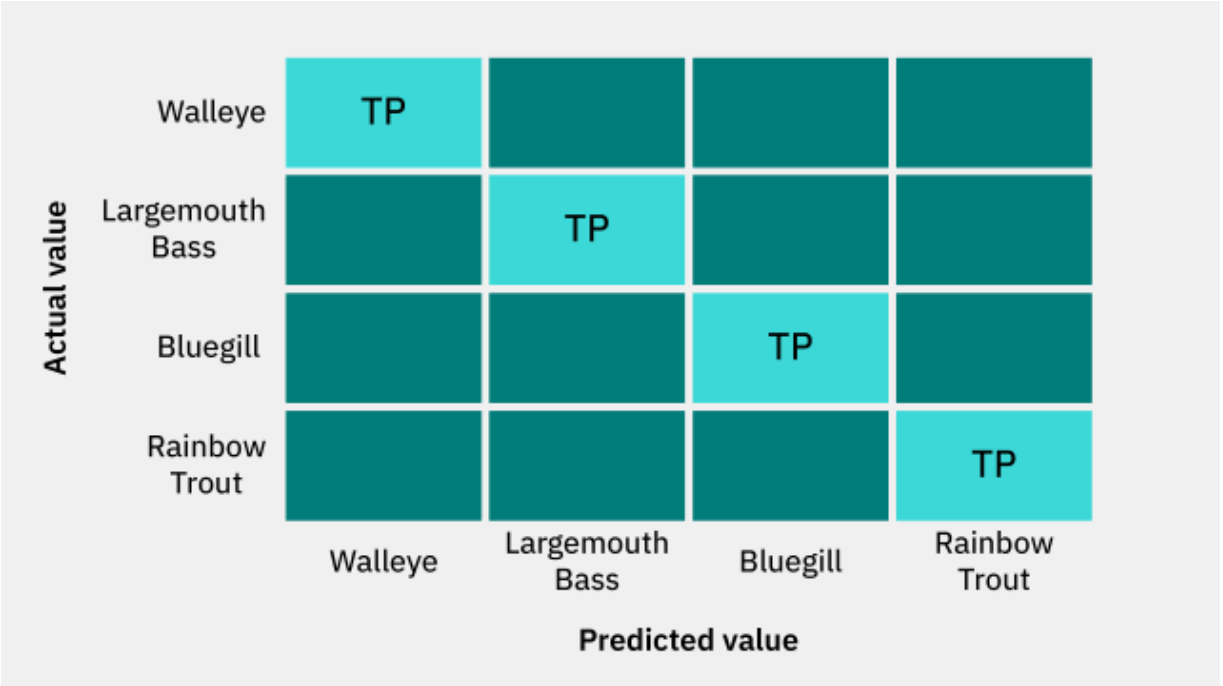
Comparativa clasificadores NaiveBayes



Adrián Yared Armas de la Nuez

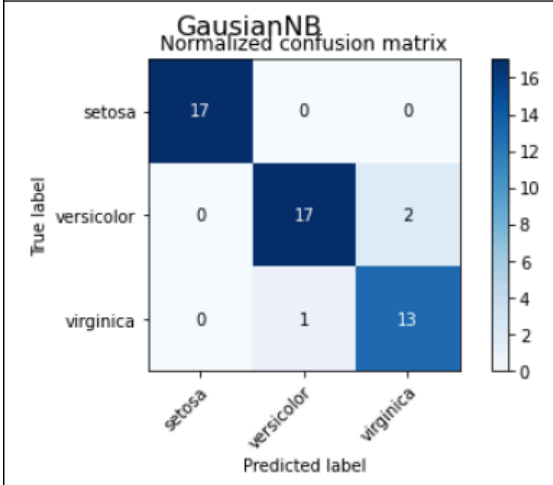
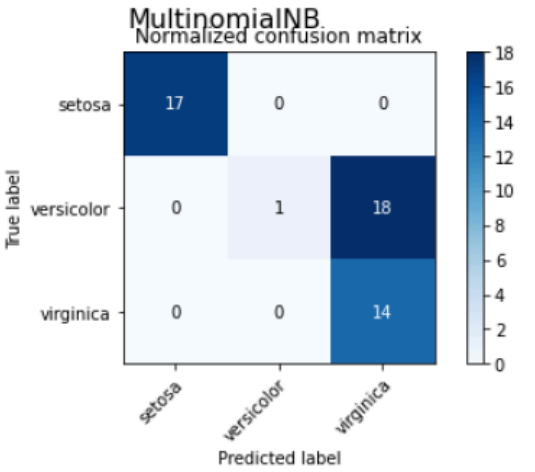
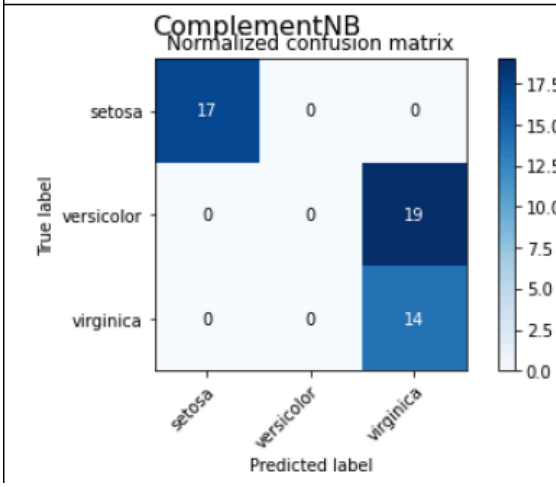
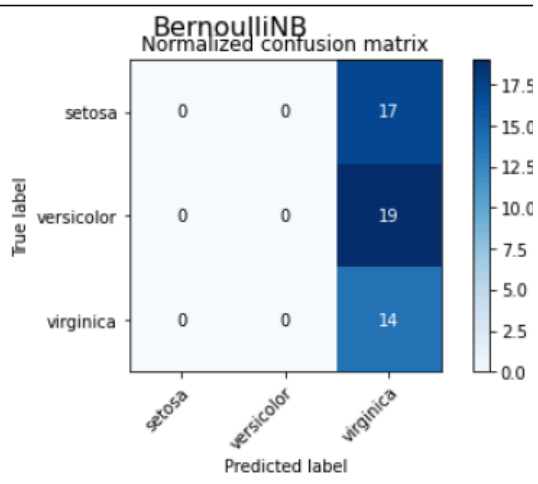
Contenido

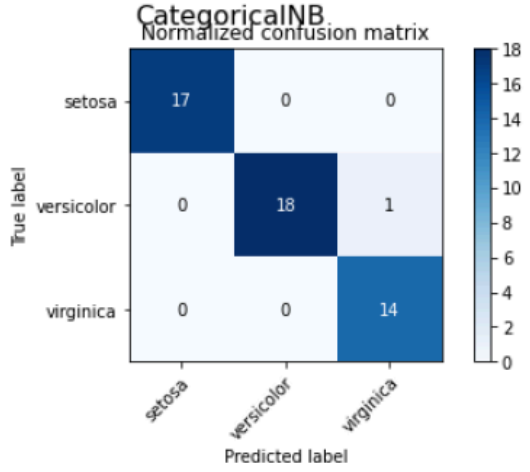
1. Enunciado

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:

GaussianNB	MultinomialNB
Precisión: 0.94	**Precisión: 0.64**
	
ComplementNB	**BernoulliNB**
Precisión: 0.62	**Precisión: 0.28**
	

CategoricalNB	
Precisión: 0,98	
	

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()
```

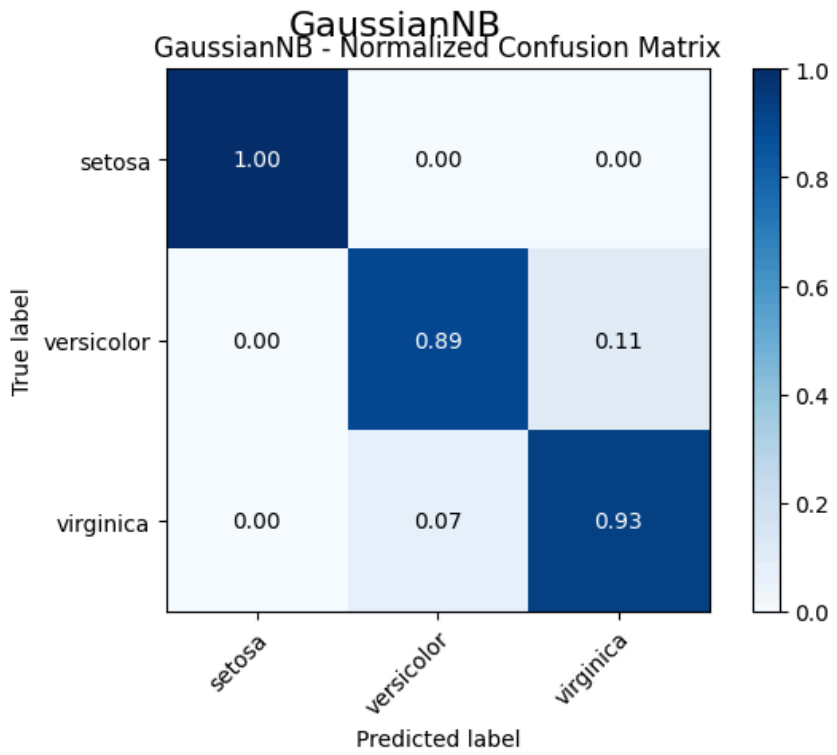
```
        # Plot normalized confusion matrix
        plot_confusion_matrix(
            ytest_num, y_pred_num,
            classes=clases_iris,
            normalize=True,
            title=f'{name} - Normalized Confusion Matrix',
            titleSup=name
        )
    except Exception as e:
        print(f"{name} encountered an error: {e}")
```

2.1.6 Resumen de los resultados

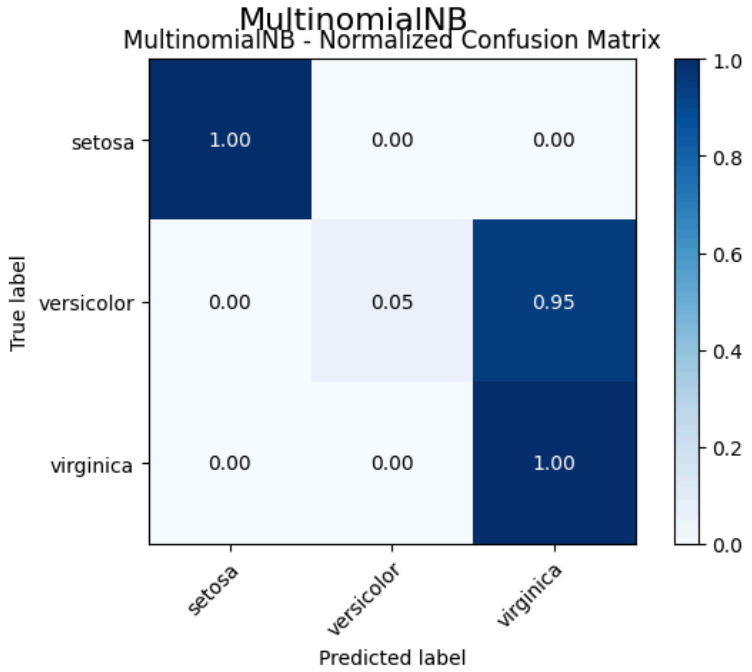
```
# Summary of results
for name, result in results.items():
    print(f"{name}: Accuracy = {result['accuracy']:.2f}")
```

2.1.7 Resultados

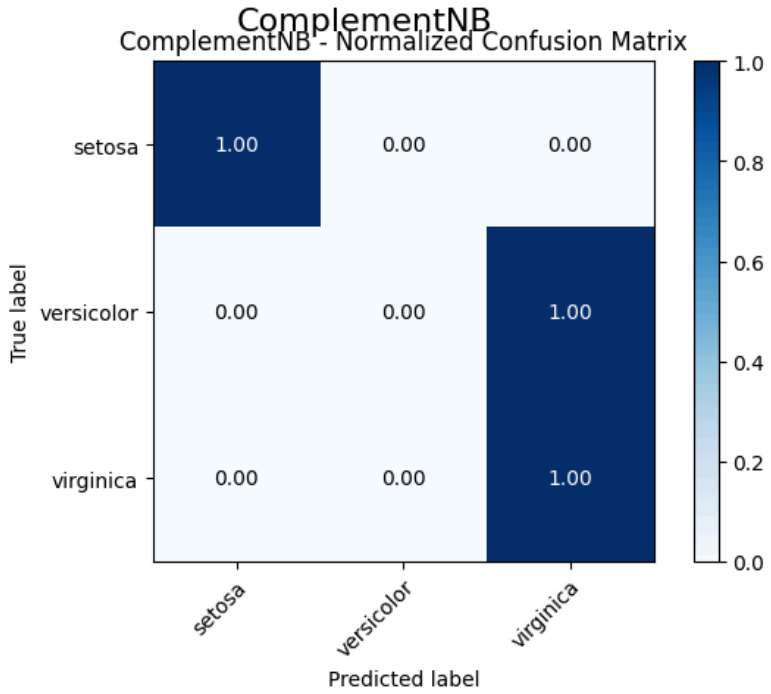
2.1.7.1 GaussianNB



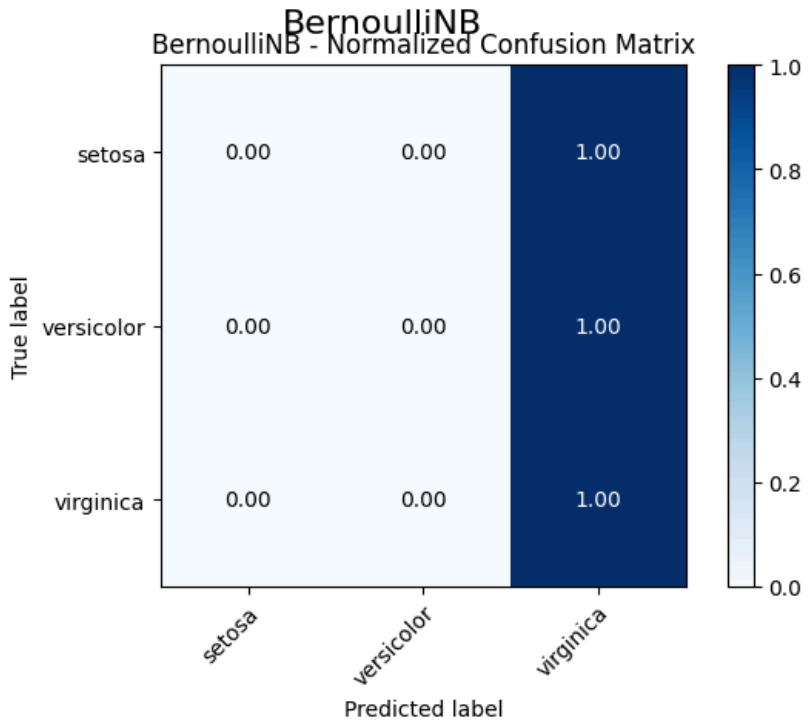
2.1.7.2 MultinomialNB



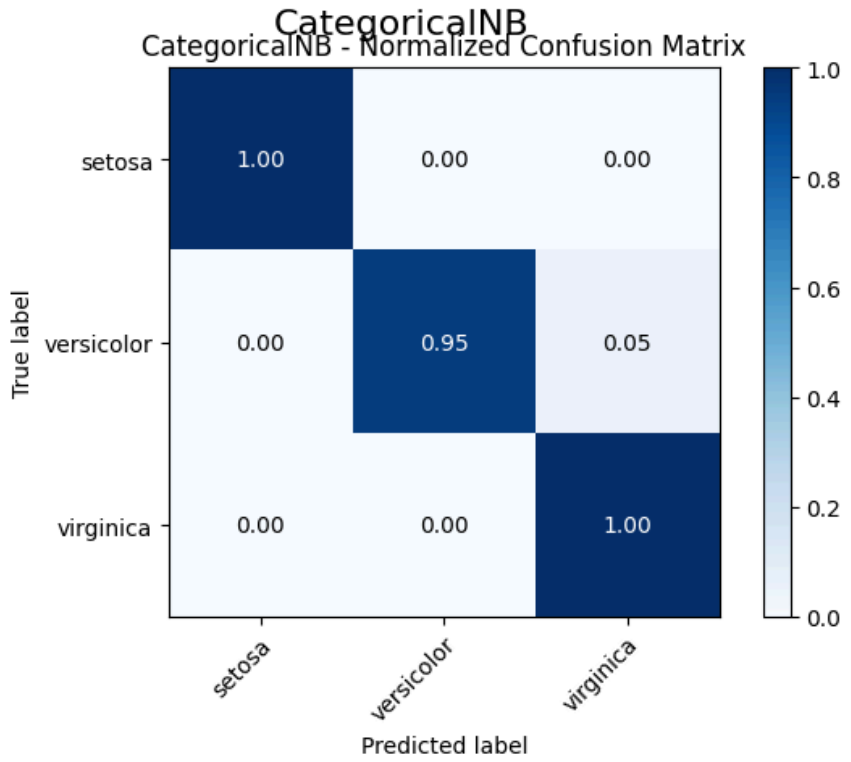
2.1.7.3 ComplementNB



2.1.7.4 BernoulliNB



2.1.7.5 CategoricalNB



2.1.7.6 Resumen de los resultados

```
GaussianNB: Accuracy = 0.94
MultinomialNB: Accuracy = 0.64
ComplementNB: Accuracy = 0.62
BernoulliNB: Accuracy = 0.28
CategoricalNB: Accuracy = 0.98
```

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
```

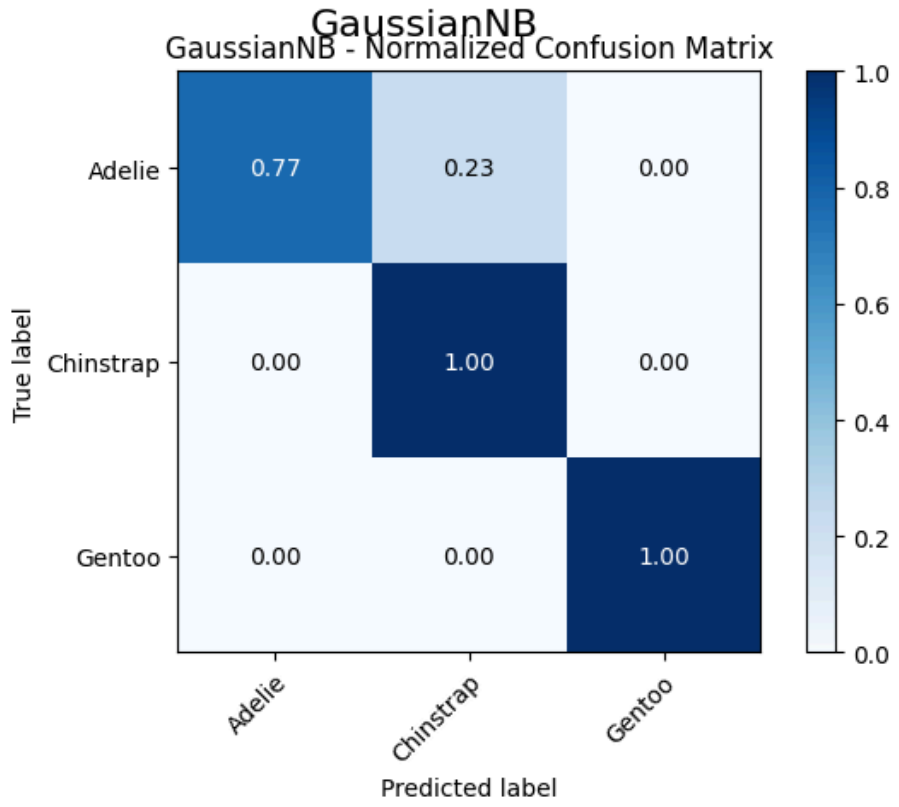
```
        )
    except Exception as e:
        print(f"{name} encountered an error: {e}")
```

2.1.8 Resumen de los resultados

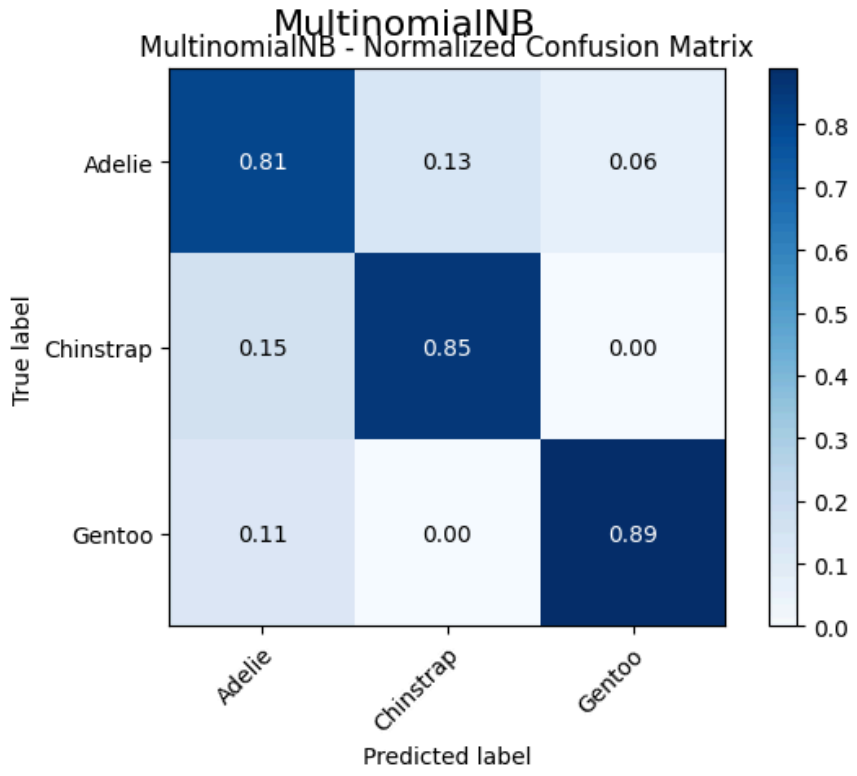
```
# Summary of results
for name, result in results.items():
    print(f"{name}: Accuracy = {result['accuracy']:.2f}")
```

2.1.9 Resultados

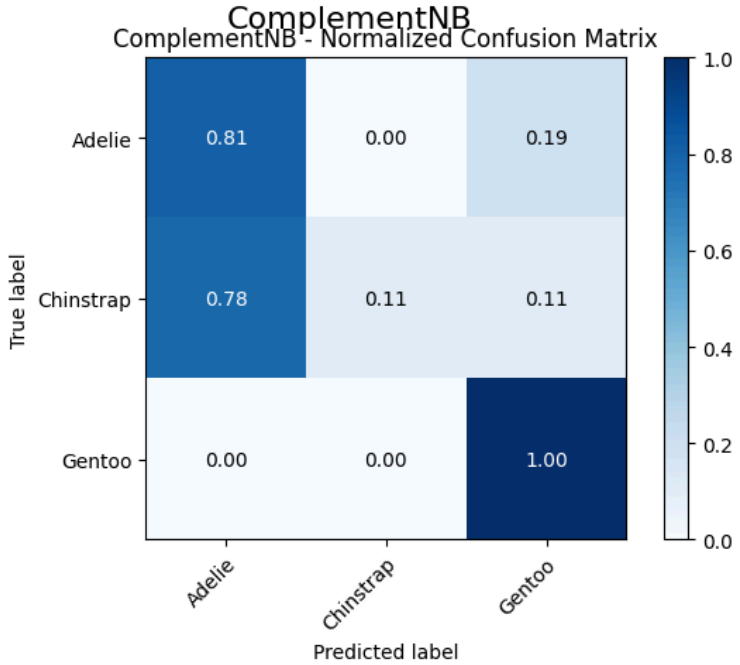
2.1.9.1 GaussianNB



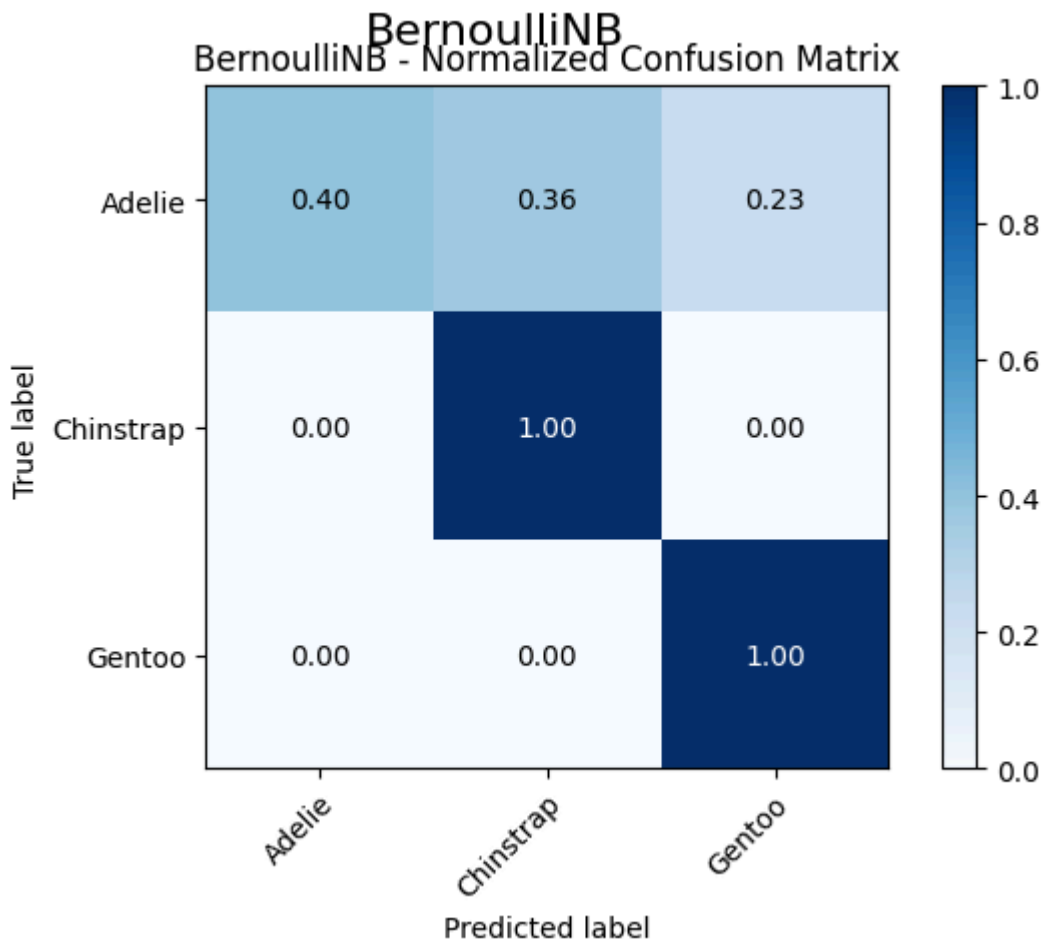
2.1.9.2 MultinomialNB



2.1.9.3 ComplementNB



2.1.9.4 BernoulliNB



2.1.9.5 Resumen del resultado

```
GaussianNB: Accuracy = 0.90
MultinomialNB: Accuracy = 0.85
ComplementNB: Accuracy = 0.70
BernoulliNB: Accuracy = 0.75
```

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