TC3006C – Inteligencia Artificial Avanzada Para La Ciencia De Datos

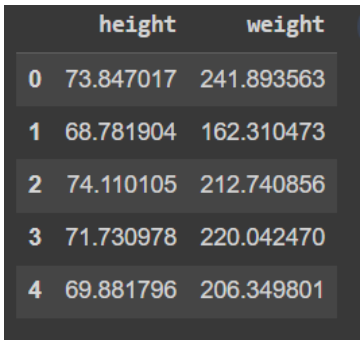
Módulo 2: Aprendizaje Automático

Análisis y Reporte sobre el desempeño del modelo

Ingeniero: César Javier Guerra Páramo

Josue Salvador Cano Martinez | A00829022

1. Implementación elegida: uso de framework, sklearn (DecisionTreeRegressor)
   En las dos entregas se trabajó con datos referentes a peso y estatura, en ambos casos la finalidad fue encontrar un modelo que permitiese correlacionar ambas variables y así predecir una a partir de la otra. Para el primer entregable se trabajó con una regresión lineal y para el segundo con la implementación de una regresión de árbol de decisión; esta última se eligió para realizar el presente análisis debido al mejor desempeño que presenta en el modelado de los datos.



	height	weight
0	73.847017	241.893563
1	68.781904	162.310473
2	74.110105	212.740856
3	71.730978	220.042470
4	69.881796	206.349801

```
# Get info of dataframe
print(df.info())
```

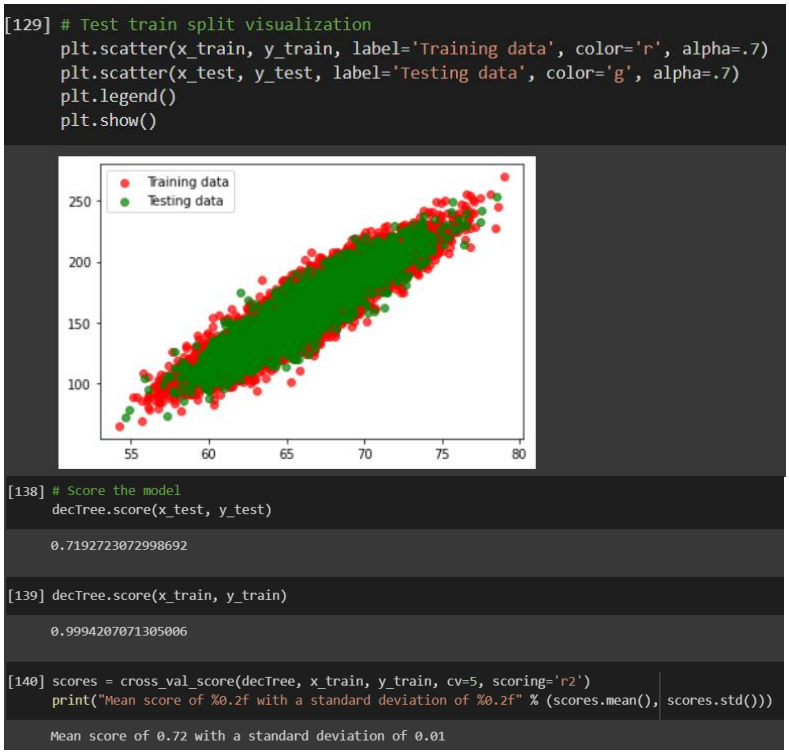
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   height  10000 non-null  float64
 1   weight  10000 non-null  float64
dtypes: float64(2)
memory usage: 156.4 KB
None
```

```
[124] # Calculate duplicated values
print("Suma de duplicados: " + str(df.duplicated().sum()))
```

```
Suma de duplicados: 0
```

   En las imágenes mostradas es posible apreciar algunos de los registros del set de datos con el que se trabajó y también la confirmación de datos sin valores duplicados que nos permite realizar un mejor análisis.

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



```
[129] # Test train split visualization
plt.scatter(x_train, y_train, label='Training data', color='r', alpha=.7)
plt.scatter(x_test, y_test, label='Testing data', color='g', alpha=.7)
plt.legend()
plt.show()
```



```
[138] # Score the model
decTree.score(x_test, y_test)
```

```
0.7192723072998692
```

```
[139] decTree.score(x_train, y_train)
```

```
0.9994207071305006
```

```
[140] scores = cross_val_score(decTree, x_train, y_train, cv=5, scoring='r2')
print("Mean score of %0.2f with a standard deviation of %0.2f" % (scores.mean(), scores.std()))
```

```
Mean score of 0.72 with a standard deviation of 0.01
```

   La fuente de datos no se proporcionaba con un archivo diferente para test, por lo que se recurrió a dividir el archivo con una proporción de 20% para test y 80% para training del modelo, en la gráfica mostrada es

posible apreciar la distribución de dichos grupos en que fueron divididos los datos. Una vez entrenado el modelo se procedió a obtener su 'score' utilizando los datos de test y de train previamente creados, donde es posible percatarse del 'accuracy' que cada uno de ellos muestra tener; algo que se puede notar es la diferencia de 'score' que tiene cada set de datos: mientras que el train arroja 0.99 (casi perfecto), el test nos demuestra lo contrario con un 0.71, estos resultados son normales y necesarios de obtener antes de concluir qué tan bueno es un modelo, donde para este caso en particular si bien realmente no es perfecto tampoco podemos decir que es malo en lo que respecta a la realización de predicciones haciendo uso del modelo creado.

b. Diagnóstico y explicación el grado de bias y varianza: bajo medio alto

```
# Calculate bias
data_true = 155.414139
data_predicted = prediction[1]
MBE = np.mean(data_predicted - data_true)
print("MBE: ", MBE)
```

```
MBE:  -4.854162400000007
```

```
[137] # Calculate variance
import statistics
variance = statistics.variance(prediction)
print(variance)
```

```
1022.7388896978894
```

Al tratarse de una regresión de árbol de decisión el bias es bajo debido a que el modelo es muy flexible al ajuste de los datos, por otro lado, tiene una varianza alta (cambiar los datos de entrenamiento produce grandes cambios en la estimación).

c. Diagnóstico y explicación el nivel de ajuste del modelo: underfitt fitt overfitt
Como se pudo demostrar anteriormente, el árbol de decisión implementado presenta un bajo bias y una alta varianza, lo cual puede entenderse como un nivel de ajuste alto (overfitt). La manera de corroborarlo en el inciso a) del primer punto: el modelo arroja un 'score' de 0.99 con los datos de train, pero al introducirle los datos de test podemos percatarnos de que dicho valor no es real.

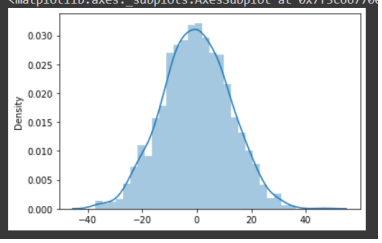
2. Técnicas de regularización implementadas: Lasso y Ridge
Para buscar mejorar el score del modelo se recurrió aplicar técnicas de regularización: Lasso y Ridge. Como se presenta en los resultados, después de implementar estas mejoras el modelo pasó de tener un score de 0.71 a 0.85; tanto Lasso como Ridge permitieron obtener resultados muy similares, lo cual se puede visualizar en las gráficas y en los scores calculados.

a. Resultados:

```
[141] # Lasso model regularization
      lasso_reg = Lasso(alpha=0.3)
      lasso_reg.fit(x_train, y_train)

      Lasso(alpha=0.3)

[142] lasso_reg.score(x_test, y_test)

      0.8513712187225047

[143] lasso_reg.score(x_train, y_train)

      0.8561149964826482
```

```
[144] # Ridge model regularization
      ridge_reg = Ridge(alpha=0.3)
      ridge_reg.fit(x_train, y_train)

      Ridge(alpha=0.3)

[145] ridge_reg.score(x_test, y_test)

      0.8513619002560128

[146] ridge_reg.score(x_train, y_train)

      0.8561208934784967
```

```
[147] # Lasso and ridge predictions
      prediction_lasso=lasso_reg.predict(x_test)
      prediction_ridge=ridge_reg.predict(x_test)

[148] prediction_lasso=prediction_lasso.reshape(2000,1)
      sns.distplot(y_test-prediction_lasso)

      /usr/local/lib/python3.7/dist-packages/seaborn/distri
        warnings.warn(msg, FutureWarning)
      <matplotlib.axes._subplots.AxesSubplot at 0x7f3c66770
```




```
sns.distplot(y_test-prediction_ridge)

/usr/local/lib/python3.7/dist-packages/seaborn/distri
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c62347
```