Tarea6- Penguins

April 19, 2023

WILLIAM ANDRÉS GÓMEZ ROA

BIOINGENIERÍA - CIENCIA DE DATOS

1 CLASIFICACIÓN MULTICLASE: ‘PENGUIN DATASET’

1.1 LR-LINEAL , LR-POLINOMIAL, KNN , SVM

Instructions

Usando el conjunto de datos: penguin dataset : The new Iris | Kaggle, use las características de:

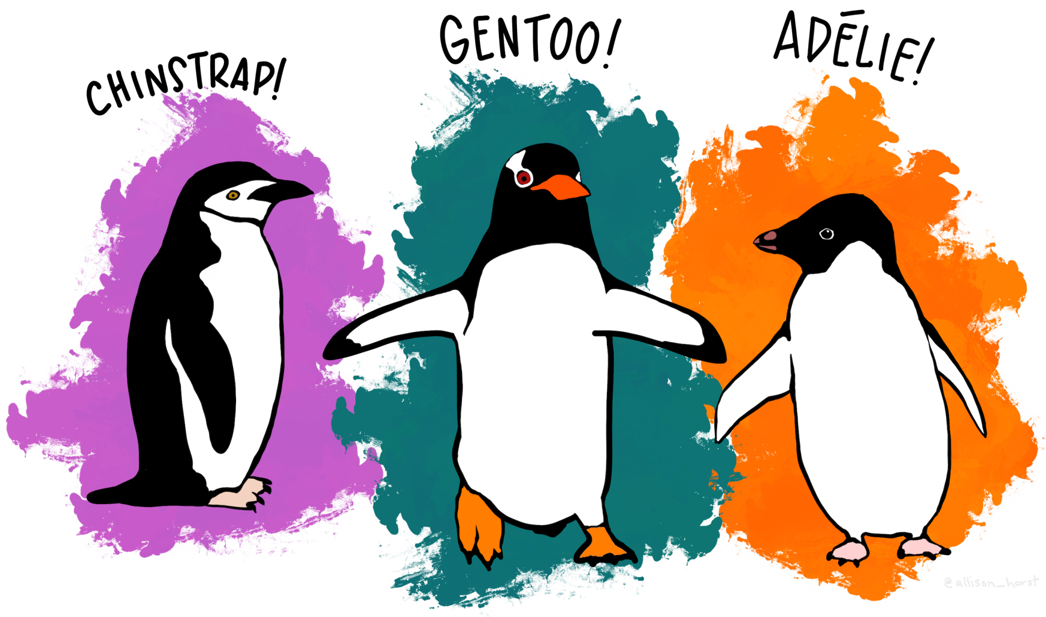
bill_length_mm bill_depth_mm flipper_length_mm body_mass_g etiqueta de especie: Adelie Chinstrap y Gentoo

Limpie el conjunto de datos usando algún método que encuentre de los muchos disponibles. en su preferencia use “pandas” para realizar esta labor.

Comprare y encuentre el mejor clasificador que usted considere conveniente de los vistos en clase. Sustente sus resultados.

(LR-lineal, LR-polinomial, KNN, y SVM)

Cargue los datos, use una partición por defecto entre validación y entrenamiento, y use las métricas vistas en clase para seleccionar el mejor modelo en su conjunto de validaciòn, . Concluya.



A continuación abriremos el archivo '.csv' que contiene los datos, con la funcion de pandas info() visualizaremos todas las columnas; crearemos un nuevo DataFrame que contenga la columna etiqueta y sus 4 caracteristicas en este. A partir de este nuevo dataframe trabajeremos.

```
[1]: import pandas as pd
     import numpy as np
```

```
[2]: data= pd.read_csv('penguins_size.csv')
```

```
[3]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   species            344 non-null    object
 1   island             344 non-null    object
 2   culmen_length_mm   342 non-null    float64
 3   culmen_depth_mm    342 non-null    float64
 4   flipper_length_mm  342 non-null    float64
 5   body_mass_g        342 non-null    float64
 6   sex                334 non-null    object
dtypes: float64(4), object(3)
memory usage: 18.9+ KB
```

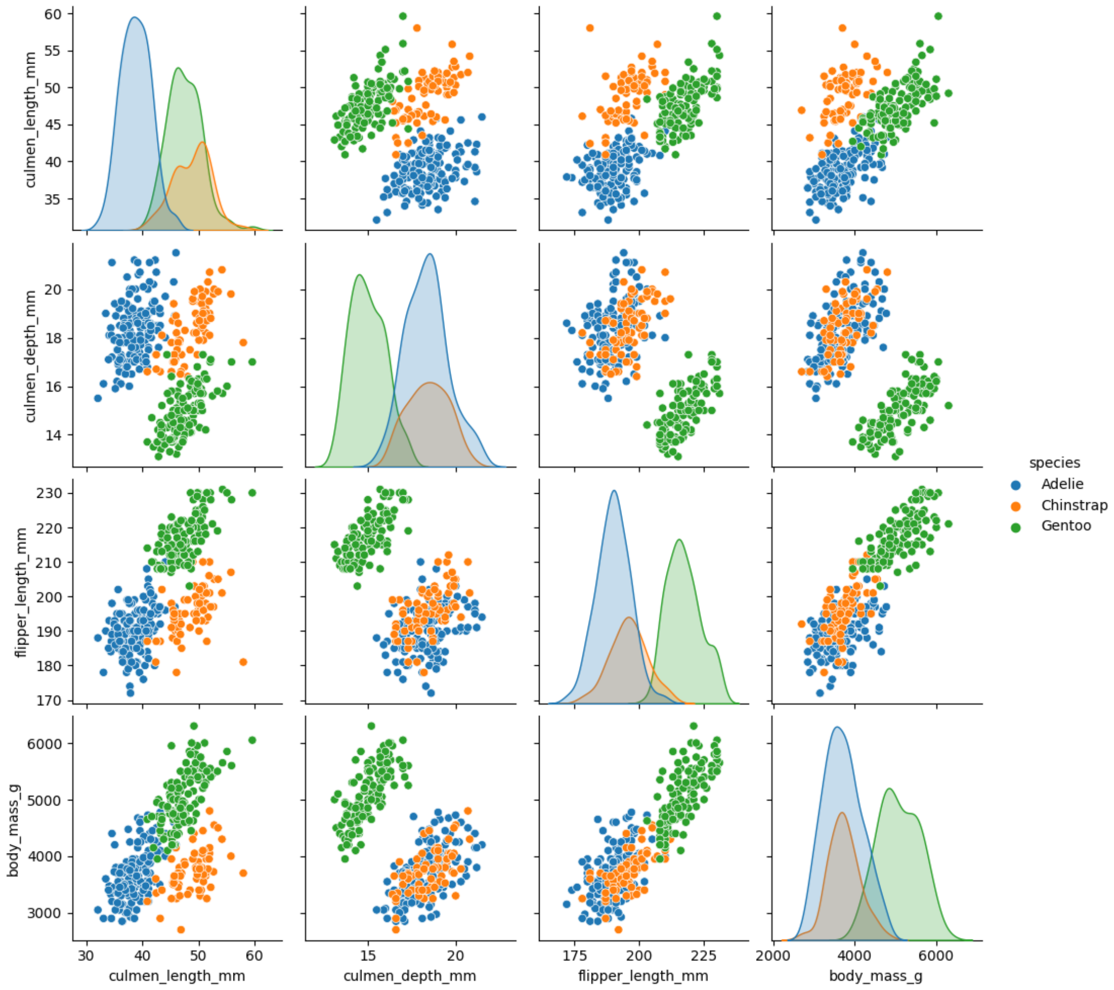
```
[4]: DATA=data[['species','culmen_length_mm', 'culmen_depth_mm','flipper_length_mm',␣
      ↪'body_mass_g']]
```

A continuación realizaremos una exploración visual de los datos utilizando la libreria de seaborn

[5]:
```
import seaborn as sns
```

[6]:
```
sns.pairplot(DATA, hue='species', diag_kind='kde', kind='scatter')
```

[6]:
```
<seaborn.axisgrid.PairGrid at 0x7f09cfb599d0>
```



LIMPIEZA DE LOS DATOS:

[7]:
```
DATA.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   species            344 non-null    object
 1   culmen_length_mm   342 non-null    float64
 2   culmen_depth_mm    342 non-null    float64
 3   flipper_length_mm  342 non-null    float64
 4   body_mass_g        342 non-null    float64
dtypes: float64(4), object(1)
memory usage: 13.6+ KB
```

Si nos fijamos tenemos 2 entradas nulas

[8]:
```
print('Entradas nulas:')
print(DATA.isnull().sum())
print('\nEntradas NaN:')
print(DATA.isna().sum())
print('\nEntradas duplicadas:')
print(DATA.duplicated().sum())

DATA = DATA.dropna()

print('\nEntradas nulas:')
print(DATA.isnull().sum())
```

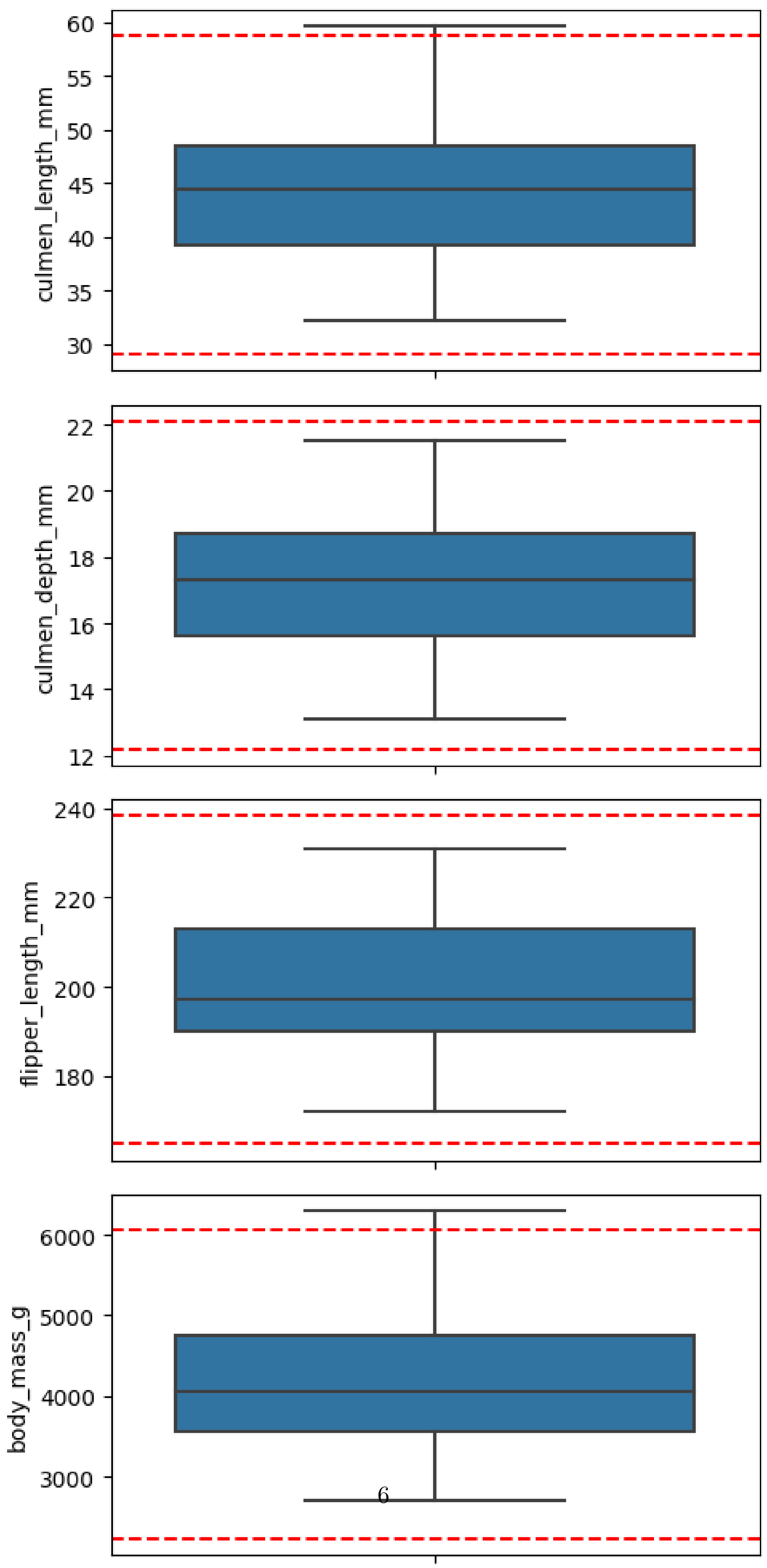
```
Entradas nulas:
species              0
culmen_length_mm     2
culmen_depth_mm      2
flipper_length_mm    2
body_mass_g          2
dtype: int64

Entradas NaN:
species              0
culmen_length_mm     2
culmen_depth_mm      2
flipper_length_mm    2
body_mass_g          2
dtype: int64

Entradas duplicadas:
0

Entradas nulas:
species              0
culmen_length_mm     0
culmen_depth_mm      0
flipper_length_mm    0
body_mass_g          0
dtype: int64
```

A continuación buscaremos datos atípicos variando el threshold que determina las unidades por debajo del Queartil 1 y por encima del Quartil 3 que nos determinan si son valores ‘outlayers’ o no. En este caso utilizaremos Threshold= 1.1 para detercatr algunos ‘atípicos’ y eliminarlos.

[9]:
```python
X = ['culmen_length_mm', 'culmen_depth_mm', 'flipper_length_mm', 'body_mass_g']

# Calcular el rango intercuartílico para cada caracteristica
Q1 = DATA[X].quantile(0.25)
Q3 = DATA[X].quantile(0.75)
IQR = Q3 - Q1
# Definir el threshold
outlier_threshold = 1.1
# Identificar los outlayers de cada columna
outliers = ((DATA[X] < (Q1 - outlier_threshold * IQR)) | (DATA[X] > (Q3 +␣
↪outlier_threshold * IQR))).any(axis=1)

print('Número de atípicos encontrados: '+ str(outliers.sum()))
```

```
Número de atípicos encontrados: 2
```

[10]:
```python
import matplotlib.pyplot as plt
lower_bound = Q1 - outlier_threshold * IQR
upper_bound = Q3 + outlier_threshold * IQR

fig, axes = plt.subplots(nrows=len(X), ncols=1, figsize=(5, 10))
for i, col in enumerate(X):
    sns.boxplot(y=DATA[col], ax=axes[i], orient='v')
    axes[i].axhline(y=lower_bound[col], color='r', linestyle='--')
    axes[i].axhline(y=upper_bound[col], color='r', linestyle='--')
    axes[i].set_ylabel(col)

plt.tight_layout()
```


culmen_length_mm
60
55
50
45
40
35
30
culmen_depth_mm
22
20
18
16
14
12
flipper_length_mm
240
220
200
180
body_mass_g
6000
5000
4000
3000

```
[11]: #Quitamos los 'outlayers'
      DATA = DATA[~outliers]
```

```
[12]: DATA.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 340 entries, 0 to 343
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   species            340 non-null    object
 1   culmen_length_mm   340 non-null    float64
 2   culmen_depth_mm    340 non-null    float64
 3   flipper_length_mm  340 non-null    float64
 4   body_mass_g        340 non-null    float64
dtypes: float64(4), object(1)
memory usage: 15.9+ KB
```

Hemos quitado filas con datos vacios y eliminado datos atípicos.

1.2 MODELOS:

```
[13]: from sklearn.model_selection import train_test_split

      X= DATA[['culmen_length_mm', 'culmen_depth_mm','flipper_length_mm',␣
       ↪'body_mass_g']]
      Y=DATA['species']

      X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,␣
       ↪random_state=42)
```

Las librerias de metricas que utilizaremos y el método GridSearchCV que hace validacion cruzada para buscar los mejores parámetros.

```
[14]: from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
       ↪f1_score, confusion_matrix, matthews_corrcoef
      from sklearn.model_selection import GridSearchCV
```

1.2.1 REGRESIÓN LINEAL:

```
[15]: from sklearn.linear_model import LinearRegression
      from sklearn.linear_model import LogisticRegression

      lr_linear = LogisticRegression(max_iter=1000, C=0.1)
      lr_linear.fit(X_train, y_train)
      y_pred = lr_linear.predict(X_test)
```

```
# Métricas
print('Logistic regression (linear):')
print('Accuracy:', accuracy_score(y_test, y_pred))
print('Precision:', precision_score(y_test, y_pred, average='weighted'))
print('Recall:', recall_score(y_test, y_pred, average='weighted'))
print('F1 score:', f1_score(y_test, y_pred, average='weighted'))
print('Confusion matrix:')
print(confusion_matrix(y_test, y_pred))
print('MCC: ',matthews_corrcoef(y_test, y_pred))
```

```
Logistic regression (linear):
Accuracy: 0.9852941176470589
Precision: 0.985726643598616
Recall: 0.9852941176470589
F1 score: 0.9850841699431233
Confusion matrix:
[[33  0  0]
 [ 1 11  0]
 [ 0  0 23]]
MCC:  0.9764377481501806
```

1.2.2 REGRESIÓN POLINOMIAL:

[16]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

poly = PolynomialFeatures(degree=3)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)

lr_poly = LogisticRegression(max_iter=1000, C=00.1)
lr_poly.fit(X_train_poly, y_train)
y_pred = lr_poly.predict(X_test_poly)

# Métricas
print('Logistic regression (Polynomial):')
print('Accuracy:', accuracy_score(y_test, y_pred))
print('Precision:', precision_score(y_test, y_pred, average='weighted'))
print('Recall:', recall_score(y_test, y_pred, average='weighted'))
print('F1 score:', f1_score(y_test, y_pred, average='weighted'))
print('Confusion matrix:')
print(confusion_matrix(y_test, y_pred))
print('MCC: ',matthews_corrcoef(y_test, y_pred))
```

```
Logistic regression (Polynomial):
Accuracy: 0.9705882352941176
Precision: 0.9747899159663864
```

```
Recall: 0.9705882352941176
F1 score: 0.9710510078157137
Confusion matrix:
[[33  0  0]
 [ 0 12  0]
 [ 0  2 21]]
MCC:  0.9541853032803713
```

1.2.3 KNN:

[17]:
```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()

param_grid = {'n_neighbors': [1, 2, 3,4, 5,6, 7,8 ,9,10, 11]}
grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)
n=grid_search.best_params_['n_neighbors']
print('K-Nearest Neighbors(KNN):')
print('Número de vecinos cercanos N: ', str(n))

knn = KNeighborsClassifier(n_neighbors=n)

knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

# Métricas

print('Accuracy:', accuracy_score(y_test, y_pred))
print('Precision:', precision_score(y_test, y_pred, average='weighted'))
print('Recall:', recall_score(y_test, y_pred, average='weighted'))
print('F1 score:', f1_score(y_test, y_pred, average='weighted'))
print('Confusion matrix:')
print(confusion_matrix(y_test, y_pred))
print('MCC: ',matthews_corrcoef(y_test, y_pred))
```

```
K-Nearest Neighbors(KNN):
Número de vecinos cercanos N:  1
Accuracy: 0.8823529411764706
Precision: 0.878954991087344
Recall: 0.8823529411764706
F1 score: 0.8802715350710127
Confusion matrix:
[[29  3  1]
 [ 4  8  0]
 [ 0  0 23]]
MCC:  0.8094808596959331
```

1.2.4 SVM:

[18]:
```
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler

param_grid = {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf'], 'gamma':␣
  ↪['scale', 'auto']}
svm = SVC()
svm_grid = GridSearchCV(svm, param_grid, cv=5)
svm_grid.fit(X_train, y_train)

KERNEL=svm_grid.best_params_['kernel']
GAMMA=svm_grid.best_params_['gamma']

print('Support Vector Machines (SVC):')
print('Mejor Kernel: ', str(KERNEL))
print('Mejor Gamma: ', str(GAMMA))

model = SVC(kernel=KERNEL, gamma=GAMMA)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_train)
model.fit(X_scaled, y_train)


X_new_scaled = scaler.transform(X_test)
y_pred = model.predict(X_new_scaled)


print('Accuracy:', accuracy_score(y_test, y_pred))
print('Precision:', precision_score(y_test, y_pred, average='weighted'))
print('Recall:', recall_score(y_test, y_pred, average='weighted'))
print('F1 score:', f1_score(y_test, y_pred, average='weighted'))
print('Confusion matrix:')
print(confusion_matrix(y_test, y_pred))
print('MCC: ',matthews_corrcoef(y_test, y_pred))
```

```
Support Vector Machines (SVC):
Mejor Kernel:  linear
Mejor Gamma:  scale
Accuracy: 0.9705882352941176
Precision: 0.9705882352941176
Recall: 0.9705882352941176
F1 score: 0.9705882352941176
Confusion matrix:
[[32  1  0]
 [ 1 11  0]
 [ 0  0 23]]
MCC:  0.9524807826694619
```

1.3 CONCLUSIÓN:

En base a el análisis realizado anteriormente con los datos de Penguin y utilizando cuatro modelos de clasificación diferentes (regresión logística con kernels lineales y polinómicos, K-vecinos más cercanos y máquinas de soporte vectorial), podemos concluir que todos los modelos funcionaron relativamente bien.

La regresión logística con kernel lineal obtuvo la exactitud más alta de todos con 0,98, seguida por el kernel polinomial con una exactitud de 0,97. K-Nearest Neighbors tuvo una exactitud ligeramente menor de 0,88, y Support Vector Machines tuvo una exactitud de 0,97.

Utilizamos la búsqueda en cuadrícula con validación cruzada para optimizar los hiperparámetros para KNN y SVM. Los mejores hiperparámetros para SVM fueron un kernel lineal y un valor gamma de "escala". Para KNN, el mejor valor para el número de vecinos fue 1.

También obtuvimos otras metricas como el coeficiente de correlación de Matthews y el modelo de regresión logistica con kernel lineal nuevamente es el mejor.

En general, los modelos de regresión logística se desempeñaron mejor en este conjunto de datos.