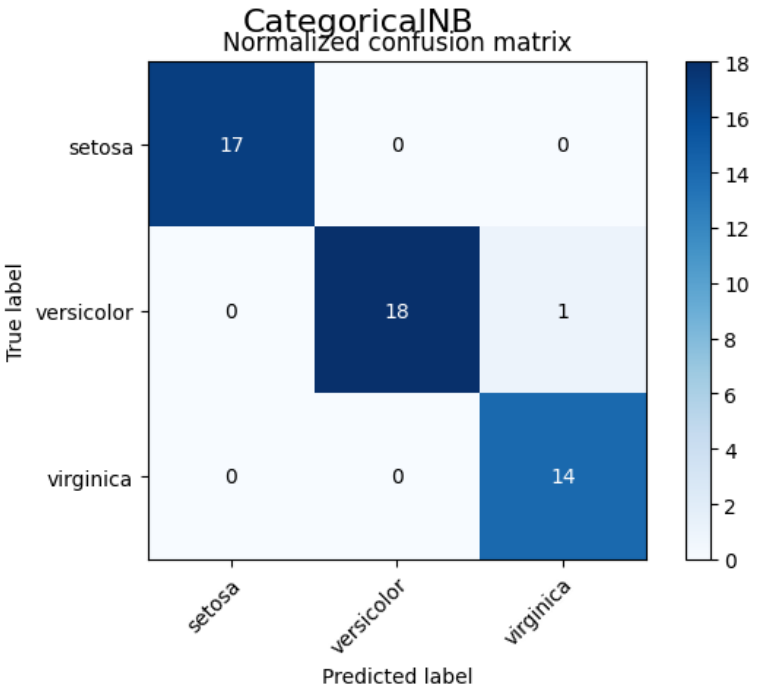
Hacemos lo mismo con los datos de entrenamiento:

```
import pandas as pd
# Transformamos el resultado de la predicción(array) a un dataframe para transformar los valores categóricos en numéricos
y_model_df= pd.DataFrame(y_model, columns = ['species'])
y_model_df['species'].replace(especies,[0, 1, 2], inplace=True)
# Y volvemos a transformar el dataframe a un array, que es el tipo de dato que espera la función plot_confusion_matrix()
y_model_array = y_model_df['species'].to_numpy()
y_model_array
```

Usamos la función con las clases dentro del array clases_iris (He hecho que el título cambie automáticamente conforme cambiamos de modelo):

```
import numpy as np
# Creamos este array porque es el parámetro con las clases que espera la función
clases_iris = np.array(especies)
plot_confusion_matrix(
    ytest_df['species'], y_model_array, classes=clases_iris, normalize=False, title='Normalized confusion matrix', titleSup=model.__class__.__name__)
```

Con eso, obtenemos la matriz de confusión:



CategoricalNB con dataset Penguin

Ahora, especifico los cambios realizados para hacerlo con el dataset Penguin:

(Como anteriormente, modifiqué el Notebook para obtener las especies automáticamente, casi no voy a tener que hacer cambios).

Me doy cuenta de que no existe un dataset llamado Penguin, ya que al sustituirlo, me da error. Sin embargo, al usar print(sns.get_dataset_names()), obtengo todos los nombres de los datasets, incluido uno con un nombre lo bastante parecido.

```python
# importación de datos
import seaborn as sns
penguins = sns.load_dataset('penguins')
penguins
```

✓ 0.4s

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Female
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Female
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	Female
...	...	...	...	...	...	...	...
339	Gentoo	Biscoe	NaN	NaN	NaN	NaN	NaN
340	Gentoo	Biscoe	46.8	14.3	215.0	4850.0	Female
341	Gentoo	Biscoe	50.4	15.7	222.0	5750.0	Male
342	Gentoo	Biscoe	45.2	14.8	212.0	5200.0	Female
343	Gentoo	Biscoe	49.9	16.1	213.0	5400.0	Male

344 rows × 7 columns

En este punto, he tenido que realizar una conversión de variables categóricas a numéricas.

```python
# 1. elegimos clasificador
from sklearn.naive_bayes import CategoricalNB
from sklearn.preprocessing import LabelEncoder
# 2. instanciamos el modelo
model = CategoricalNB()
label_encoder = LabelEncoder()
print(model.__class__.__name__)
for column in Xtrain.columns:
    Xtrain[column] = label_encoder.fit_transform(Xtrain[column])
    # Apply label encoding to each column in Xtrain
for column in Xtest.columns:
    Xtest[column] = label_encoder.fit_transform(Xtest[column])
    # Now you can fit your model
model.fit(Xtrain, ytrain)
```

✓ 0.0s

```
CategoricalNB
```

CategoricalNB ❶ ❷

```
CategoricalNB()
```

Medimos la precisión del modelo:

```python
# 5 evaluación
from sklearn.metrics import accuracy_score
accuracy_score(ytest, y_model) # Precisión del modelo
```

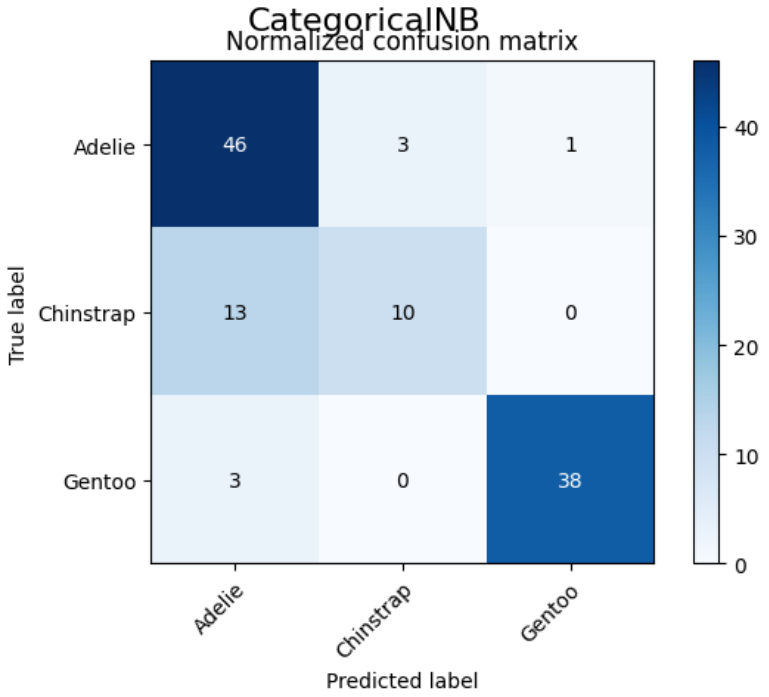
✓ 0.0s

```
0.8245614035087719
```

Mostramos la matriz de confusión:

```
import numpy as np
# Creamos este array porque es el parámetro con las clases que espera la función
clases_penguins = np.array(especies)
plot_confusion_matrix(
    ytest_df['species'], y_model_array, classes=clases_penguins, normalize=False, title='Normalized confusion matrix', titleSup=model.__class__.__name__)
```

Este es el resultado:



BernoulliNB con dataset Iris

Ahora, haremos lo mismo, pero con el clasificador BernoulliNB:
Teniendo en cuenta que ya tenemos todos los cambios automatizados, únicamente tenemos que hacer una copia del notebook con el que usamos el dataset de iris, y sustituir su modelo:

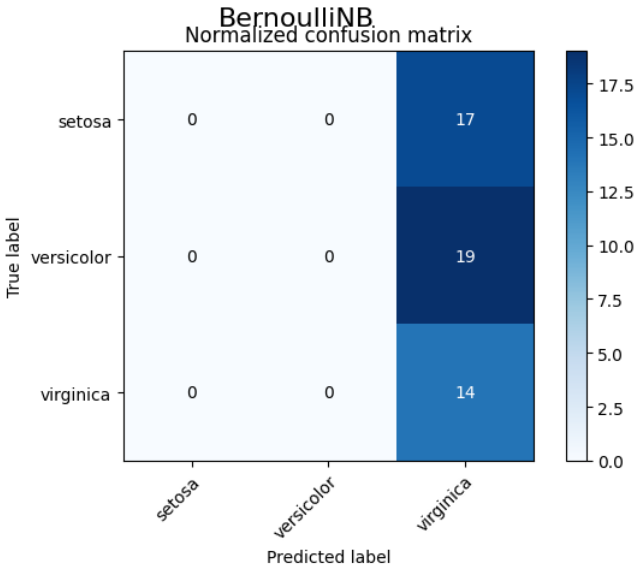
```
# 1. elegimos clasificador
from sklearn.naive_bayes import BernoulliNB
# 2. instanciamos el modelo
model = BernoulliNB()
print(model.__class__.__name__)
# 3. Entrenamiento con los datos
# Importante: No es necesario realizar una conversión de variable categóricas a nuemrícas en ytrain porque
#             el modelo está diseñado/pensado para trabajar directamente con variables categóricas.
model.fit(Xtrain, ytrain)
```

✓ 0.0s

```
BernoulliNB
```

BernoulliNB ❶ ❷
BernoulliNB()

Aquí su respectiva matriz de confusión:



BernoulliNB con dataset penguins

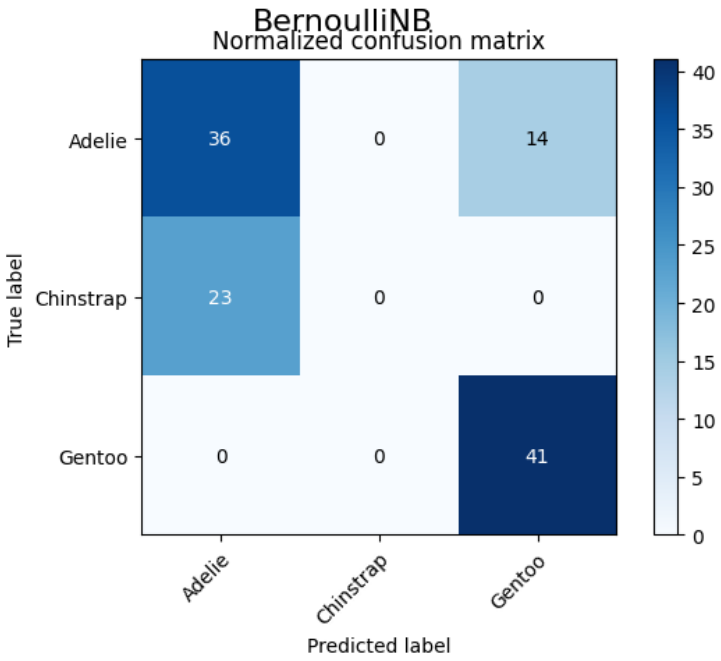
Mismo principio para el dataset de penguins. Cambiamos el modelo a utilizar:

```
# 1. elegimos clasificador
from sklearn.naive_bayes import BernoulliNB
from sklearn.preprocessing import LabelEncoder
# 2. instanciamos el modelo
model = BernoulliNB()
label_encoder = LabelEncoder()
print(model.__class__.__name__)
for column in Xtrain.columns:
    Xtrain[column] = label_encoder.fit_transform(Xtrain[column])
    # Apply label encoding to each column in Xtrain
for column in Xtest.columns:
    Xtest[column] = label_encoder.fit_transform(Xtest[column])
    # Now you can fit your model
model.fit(Xtrain, ytrain)
```

```
✓ 0.0s
BernoulliNB
```

```
BernoulliNB
BernoulliNB()
```

Y como tenemos todos los cambios automatizados, solo tenemos que ir a ver la matriz de confusión resultante:



GaussianNB con dataset Iris

Ahora, probaremos el naive Bayes Gaussiano en el dataset de iris:

```
# 1. elegimos clasificador
from sklearn.naive_bayes import CategoricalNB, BernoulliNB, MultinomialNB,  GaussianNB, ComplementNB
# 2. instanciamos el modelo
model = GaussianNB()
print(model.__class__.__name__)
# 3. Entrenamiento con los datos
# Importante: No es necesario realizar una conversión de variable categóricas a nuemrícas en ytrain porque
#             el modelo está diseñado/pensado para trabajar directamente con variables categóricas.
model.fit(Xtrain, ytrain)
```

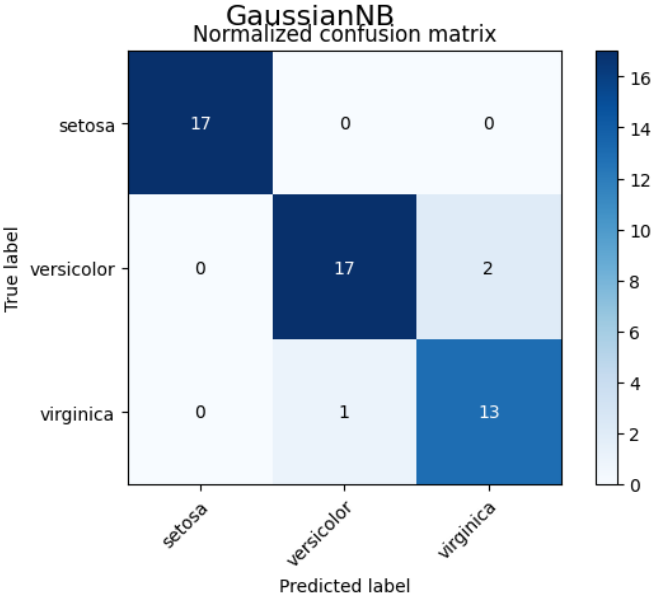
✓ 0.0s

```
GaussianNB
```

GaussianNB

GaussianNB()

Obtenemos su matriz de confusión:



GaussianNB con dataset penguins

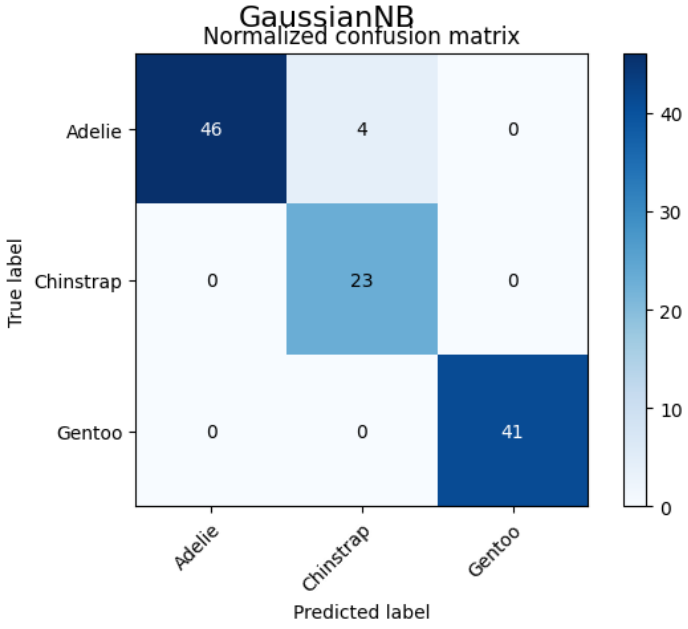
Mismo procedimiento con el data set de penguins:
Cambiamos el modelo:

```
# 1. elegimos clasificador
from sklearn.naive_bayes import CategoricalNB, BernoulliNB, MultinomialNB,  GaussianNB, ComplementNB
# 2. instanciamos el modelo
model = GaussianNB()
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
print(model.__class__.__name__)
for column in Xtrain.columns:
    Xtrain[column] = label_encoder.fit_transform(Xtrain[column])
    # Apply label encoding to each column in Xtrain
for column in Xtest.columns:
    Xtest[column] = label_encoder.fit_transform(Xtest[column])
    # Now you can fit your model
model.fit(Xtrain, ytrain)
```

```
GaussianNB
```

GaussianNB
GaussianNB()

Obtenemos su matriz de confusión:



MultinominalNB con dataset Iris

Ahora, hacemos lo mismo con el modelo MultinomialNB y el dataset Iris:
Cambiamos el modelo:

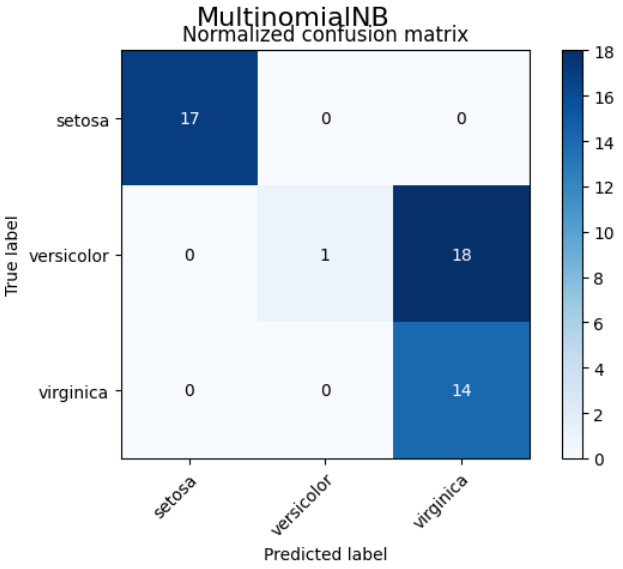
```
    # 1. elegimos clasificador
    from sklearn.naive_bayes import CategoricalNB, BernoulliNB, MultinomialNB,  GaussianNB, ComplementNB
    # 2. instanciamos el modelo
    model = MultinomialNB()
    print(model.__class__.__name__)
    # 3. Entrenamiento con los datos
    # Importante: No es necesario realizar una conversión de variable categóricas a nuemrícas en ytrain porque
    #             el modelo está diseñado/pensado para trabajar directamente con variables categóricas.
    model.fit(Xtrain, ytrain)
```

✓ 0.0s

```
MultinomialNB
```

MultinomialNB ❶ ❷
MultinomialNB()

Obtenemos su matriz de confusión:



MultinominalNB con dataset penguins

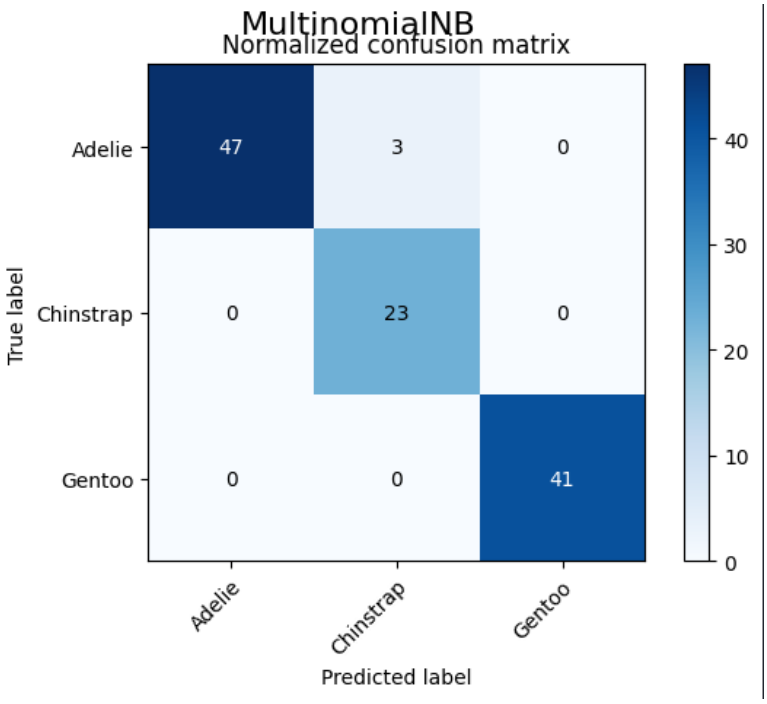
Ahora, con el dataset de penguins:
Cambiamos el modelo:

```
    # 1. elegimos clasificador
    from sklearn.naive_bayes import CategoricalNB, BernoulliNB, MultinomialNB,  GaussianNB, ComplementNB
    # 2. instanciamos el modelo
    model = MultinomialNB()
    from sklearn.preprocessing import LabelEncoder
    label_encoder = LabelEncoder()
    print(model.__class__.__name__)
    for column in Xtrain.columns:
        Xtrain[column] = label_encoder.fit_transform(Xtrain[column])
        # Apply label encoding to each column in Xtrain
    for column in Xtest.columns:
        Xtest[column] = label_encoder.fit_transform(Xtest[column])
        # Now you can fit your model
    model.fit(Xtrain, ytrain)
```

```
MultinomialNB
```

MultinomialNB ❶ ❷
MultinomialNB()

Su matriz de confusión:



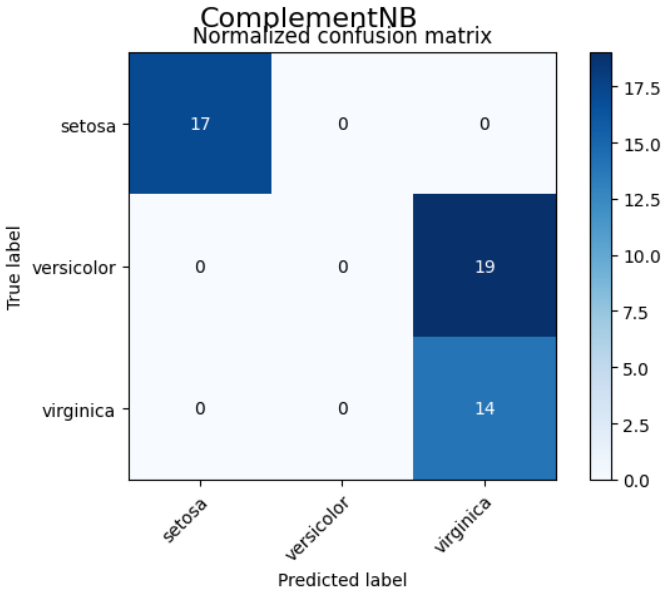
ComplementNB con dataset Iris

Cambiamos el modelo:

```
# 1. elegimos clasificador
from sklearn.naive_bayes import CategoricalNB, BernoulliNB, MultinomialNB,  GaussianNB, ComplementNB
# 2. instanciamos el modelo
model = ComplementNB()
print(model.__class__.__name__)
# 3. Entrenamiento con los datos
# Importante: No es necesario realizar una conversión de variable categóricas a nuemrícas en ytrain porque
#             el modelo está diseñado/pensado para trabajar directamente con variables categóricas.
model.fit(Xtrain, ytrain)
```

```
0.0s
ComplementNB
ComplementNB
ComplementNB()
```

Y obtenemos su matriz de confusión:



ComplementNB con dataset penguins

Cambiamos el modelo utilizado:

```
# 1. elegimos clasificador
from sklearn.naive_bayes import CategoricalNB, BernoulliNB, MultinomialNB,  GaussianNB, ComplementNB
# 2. instanciamos el modelo
model = ComplementNB()
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
print(model.__class__.__name__)
for column in Xtrain.columns:
    Xtrain[column] = label_encoder.fit_transform(Xtrain[column])
    # Apply label encoding to each column in Xtrain
for column in Xtest.columns:
    Xtest[column] = label_encoder.fit_transform(Xtest[column])
    # Now you can fit your model
model.fit(Xtrain, ytrain)
```

✓ 0.0s

```
ComplementNB
```

ComplementNB

```
ComplementNB()
```

Y obtenemos su matriz de confusión: