



Tecnológico de Monterrey

**Instituto Tecnológico y de Estudios Superiores de Monterrey,
Campus Monterrey**

Escuela de ingeniería y ciencias

**Inteligencia Artificial Avanzada para la Ciencia de Datos I
(Grupo 101)**

**Momento de Retroalimentación: Módulo 2 Análisis y Reporte sobre
el Desempeño del Modelo (Portafolio Análisis)**

Alumnos:

Amy Murakami Tsutsumi

A01750185

Profesor:

Iván Mauricio Amaya Contreras

Fecha:

09/09/2022

Nombre del archivo a revisar

MomRetroM2Análisis.py
MomentoRetroM2Análisis.pdf

Librería

La librería principal utilizada fue `sklearn.neural_network- MLPClassifier` para poder implementar el algoritmo de redes neuronales.

Otras librerías que se utilizarán son:

- `pandas` : para la creación y operaciones de dataframes.
- `numpy` : para la creación de vectores y matrices.
- `sklearn.preprocessing - StandardScaler`: para el escalamiento de datos.
- `sklearn.model_selection - train_test_split` : para la división de los datos en subconjuntos de entrenamiento y prueba.
- `sklearn.metrics - confusion_matrix & classification_report` : para la visualización de desempeño del algoritmo y las métricas de clasificación.
- `matplotlib.pyplot`: para la generación de gráficos.
- `seaborn` : basada en `matplotlib` para la graficación de datos estadísticos.
- `mlxtend.plotting - plot_learning_curves`: para la graficación de las learning curves.

Dataset

Para este portafolio de implementación se utilizará el dataset de "Heart Failure Prediction Dataset" que se obtuvo de la siguiente liga:

<https://www.kaggle.com/datasets/fedesoriano/heart-failure-prediction>

Este dataset contiene información con atributos que se utilizarán para determinar y predecir si una persona es propensa a tener un ataque cardíaco. Estos atributos son:

1. Age: edad del paciente
2. Sex: sexo del paciente [M: Hombre, F: Mujer]
3. ChestPainType: tipo de dolor en el pecho [TA: Angina típica, ATA: Angina atípica, NAP: Dolor no anginoso, ASY: Asintomático]
4. RestingBP: presión arterial en reposo [mm Hg]
5. Cholesterol: colesterol sérico [mm/dl]
6. FastingBS: glucemia en ayunas [1: si FastingBS > 120 mg/dl, 0: en caso contrario]
7. RestingECG: resultados del electrocardiograma [Normal: Normal, ST: con anomalía de la onda ST-T (inversiones de la onda T y/o elevación o depresión del ST de > 0,05 mV), HVI: que muestra hipertrofia ventricular izquierda probable o definitiva según los criterios de Estes].
8. MaxHR: frecuencia cardíaca máxima alcanzada [Valor numérico entre 60 y 202]

9. ExerciseAngina: angina inducida por el ejercicio [Y: Sí, N: No]
10. Oldpeak: oldpeak = ST [Valor numérico medido en depresión]
11. ST_Slope: la pendiente del segmento ST máximo del ejercicio [Up: pendiente ascendente, Flat: plano, Down: pendiente descendente]
12. HeartDisease: clase de salida [1: enfermedad cardiaca, 0: Normal]

Este dataset cuenta con 918 registros de pacientes.

Separación y evaluación del modelo con un conjunto de prueba y un conjunto de validación (Train/Test/Validation)

Ahora se separará un conjunto de datos para realizar la validación. Se utilizó el dataset de entrenamiento y se dividió para que el 20% se utilizara para validación. Por lo tanto, los datos se dividen en 60% para entrenamiento, 20% para validarlo y 20% para probarlo.

▼ Escalamiento de datos

Ahora se realizará el escalamiento de datos para que los modelos puedan encontrar fácilmente convergencia entre los datos. Esto se realiza con fin de optimizar el método.

```
[12] # Separar el dataframe
df_x = dfHeart.drop(['HeartDisease'], axis=1)
df_y = dfHeart['HeartDisease']

[57] # Escalamiento
escalador_fill = StandardScaler()
escalador_fill.fit(df_x)
dt_x = pd.DataFrame(escalador_fill.transform(df_x))

[58] #Modularización del data-set
x_train, x_test, y_train, y_test = train_test_split(df_x, df_y, test_size = 0.20, random_state= 2)
x_train2, x_val, y_train2, y_val = train_test_split(x_train, y_train, test_size = 0.20, random_state= 2)
```

▼ Modelos de predicción

Ahora se realizarán 5 modelos diferentes de redes neuronales para encontrar el más eficiente.

```
4 s ▶ modelo1 = MLPClassifier(random_state = 1,
                           hidden_layer_sizes = (5),
                           activation = "relu",
                           verbose = False,
                           solver = "adam",
                           learning_rate = "adaptive",
                           max_iter = 10000)

modelo2 = MLPClassifier(random_state = 1,
                        hidden_layer_sizes = (10, 9),
                        activation = "relu",
                        verbose = False,
                        solver = "adam",
                        learning_rate = "adaptive",
                        max_iter = 10000)
```

✓
4 s



```
modelo3 = MLPClassifier(random_state = 1,
                        hidden_layer_sizes = (10, 8, 10),
                        activation = "relu",
                        verbose = False,
                        solver = "adam",
                        learning_rate = "adaptive",
                        max_iter = 10000)

modelo4 = MLPClassifier(random_state = 1,
                        hidden_layer_sizes = (10, 15, 10, 4),
                        activation = "relu",
                        verbose = False,
                        solver = "adam",
                        learning_rate = "adaptive",
                        max_iter = 10000)

modelo5 = MLPClassifier(random_state = 1,
                        hidden_layer_sizes = (10, 9, 9, 10),
                        activation = "relu",
                        verbose = False,
                        solver = "adam",
                        learning_rate = "adaptive",
                        max_iter = 10000)
```

✓
4 s



```
modelo1.fit(x_train, y_train)
modelo2.fit(x_train, y_train)
modelo3.fit(x_train, y_train)
modelo4.fit(x_train, y_train)
modelo5.fit(x_train, y_train)

print("Training score modelo 1: ", modelo1.score(x_train,y_train))
print("Validation score modelo 1: ", modelo1.score(x_val, y_val))
print("Test score modelo 1: ", modelo1.score(x_test, y_test))

print("\nTraining score modelo 2: ", modelo2.score(x_train,y_train))
print("Validation score modelo 2: ", modelo2.score(x_val, y_val))
print("Test score modelo 2: ", modelo2.score(x_test, y_test))

print("\nTraining score modelo 3: ", modelo3.score(x_train,y_train))
print("Validation score modelo 3: ", modelo3.score(x_val, y_val))
print("Test score modelo 3: ", modelo3.score(x_test, y_test))

print("\nTraining score modelo 4: ", modelo4.score(x_train,y_train))
print("Validation score modelo 4: ", modelo4.score(x_val, y_val))
print("Test score modelo 4: ", modelo4.score(x_test, y_test))

print("\nTraining score modelo 5: ", modelo5.score(x_train,y_train))
print("Validation score modelo 5: ", modelo5.score(x_val, y_val))
print("Test score modelo 5: ", modelo5.score(x_test, y_test))
```

```
Training score modelo 1: 0.8746594005449592
Validation score modelo 1: 0.9387755102040817
Test score modelo 1: 0.8097826086956522

Training score modelo 2: 0.8623978201634878
Validation score modelo 2: 0.9251700680272109
Test score modelo 2: 0.842391304347826

Training score modelo 3: 0.8746594005449592
Validation score modelo 3: 0.9319727891156463
Test score modelo 3: 0.8206521739130435

Training score modelo 4: 0.8583106267029973
Validation score modelo 4: 0.9251700680272109
Test score modelo 4: 0.842391304347826

Training score modelo 5: 0.8828337874659401
Validation score modelo 5: 0.9387755102040817
Test score modelo 5: 0.842391304347826
```

Podemos visualizar que los modelos 1, 3 y 5 son los más precisos con los datos de entrenamiento. Sin embargo, los modelos más precisos con los datos de validación son el 1 y 5. Además los modelos más preciso para los datos de prueba son el 2, 4 y 5.

▼ Predicciones

Ahora se realizarán las predicciones de los cinco modelos.

```
✓ [179] pred1 = modelo1.predict(x_test)
0s      pred2 = modelo2.predict(x_test)
      pred3 = modelo3.predict(x_test)
      pred4 = modelo4.predict(x_test)
      pred5 = modelo5.predict(x_test)

✓ [180] #Modelo 1
0s      print('Modelo 1: ')
      print(modelo1.predict_proba([df_x.loc[0]]))
      print('Heart Disease? Estimated: ', modelo1.predict([df_x.loc[0]]),
            'Real: ', df_y.loc[0])
      #Modelo 2
      print('\nModelo 2: ')
      print(modelo2.predict_proba([df_x.loc[0]]))
      print('Heart Disease? Estimated: ', modelo2.predict([df_x.loc[0]]),
            'Real: ', df_y.loc[0])
```

```

#Modelo 3
print('\nModelo 3: ')
print(modelo3.predict_proba([df_x.loc[0]]))
print('Heart Disease? Estimated: ', modelo3.predict([df_x.loc[0]]),
      'Real: ', df_y.loc[0])
#Modelo 4
print('\nModelo 4: ')
print(modelo4.predict_proba([df_x.loc[0]]))
print('Heart Disease? Estimated: ', modelo4.predict([df_x.loc[0]]),
      'Real: ', df_y.loc[0])
#Modelo 5
print('\nModelo 5: ')
print(modelo5.predict_proba([df_x.loc[0]]))
print('Heart Disease? Estimated: ', modelo5.predict([df_x.loc[0]]),
      'Real: ', df_y.loc[0])

```

```

Modelo 1:
[[0.96910751 0.03089249]]
Heart Disease? Estimated:  [0] Real:  0

```

```

Modelo 2:
[[0.93545654 0.06454346]]
Heart Disease? Estimated:  [0] Real:  0

```

```

Modelo 3:
[[0.97683472 0.02316528]]
Heart Disease? Estimated:  [0] Real:  0

```

```

Modelo 4:
[[0.91801577 0.08198423]]
Heart Disease? Estimated:  [0] Real:  0

```

```

Modelo 5:
[[0.96310434 0.03689566]]
Heart Disease? Estimated:  [0] Real:  0

```

Podemos observar que en todos los modelos se está realizando la predicción de manera correcta. Como primer valor muestra la probabilidad de que el HeartDisease tenga un valor de 0 y 1. Después se muestra el valor estimado con los modelos y el valor real. En todos los modelos coinciden que el valor estimado por cada modelo y el valor real es 0, por lo que se puede confirmar que son modelos efectivos para la predicción de enfermedades cardíacas.

Validación

Para la etapa de validación se utilizarán matrices de confusión y reportes de clasificación.

```
#Modelo 1
print('Modelo 1: ')
sns.set()
f, ax = plt.subplots()
matriz1 = confusion_matrix(y_test, pred1)
print(classification_report(y_test, pred1))
print('\nMatriz de confusión: ')
sns.heatmap(matriz1, annot=True, ax=ax, cbar=False, fmt='g', cmap='Dark2_r')
```

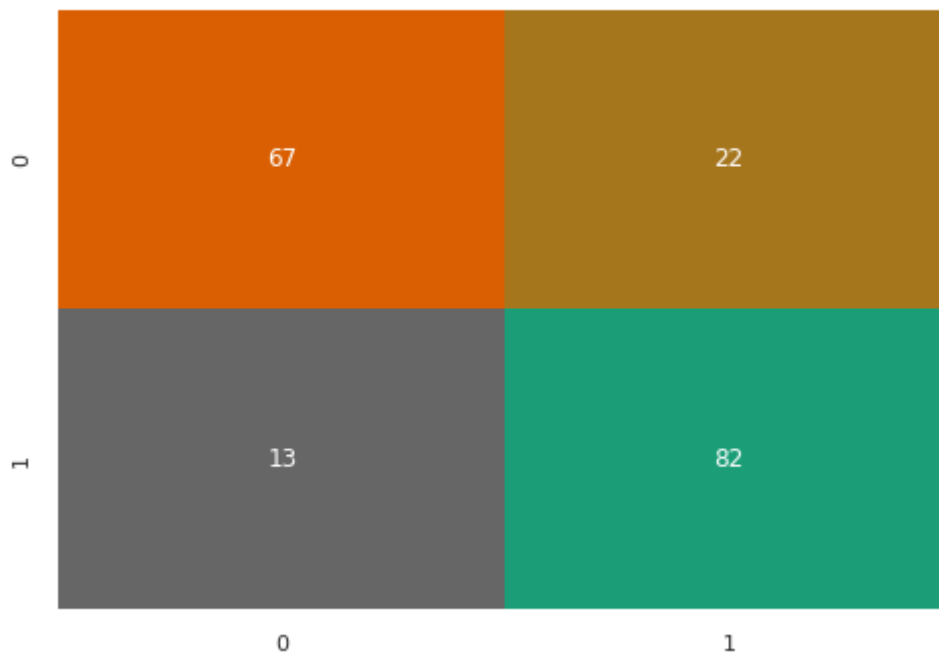
```
Modelo 1:
              precision    recall  f1-score   support

      0       0.84      0.75      0.79         89
      1       0.79      0.86      0.82         95

 accuracy      0.81
 macro avg     0.81
 weighted avg  0.81
```

Matriz de confusión:

<matplotlib.axes._subplots.AxesSubplot at 0x7fb79ba10810>



Para el primer modelo podemos observar que el valor de f1-score es 0.81 lo que indica que es un buen modelo de predicción. Por otro lado, en la matriz de confusión se tienen 67 valores true positive y 82 valores true negative; es decir, valores predichos que coinciden con el valor real. Además, se tienen 22 valores false positive y 13 false negative es decir valores erróneos.

```

▶ #Modelo 2
print('Modelo 2: ')
sns.set()
f, ax = plt.subplots()
matriz2 = confusion_matrix(y_test, pred2)
print(classification_report(y_test, pred2))
print('\nMatriz de confusión: ')
sns.heatmap(matriz2, annot=True, ax=ax, cbar=False, fmt='g', cmap='Paired')

```

```

Modelo 2:
              precision    recall  f1-score   support

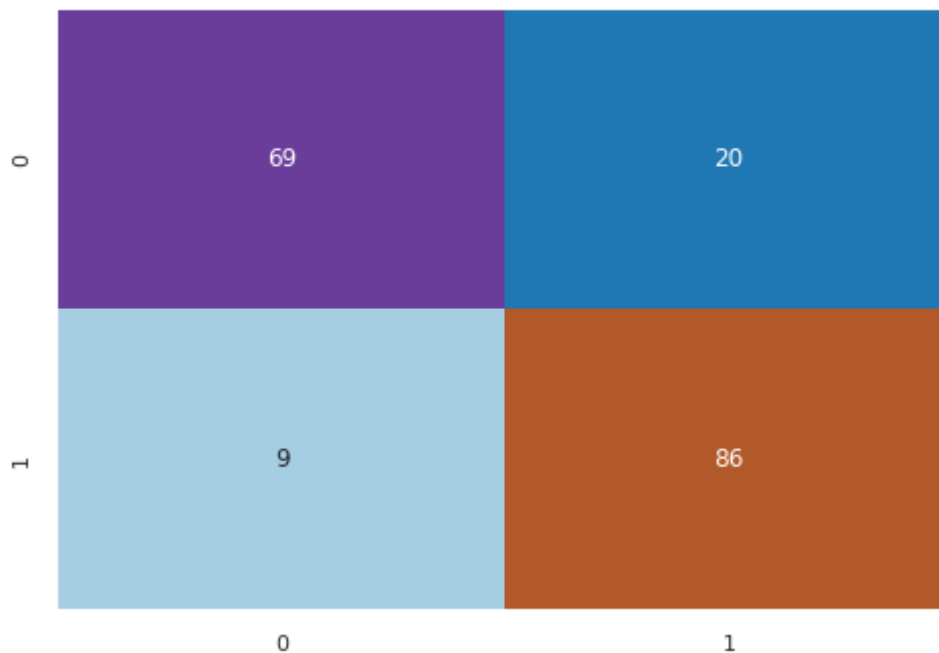
     0       0.88        0.78        0.83        89
     1       0.81        0.91        0.86        95

 accuracy          0.84          184
 macro avg          0.85          184
 weighted avg       0.85          184

```

Matriz de confusión:

<matplotlib.axes._subplots.AxesSubplot at 0x7fb79b9b6cd0>



Para el segundo modelo podemos observar que el valor de f1-score es 0.84 lo que indica que es un mejor modelo de predicción que el primero. Por otro lado, en la matriz de confusión se tienen 69 valores true positive y 86 valores true negative; es decir, valores predichos que coinciden con el valor real. Además, se tienen 20 valores false positive y 9 false negative es decir valores erróneos.



```
#Modelo 3
print('Modelo 3: ')
sns.set()
f, ax = plt.subplots()
matriz3 = confusion_matrix(y_test, pred3)
print(classification_report(y_test, pred3))
print('\nMatriz de confusión: ')
sns.heatmap(matriz3, annot=True, ax=ax, cbar=False, fmt='g', cmap='Pastel1')
```



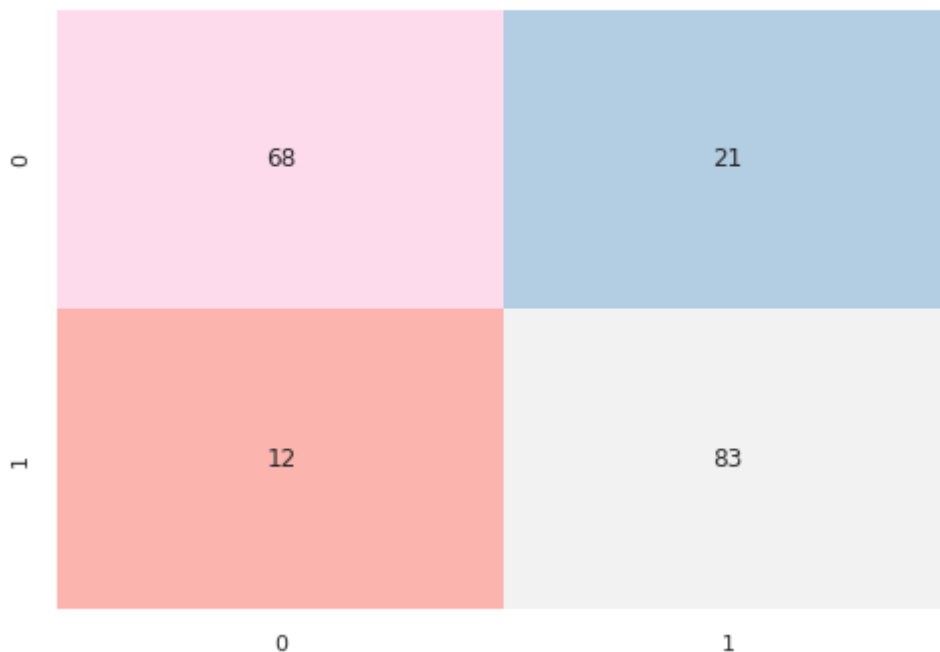
```
Modelo 3:
              precision    recall  f1-score   support

     0         0.85        0.76        0.80         89
     1         0.80        0.87        0.83         95

 accuracy          0.82
 macro avg         0.82
 weighted avg      0.82
```

Matriz de confusión:

<matplotlib.axes._subplots.AxesSubplot at 0x7fb79b950710>



Para el tercer modelo podemos observar que el valor de f1-score es 0.82 lo que indica que es un buen modelo, pero no tiene la misma precisión que el segundo. Por otro lado, en la matriz de confusión se tienen 68 valores true positive y 83 valores true negative; es decir, valores predichos que coinciden con el valor real. Además, se tienen 21 valores false positive y 12 false negative es decir valores erróneos.

```

▶ #Modelo 4
print('Modelo 4: ')
sns.set()
f, ax = plt.subplots()
matriz4 = confusion_matrix(y_test,pred4)
print(classification_report(y_test,pred4))
print('\nMatriz de confusión: ')
sns.heatmap(matriz4, annot=True, ax=ax, cbar=False, fmt='g', cmap='Pastel2')

```

```

↳ Modelo 4:
              precision    recall  f1-score   support

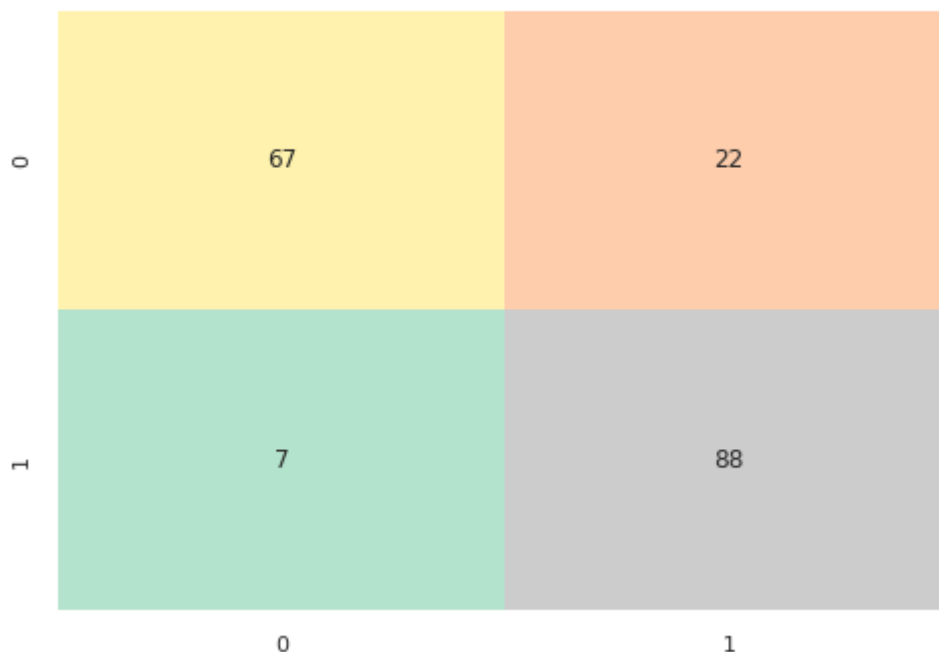
     0         0.91        0.75        0.82         89
     1         0.80        0.93        0.86         95

 accuracy          0.84         184
 macro avg         0.85         0.84         0.84         184
 weighted avg         0.85         0.84         0.84         184

```

Matriz de confusión:

<matplotlib.axes._subplots.AxesSubplot at 0x7fb79b8f4a10>



Para el cuarto modelo podemos observar que el valor de f1-score es 0.84 lo que indica que es de los mejores modelos hasta ahora. Por otro lado, en la matriz de confusión se tienen 67 valores true positive y 88 valores true negative; es decir, valores predichos que coinciden con el valor real. Además, se tienen 22 valores false positive y 7 false negative es decir valores erróneos.

```

▶ #Modelo 5
print('Modelo 5: ')
sns.set()
f, ax = plt.subplots()
matriz5 = confusion_matrix(y_test,pred5)
print(classification_report(y_test,pred5))
print('\nMatriz de confusión: ')
sns.heatmap(matriz5, annot=True, ax=ax, cbar=False, fmt='g', cmap='PuRd')

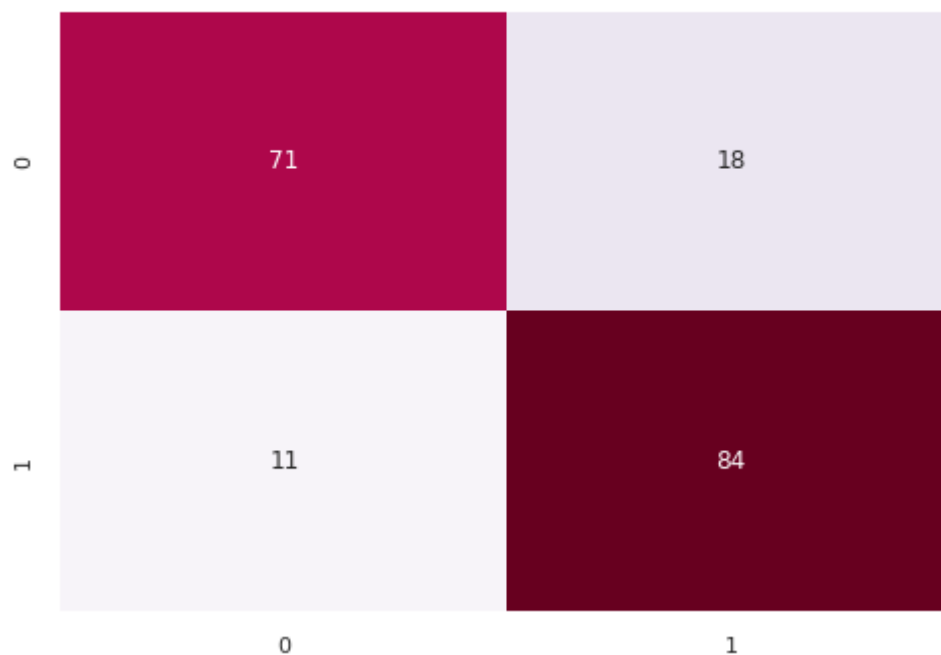
```

Modelo 5:

	precision	recall	f1-score	support
0	0.87	0.80	0.83	89
1	0.82	0.88	0.85	95
accuracy			0.84	184
macro avg	0.84	0.84	0.84	184
weighted avg	0.84	0.84	0.84	184

Matriz de confusión:

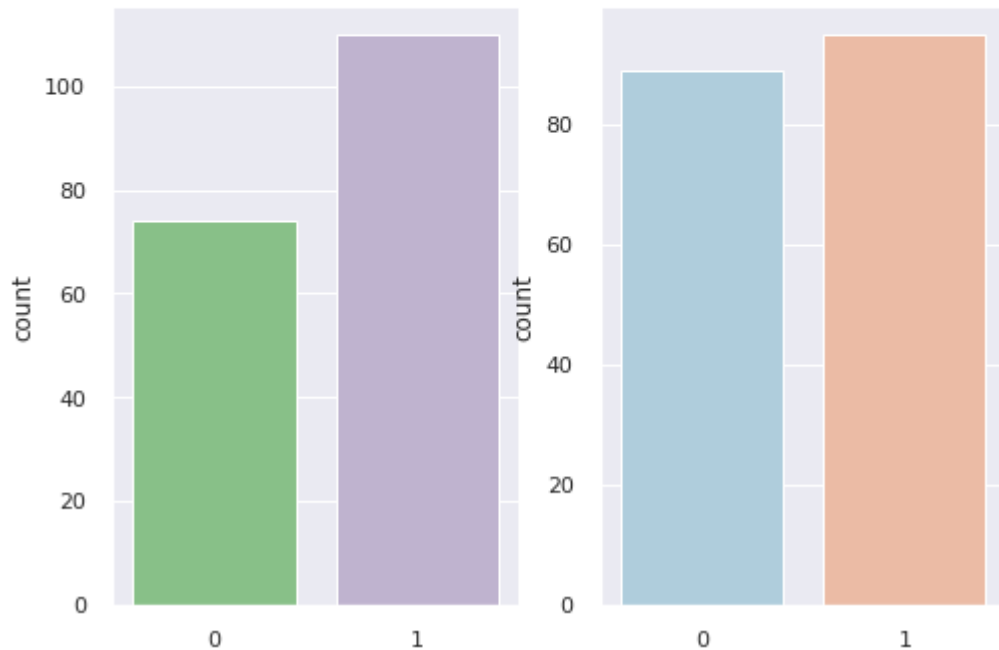
<matplotlib.axes._subplots.AxesSubplot at 0x7fb79b86a790>



Por último, para el quinto modelo podemos observar que el valor de f1-score es 0.84 lo que indica que es de los mejores modelos junto con el modelo 2 y 4. Por otro lado, en la matriz de confusión se tienen 71 valores true positive y 84 valores true negative; es decir, valores predichos que coinciden con el valor real. Además, se tienen 18 valores false positive y 11 false negative es decir valores erróneos.

```
fig, ax = plt.subplots(1,2)
sns.countplot(pred4, ax=ax[0], palette="Accent")
sns.countplot(y_test, ax=ax[1], palette="RdBu_r")
fig.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning
```



En la gráfica anterior se muestra del lado izquierdo los número de valores prededidos con el modelo cuatro y del lado derecho los valores reales. Se puede observar que los datos prededidos con valor 1 se acercan bastante a los valores reales. Sin embargo, hay una diferencia significativa en los valores igual a cero.

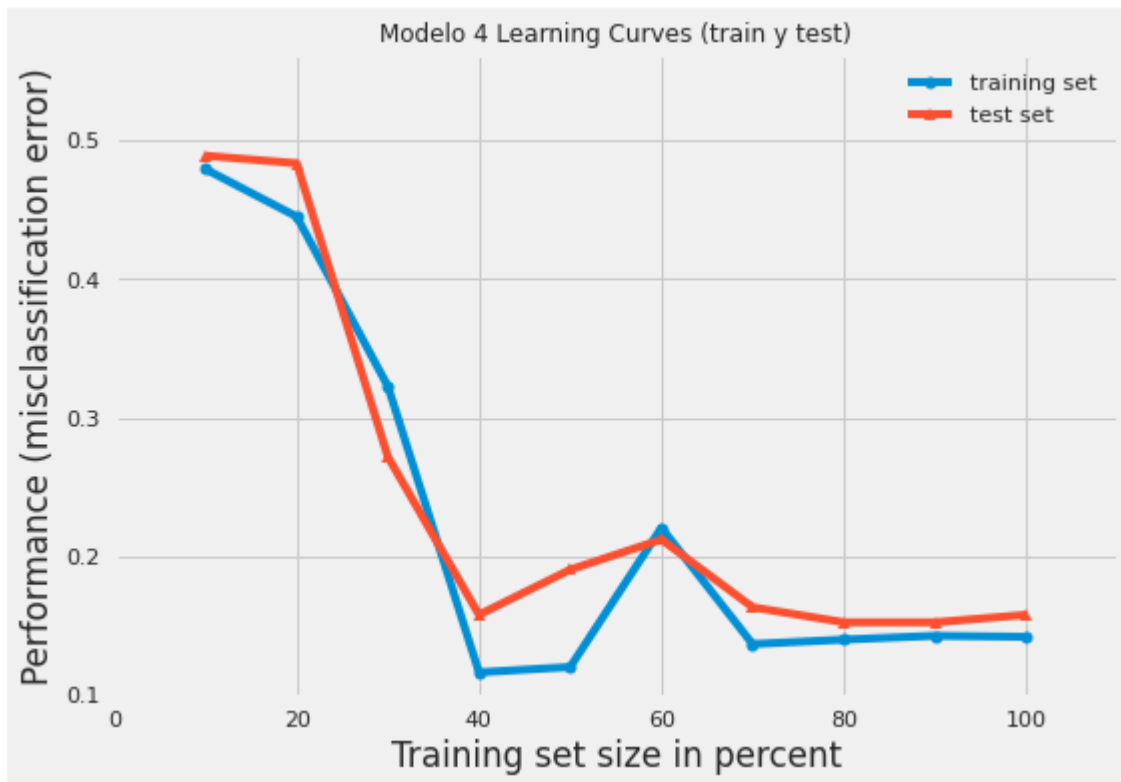
Diagnóstico y explicación del nivel de ajuste del modelo: underfitt fitt overfitt

▼ Análisis

▼ Diagnóstico sobre el nivel de ajuste del modelo

Ahora se graficarán las learning curves para determinar si es underfitting u overfitting. Primero graficaremos las learning curves utilizando los datos de entrenamiento y pruebas para el modelo 4 que fue el más preciso.

```
plot_learning_curves(x_train, y_train, x_test, y_test, modelo4)
plt.title("Modelo 4 Learning Curves (train y test)")
plt.show()
```

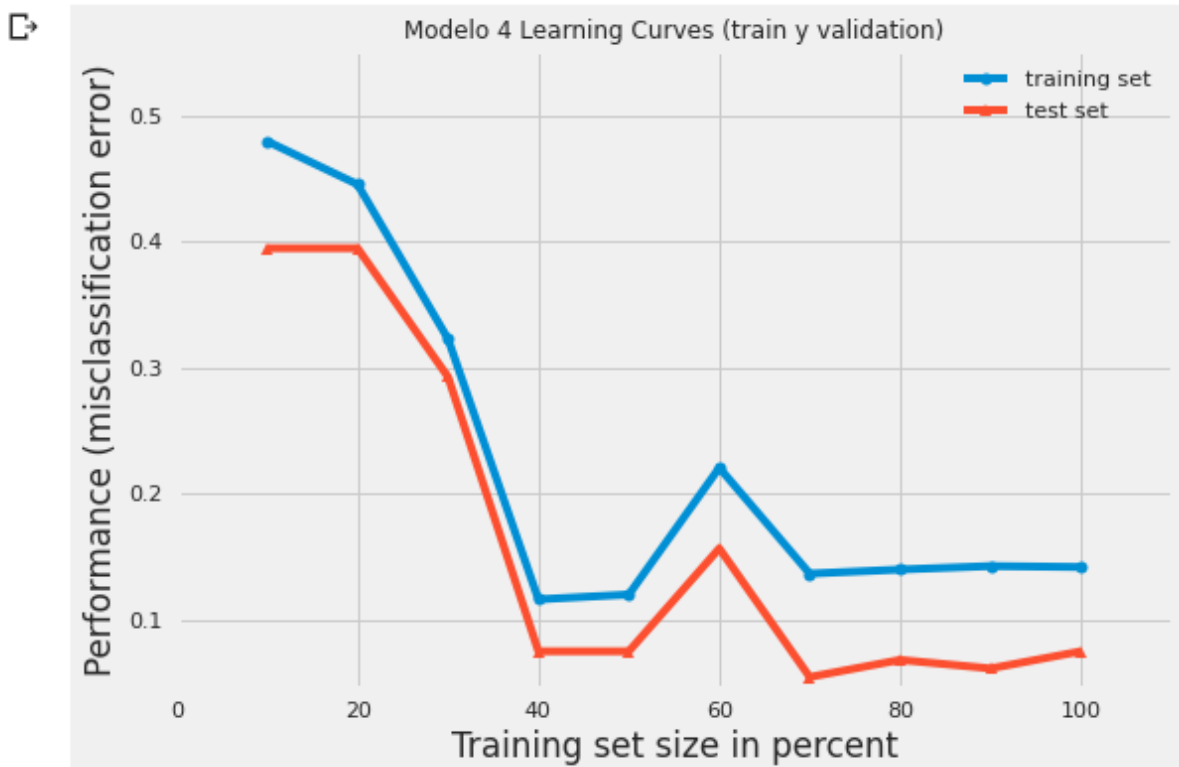


Podemos observar que tanto los valores de training como los del test son muy similares. Por lo tanto, el modelo sufre de underfitting.

```

▶ plot_learning_curves(x_train, y_train, x_val, y_val, modelo4)
  plt.title("Modelo 4 Learning Curves (train y validation)")
  plt.show()

```



Podemos observar que tanto los valores de training como los de validation son muy similares. Por lo tanto, el modelo sufre de underfitting al igual que la primera gráfica.

Diagnóstico y explicación del grado de bias o sesgo: bajo medio alto

Debido a que en el diagnóstico sobre el nivel de ajuste del modelo se obtuvo que el modelo sufría de underfitting, entonces se puede concluir que el modelo tiene un sesgo o bias alto. Además, se podría decir que es demasiado simple para representar la naturaleza de los datos.


Diagnóstico y explicación del grado de varianza: bajo medio alto

Debido a que en el diagnóstico sobre el nivel de ajuste del modelo se obtuvo que el modelo sufría de underfitting, entonces se puede concluir que el modelo tiene una varianza baja. Por lo tanto, existe un desajuste lo que hace que el modelo tenga un rendimiento predictivo deficiente.

Uso de técnicas de regularización o ajuste de parámetros para mejorar el desempeño del modelo

▼ Técnicas de regularización o el ajuste de parámetros para mejorar el desempeño del modelo

Debido a que ambas gráficas de learning curves mostraron que el modelo 4 sufre de underfitting entonces se implementará un modelo más con mayor complejidad.

```
1 s  modelo6 = MLPClassifier(random_state = 1,
                        hidden_layer_sizes = (10, 16, 10, 10, 10, 9, 15, 10, 10),
                        activation = "relu",
                        verbose = False,
                        solver = "adam",
                        learning_rate = "adaptive",
                        max_iter = 10000)

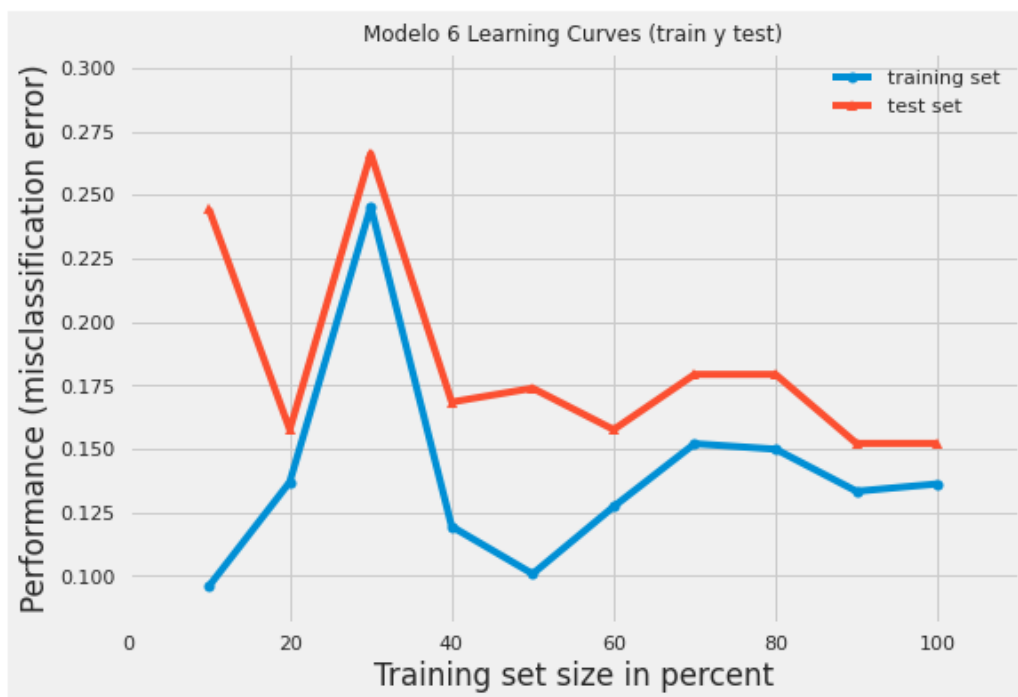
modelo6.fit(x_train, y_train)

print("Training score modelo 6: ", modelo6.score(x_train,y_train))
print("Validation score modelo 6: ", modelo6.score(x_val, y_val))
print("Test score modelo 6: ", modelo6.score(x_test, y_test))
```

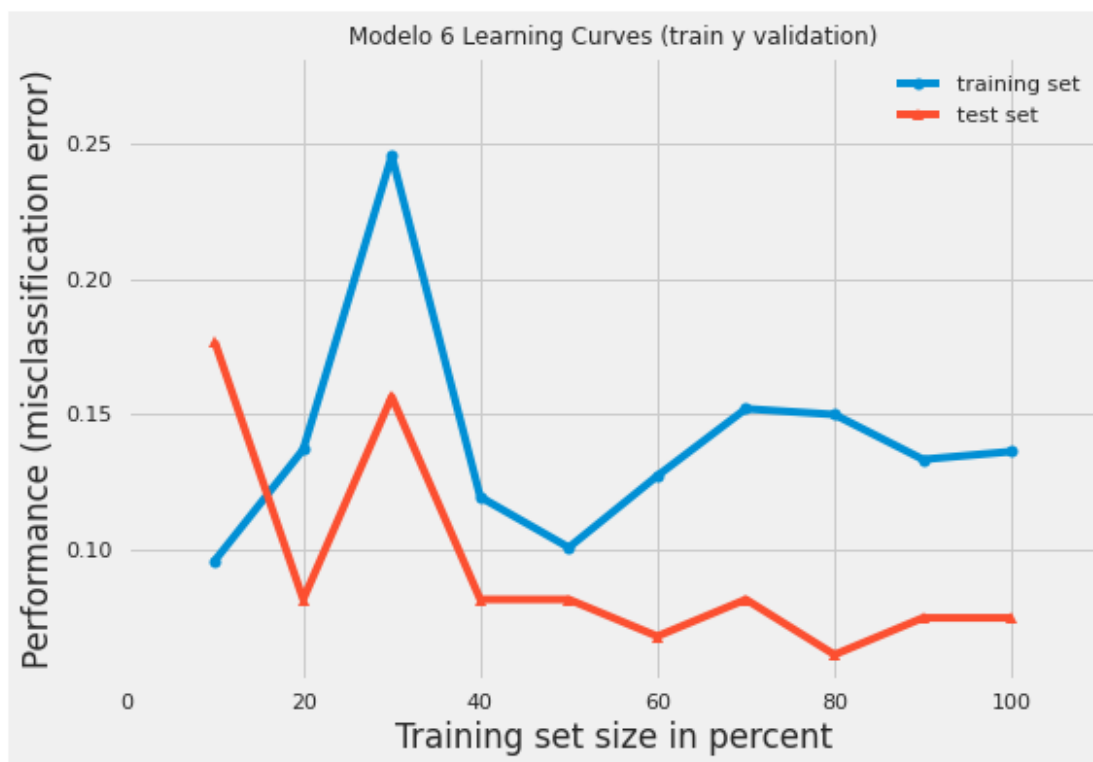
```
Training score modelo 6: 0.8637602179836512
Validation score modelo 6: 0.9251700680272109
Test score modelo 6: 0.8478260869565217
```

Como muestran los datos anteriores, el modelo 6 es preciso, pero los valores obtenidos son muy parecidos a los primeros cinco modelos. Por lo tanto, se realizarán las learning curves para analizar el nuevo modelo.

```
plot_learning_curves(x_train, y_train, x_test, y_test, modelo6)
plt.title("Modelo 6 Learning Curves (train y test)")
plt.show()
```



```
[256] plot_learning_curves(x_train, y_train, x_val, y_val, modelo6)
plt.title("Modelo 6 Learning Curves (train y validation)")
plt.show()
```



Las gráficas anteriores muestran una mejora significativa de los datos. Sobre todo en la gráfica con los datos de entrenamiento y prueba. Ya que el modelo tiene un buen balance, tiene un error aceptable en el entrenamiento y un error aceptable de generalización en el subconjunto de validación. Por lo tanto, muestra un diagnóstico deseado.

Comparación del desempeño del modelo antes y después de incluir las mejoras

▼ Comparación del desempeño del modelo antes y después de incluir las mejoras

Para comparar el desempeño del modelo antes y después se utilizarán el modelo 4 implementado antes de las modificaciones y el modelo 6 en el que se ajustaron los parámetros para mejorar su desempeño.

```
0 s ✓ ▶ print("Modelo 4")
print("Training score modelo 4: ", modelo4.score(x_train,y_train))
print("Validation score modelo 4: ", modelo4.score(x_val, y_val))
print("Test score modelo 4: ", modelo4.score(x_test, y_test))

print("\nModelo 6")
print("Training score modelo 6: ", modelo6.score(x_train,y_train))
print("Validation score modelo 6: ", modelo6.score(x_val, y_val))
print("Test score modelo 6: ", modelo6.score(x_test, y_test))
```

Modelo 4
Training score modelo 4: 0.8583106267029973
Validation score modelo 4: 0.9251700680272109
Test score modelo 4: 0.842391304347826

Modelo 6
Training score modelo 6: 0.8637602179836512
Validation score modelo 6: 0.9251700680272109
Test score modelo 6: 0.8478260869565217

Como se puede observar tanto los valores de training score y test score del modelo 6 es más preciso que el modelo 4. Sin embargo, la precisión del validation score de ambos modelos es el mismo.


```

▶ #Modelo 4
print('\nModelo 4: ')
print(modelo4.predict_proba(df_x.loc[0]))
print('Heart Disease? Estimated: ', modelo4.predict(df_x.loc[0]),
      'Real: ', df_y.loc[0])

#Modelo 6
print('\nModelo 6: ')
print(modelo6.predict_proba(df_x.loc[0]))
print('Heart Disease? Estimated: ', modelo6.predict(df_x.loc[0]),
      'Real: ', df_y.loc[0])

```

```

↳
Modelo 4:
[[0.91801577 0.08198423]]
Heart Disease? Estimated: [0] Real: 0

Modelo 6:
[[0.95623503 0.04376497]]
Heart Disease? Estimated: [0] Real: 0

```

Ahora, al realizar las predicciones se puede observar que ambos modelos están realizando la predicción de manera correcta. Como primer valor muestra la probabilidad de que el HeartDisease tenga un valor de 0 y 1. Después se muestra el valor estimado con los modelos y el valor real. En ambos modelos coinciden que el valor estimado por cada modelo y el valor real es 0, por lo que se puede confirmar que son modelos efectivos para la predicción de enfermedades cardíacas. Además se puede visualizar que la probabilidad de que se obtenga un valor de 0 en la predicción del modelo 6 es mayor que en el modelo 4.

```

▶ #Modelo 4
print('Modelo 4: ')
sns.set()
f, ax = plt.subplots()
matriz4 = confusion_matrix(y_test, pred4)
print(classification_report(y_test, pred4))
print('\nMatriz de confusión: ')
sns.heatmap(matriz4, annot=True, ax=ax, cbar=False, fmt='g', cmap='Pastel2')

```

```

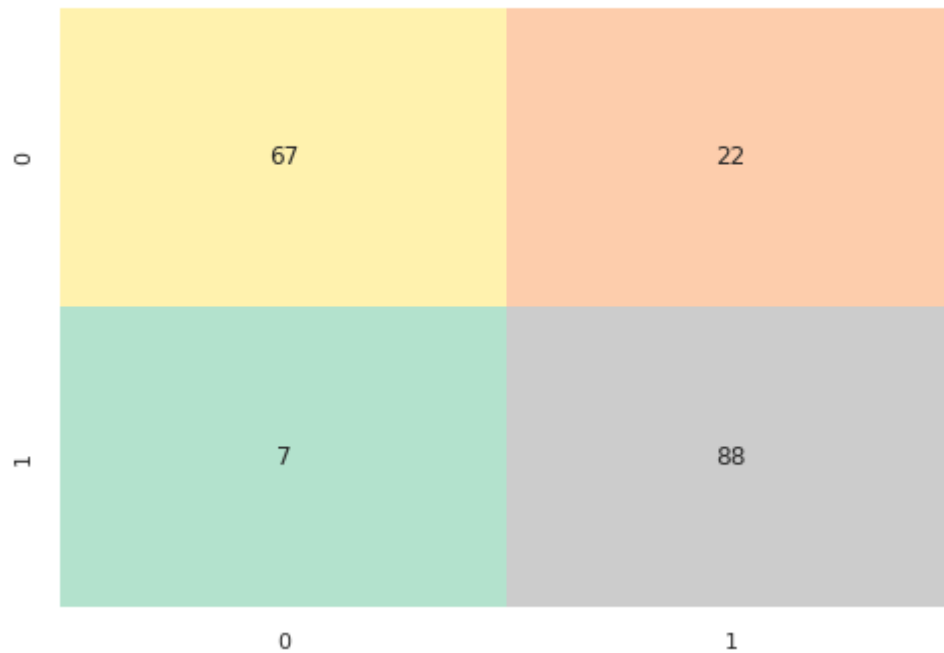
↳
Modelo 4:

```

	precision	recall	f1-score	support
0	0.91	0.75	0.82	89
1	0.80	0.93	0.86	95
accuracy			0.84	184
macro avg	0.85	0.84	0.84	184
weighted avg	0.85	0.84	0.84	184

Matriz de confusión:

<matplotlib.axes._subplots.AxesSubplot at 0x7fb79b31c590>



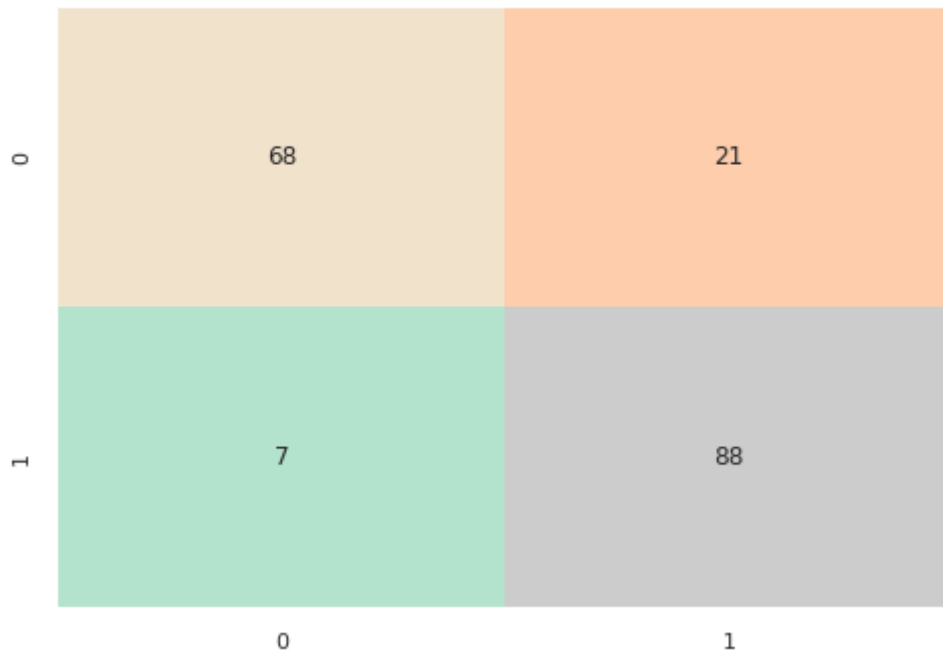
```
#Modelo 6
pred6 = modelo6.predict(x_test)
print('Modelo 6: ')
sns.set()
f, ax = plt.subplots()
matriz6 = confusion_matrix(y_test,pred6)
print(classification_report(y_test,pred6))
print('\nMatriz de confusión: ')
sns.heatmap(matriz6, annot=True, ax=ax, cbar=False, fmt='g', cmap='Pastel2')
```

Modelo 6:

	precision	recall	f1-score	support
0	0.91	0.76	0.83	89
1	0.81	0.93	0.86	95
accuracy			0.85	184
macro avg	0.86	0.85	0.85	184
weighted avg	0.86	0.85	0.85	184

Matriz de confusión:

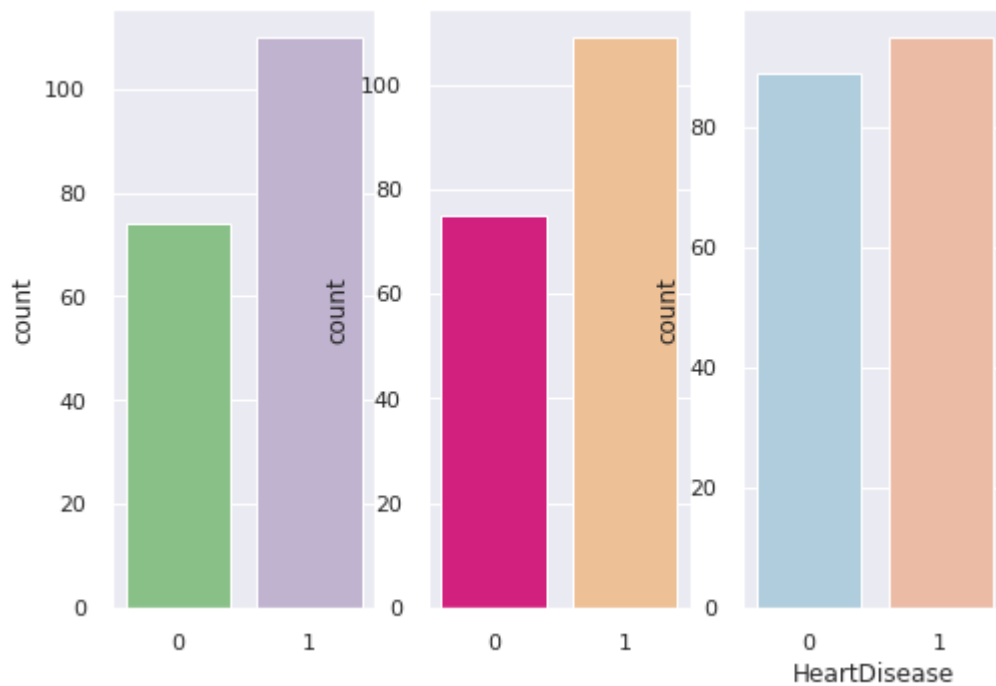
<matplotlib.axes._subplots.AxesSubplot at 0x7fb79b7da150>



Para el cuarto modelo podemos observar que el valor de f1-score es 0.84. En la matriz de confusión se tienen 67 valores true positive y 88 valores true negative; es decir, valores predichos que coinciden con el valor real. Además, se tienen 22 valores false positive y 7 false negative es decir valores erróneos.

Por otro lado, en el sexto modelo se muestra un valor de f1-score de 0.85 lo que indica una mejora en el desempeño de este modelo. En la matriz de confusión se tienen 68 valores true positive y 88 valores true negative; es decir, valores predichos que coinciden con el valor real. Además, se tienen 21 valores false positive y 7 false negative es decir valores erróneos.

```
fig, ax = plt.subplots(1,3)
sns.countplot(pred4, ax=ax[0], palette="Accent")
sns.countplot(pred6, ax=ax[1], palette="Accent_r")
sns.countplot(y_test, ax=ax[2], palette="RdBu_r")
fig.show()
```



En la gráfica anterior se muestra del lado izquierdo el número de valores predecidos con el modelo cuatro, en medio el número de valores predecidos con el modelo seis y del lado derecho los valores reales. Se puede observar que los datos predecidos con valor 1 en ambas gráficas (modelo 4 y 6) se acercan bastante a los valores reales. A pesar de que hay una diferencia significativa en los valores igual a cero de ambos modelos y los valores reales, podemos notar una pequeña mejora en el modelo 6.