Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Monterrey

Escuela de ingeniería y ciencias

Inteligencia Artificial Avanzada para la Ciencia de Datos I

Momento de Retroalimentación: Módulo 2 Análisis y Reporte sobre el desempeño del modelo. (Portafolio Análisis)

Alumna:

Ariadna Jocelyn Guzmán Jiménez A01749373

Profesor:

Iván Mauricio Anaya Contreras

Fecha:

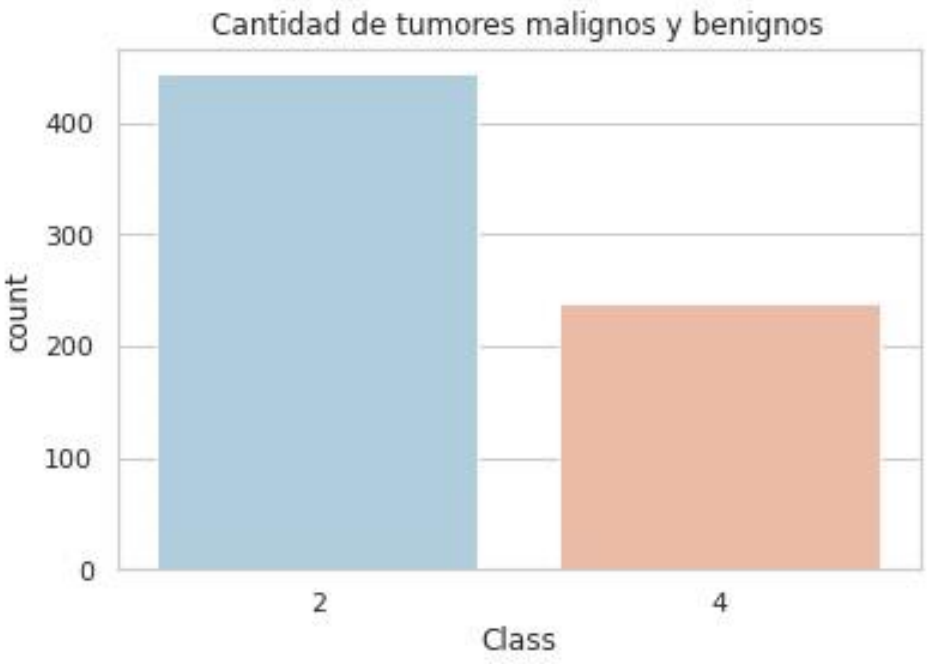
9 de septiembre de 2022

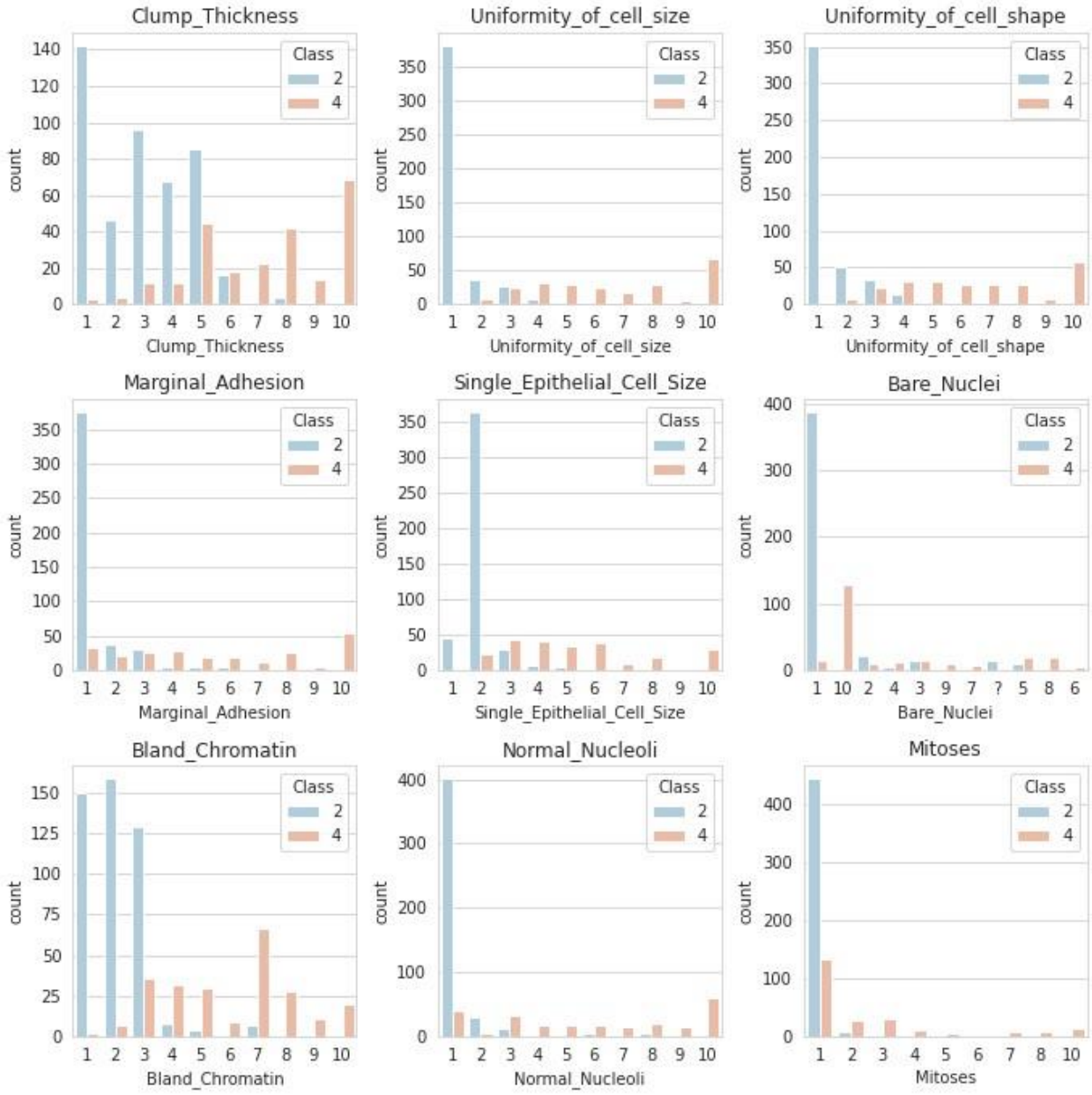
Random Forest Classifier

El algoritmo de bosque aleatorio es utilizado comúnmente para machine learning ya que combina las salidas múltiples de un árbol de decisión para poder alcanzar sólo un resultado. Esto, ha dado pie a que su utilidad se logre de manera fácil, pudiéndose utilizar para problemas de clasificación y regresión.

En esta ocasión, nos enfrentamos a una situación donde contamos con un dataset de nombre ***"Breast Cancer Wisconsin (Original)"*** que nos da datos de agrupaciones cronológicas sobre la información de tumores de cáncer de mama. Dado lo anterior, cuenta con 12 columnas, donde 11 de ellas, contienen información acerca de las características del tumor en escala del 1 al 10, y la doceava, es la clase que define con respecto a los datos anteriores, si el tumor es benigno (representado con el número 2) o maligno (representado con el número 4).

Como inicio para la implementación del algoritmo, se hizo un análisis previo de los datos, para entender más a fondo sobre lo que nos decía el dataset. Para ello, se realizaron gráficas de comparación entre los datos, visualizando como se comportaban los tumores de acuerdo con cada columna perteneciente al dataframe y finalmente, con cuantos tumores malignos y benignos contábamos.





Posteriormente, se realizó un tratamiento de datos donde verificábamos si existían valores nulos o algún valor que no nos beneficiara en el proceso, donde encontramos, que la base de datos contaba con un valor tipo string “?” en la columna Bare Nuclei, que lo interpreta como un valor no disponible. Para ello, eliminamos las filas que contenían este dato y realizamos la conversión de la columna a una con tipos de datos int para continuar con nuestro proceso.

Separación del dataset

De inicio, hicimos la separación de valores de las columnas como se mencionó en un principio, lo declara el dataset, siendo y la columna ***"Class".***

```
# Dividir los datos en variables x, y
x = cancer.iloc[:, 0:8]
y = cancer.iloc[:, 9].values
```

Con ayuda de la librería ***sklearn.model_selection,*** utilizamos la función train_test_split para poder dividir nuestros datos en entrenamiento (60%), validación (20%) y prueba (20%), esto, para poder comprobar la eficiencia del algoritmo posteriormente.

```
# Dividir el dataset
from sklearn.model_selection import train_test_split

# Modelo prueba 20% y entrenamiento 60% y validación 20%
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 0)
x_train2, x_val, y_train2, y_val = train_test_split(x_train, y_train, test_size=0.2, random_state=0)
```

Es importante antes de implementar el modelo, escalar los datos, ya que transforma los valores de forma que todos compartan un mismo promedio y desviación media, evitando fugas de datos y aplicándolo de forma equitativa para el entrenamiento, prueba y validación.

```
# Escalamiento de los datos
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
x_val = scaler.transform(x_val)
```

La implementación de nuestro algoritmo se realizó con la librería ***"sklearn.ensemble"*** con 5 pruebas diferentes en donde, veíamos el comportamiento a través de un cambio de estimadores de forma ascendente (2, 4, 6, 8 y 10). Estas comparaciones, permite ver más a fondo cuál es el parámetro más conveniente para poder continuar con la implementación y calcular predicciones. Dentro de estos resultados, notamos que, entre más estimadores, viendo que, a partir del 6, la precisión mejora.

```python
# Aplicacion de Random Forest
from sklearn.ensemble import RandomForestClassifier

for i in range (0, 12, 2):
  if (i > 0):
    forest = RandomForestClassifier(n_estimators = i, criterion = 'entropy', random_state = 0)
    model = forest.fit(x_train, y_train)
    print('Random Forest Classifier Training Accuracy with {} estimators: {}'.format(i, forest.score(x_train, y_train)))
    print('Random Forest Classifier Test Accuracy with {} estimators: {}'.format(i, forest.score(x_test, y_test)))
    print('Random Forest Classifier Validation Accuracy with {} estimators: {} \n'.format(i, forest.score(x_val, y_val)))
```

```
Random Forest Classifier Training Accuracy with 2 estimators: 0.978021978021978
Random Forest Classifier Test Accuracy with 2 estimators: 0.9416058394160584
Random Forest Classifier Validation Accuracy with 2 estimators: 0.9636363636363636

Random Forest Classifier Training Accuracy with 4 estimators: 0.9926739926739927
Random Forest Classifier Test Accuracy with 4 estimators: 0.9562043795620438
Random Forest Classifier Validation Accuracy with 4 estimators: 0.990909090909091

Random Forest Classifier Training Accuracy with 6 estimators: 0.9981684981684982
Random Forest Classifier Test Accuracy with 6 estimators: 0.9635036496350365
Random Forest Classifier Validation Accuracy with 6 estimators: 1.0

Random Forest Classifier Training Accuracy with 8 estimators: 0.9963369963369964
Random Forest Classifier Test Accuracy with 8 estimators: 0.9635036496350365
Random Forest Classifier Validation Accuracy with 8 estimators: 1.0

Random Forest Classifier Training Accuracy with 10 estimators: 0.9963369963369964
Random Forest Classifier Test Accuracy with 10 estimators: 0.9635036496350365
Random Forest Classifier Validation Accuracy with 10 estimators: 1.0
```

Vemos que el peor modelo en esta ocasión, fue implementarlo con 2 estimadores, mientras que el mejor, fue con 10, por lo cual, tomaremos este modelo base para poder predecir, pero antes, vemos la diferencia entre estas dos.

Comparación del mejor y peor modelo

```python
forest2 = RandomForestClassifier(n_estimators = 2, criterion = 'entropy', random_state = 0)
model2 = forest2.fit(x_train, y_train)

forest10 = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)
model10 = forest10.fit(x_train, y_train)

print('Best vs worst train accuracy: {}'.format(forest10.score(x_train, y_train) - forest2.score(x_train, y_train)))
print('Best vs worst test accuracy: {}'.format(forest10.score(x_test, y_test) - forest2.score(x_test, y_test)))
print('Best vs worst validate accuracy: {}'.format(forest10.score(x_val, y_val) - forest2.score(x_val, y_val)))
```

```
Best vs worst train accuracy: 0.01831501831501836
Best vs worst test accuracy: 0.021897810218978186
Best vs worst validate accuracy: 0.036363636363636376
```

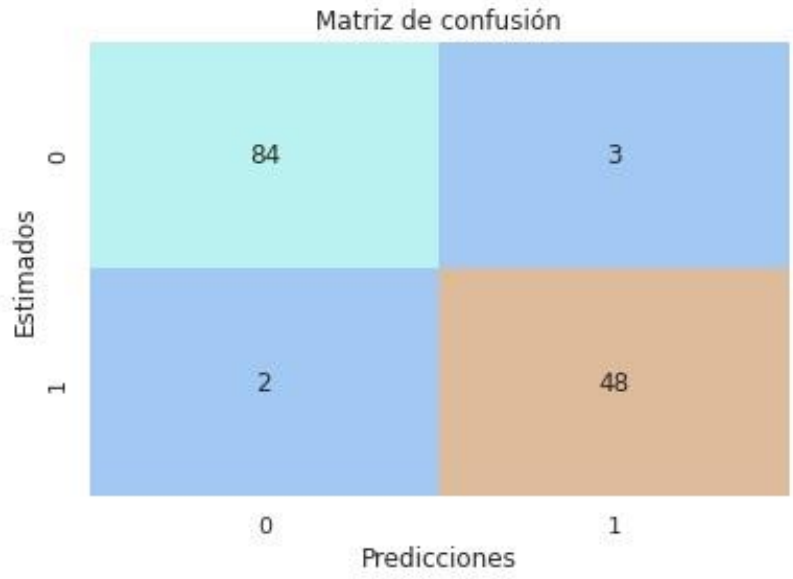
Posteriormente, vemos nuestras predicciones, dándonos cuenta de que la diferencia entre los valores estimados y los predichos, es mínima:

```
==================== Predicciones con 10 estimadores ====================

Predicciones:  [2 2 4 4 2 2 2 4 2 2 4 2 4 2 2 2 4 4 4 2 2 2 4 2 4 4 2 2 2 4 2 4 4 2 2 2 4
 4 2 4 2 2 2 2 2 2 2 4 2 2 4 2 4 2 2 2 4 4 2 4 2 2 2 2 2 2 2 2 4 4 2 2 2 2
 2 2 4 2 2 2 4 2 4 2 2 4 2 4 4 2 4 2 4 4 4 4 4 4 2 2 2 2 2 4 4 2 2 4 2 2 2 4
 2 2 4 2 2 2 4 2 2 2 4 2 2 4 4 2 4 2 4 2 2 4 2 2 4 2]

Valores estimados:  [2 2 4 4 2 2 2 4 2 2 4 2 4 2 2 2 4 4 4 2 2 2 4 2 4 4 2 2 2 4 2 4 4 2 2 2 4
 4 2 4 2 2 2 2 2 2 2 4 2 2 4 2 4 2 2 2 4 2 2 4 2 2 2 2 2 2 2 2 4 4 2 2 2 2
 2 2 4 2 2 2 4 2 4 2 2 4 2 2 4 2 4 2 4 4 4 2 4 4 4 2 2 2 4 4 2 2 4 4 2 2 4
 2 2 4 2 2 2 4 2 2 2 4 2 2 4 4 2 4 2 4 2 2 4 2 2 4 2]
```

Esto, además de solo detonarlo por una comparación entre datos, podemos verlo de forma más sencilla a través de una matriz de confusión y el cálculo de los errores medios del modelo, donde observamos la verdadera cantidad de diferencia y valores no acertados en la predicción.



132 valores correctamente estimados y 5 fallidos

```
Error absoluto medio: 0.072992700729927
Error cuadrático medio:  0.145985401459854
```

Diferencias entre estimaciones y predicciones

Finalmente, vemos los elementos gráficos de los comportamientos de cada una de las divisiones del dataset, donde en para ambas, se muestra que el ***"training test"*** fue de más ayuda para los cálculos de las predicciones, además de que se presenta una interacción ideal ya que se cuenta con un error aceptable en entrenamiento y

generalización no superando un error absoluto de 0.10. Por ende, se considera que, debido al error bajo, también hay una **varianza y sesgo bajo**.