Módulo 2 Análisis y Reporte sobre el desempeño del modelo

Juan Carlos Varela Téllez A01367002
Fecha de inicio: 09/09/2022
Fecha de finalizacion: 09/09/2022

En caso de no tener las bibliotecas necesarias, utilizar los siguientes comandos:
python -m pip install numpy
python -m pip install pandas
python -m pip install seaborn
python -m pip install matplotlib
python -m pip install scikit-learn

Este codigo es una continuacion indirecta del siguiente repositorio: https://github.com/JuanVaTe/RetoModulo2Framework Se recomienda leerlo antes de continuar, aunque no es necesario para entender este archivo.

Cuando se habla de un modelo de machine learning, en el ojo publico se piensa que es magia; un conjunto de comandos magicos donde se meten datos y salen mas datos, sin embargo, esto no es verdad. Es un conjunto de instrucciones estructuradas que se utilizan en una situacion en especifico, y es por esta razon que no todos los modelos se deben de utilizar en todas las situaciones. Hay un arte llamado *afinamiento de modelos* que es basicamente eso, afinar un modelo para que funcione con un conjunto de datos en especifico. Este concepto es lo que se va a mostraren este codigo.
Utilizaremos un modelo de arbol de decision predisenado por mi. El codigo completo lo puedes encontrar aqui:
https://github.com/JuanVaTe/RetoModulo2Framework
Este modelo se optimizo para poder predecir a las personas mas propensas a tener una apoplejia, sin embargo, vamos a ver que tan bien es su rendimiento cuando se utiliza el mismo modelo para una situacion completamente diferente, y a partir de ahi vamos a afinarlo hasta que cumpla nuestras expectativas con nuestros datos nuevos.

Para poder leer, procesar y analizar los datos e información que sacaremos de dichos datos es necesario importar ciertas bibliotecas que nos ayudarán de forma importante:

- Pandas: Esta biblioteca nos ayuda a leer nuestros datos, al igual que modificar nuestros datos a traves de un data-frame para manipularlos y analizarlos. Para más información haz click aquí.
- Numpy: Esta biblioteca nos da diferentes herramientas matematicas vectorizadas para acelerar nuestros calculos. Para más información haz click aquí.
- Matplotlib: esta biblioteca nos da la posibilidad de crear diferentes tipos de gráficos con mucha personalización. Para más información haz click aqui.
- Seaborn: Esta biblioteca también nos da herramientas para poder graficar y visualizar datos, sin embargo, es para uso rápido ya que tiene muchas plantillas que podemos utilizar. Para más información haz click aquí.
- Scikit-learn: Esta biblioteca es de las más importantes que se utiliza ya que contiene la gran mayoría de herramientas de machine learning que se van a utilizar en este reto, desde regresiones hasta bosques aleatorios. Para más información haz click aqui.

Ahora leeremos un data-set nuevo, la informacion del data y el data-set en si lo puedes encontrar aqui.

Primero necesitamos saber que datos contiene nuestro data-set:
Informacion sacada de https://www.kaggle.com/datasets/uciml/mushroom-classification

Attribute Information: (classes: edible=e, poisonous=p)

- cap-shape: bell=b,conical=c,convex=x,flat=f, knobbed=k,sunken=s
- cap-surface: fibrous=f,grooves=g,scaly=y,smooth=s
- cap-color: brown=n,buff=b,cinnamon=c,gray=g,green=r,pink=p,purple=u,red=e,white=w,yellow=y
- bruises: bruises=t,no=f
- odor: almond=a,anise=l,creosote=c,fishy=y,foul=f,musty=m,none=n,pungent=p,spicy=s
- gill-attachment: attached=a,descending=d,free=f,notched=n
- gill-spacing: close=c,crowded=w,distant=d
- gill-size: broad=b,narrow=n
- gill-color: black=k,brown=n,buff=b,chocolate=h,gray=g, green=r,orange=o,pink=p,purple=u,red=e,white=w,yellow=y
- stalk-shape: enlarging=e,tapering=t
- stalk-root: bulbous=b,club=c,cup=u,equal=e,rhizomorphs=z,rooted=r,missing=?
- stalk-surface-above-ring: fibrous=f,scaly=y,silky=k,smooth=s
- stalk-surface-below-ring: fibrous=f,scaly=y,silky=k,smooth=s
- stalk-color-above-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e,white=w,yellow=y
- stalk-color-below-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e,white=w,yellow=y
- veil-type: partial=p,universal=u

- veil-color: brown=n,orange=o,white=w,yellow=y
- ring-number: none=n,one=o,two=t
- ring-type: cobwebby=c,evanescent=e,flaring=f,large=l,none=n,pendant=p,sheathing=s,zone=z
- spore-print-color: black=k,brown=n,buff=b,chocolate=h,green=r,orange=o,purple=u,white=w,yellow=y
- population: abundant=a,clustered=c,numerous=n,scattered=s,several=v,solitary=y
- habitat: grasses=g,leaves=l,meadows=m,paths=p,urban=u,waste=w,woods=d

Nuestro data-set cuenta con 22 columnas, 21 siendo variables independientes y 1 siendo la variable independiente.
Dentro de la documentacion y muestra de los datos, podemos observar que la columna `veil-type` cuenta con unicamente un valor, asi que al preprocesarlo quitaremos esta carcteristica.

Preprocesamiento y limpieza de datos

Lo bueno: no hay valores nulos pero si valores no nulos que representen valores nulos (como un valor de '?', que es igual a no tener valor), asi que despues del analisis estadistico vamos a decidir si conservar la columna ya que, son muchas caracteristicas, o quitar las filas que tiene este dato invalido, ya que tenemos muchas entradas.
Lo malo: todas las caracteristicas son cualitativas y casi ninguna es binaria, lo cual va a ser un problema al cuantificar estos valores ya que nos vamos a quedar con muchas columnas.
Para que no haya dudas, la variable dependiente indica si un champinon es venenoso o comestible
Ya que no se tiene que hacer limpieza de datos, tenemos que empezar con el preprocesamiento directamente, asi que es tiempo de separar nuestras variables dependientes e independientes.
Ya que nuestros datos estan separados, vamos a empezar a cuantificarlos, que es basicamente cuando reemplazas datos no numericos con datos numericos. Pandas cuenta con una funcion que nos va a ayudar con eso que es la funcion `get_dummies()`.

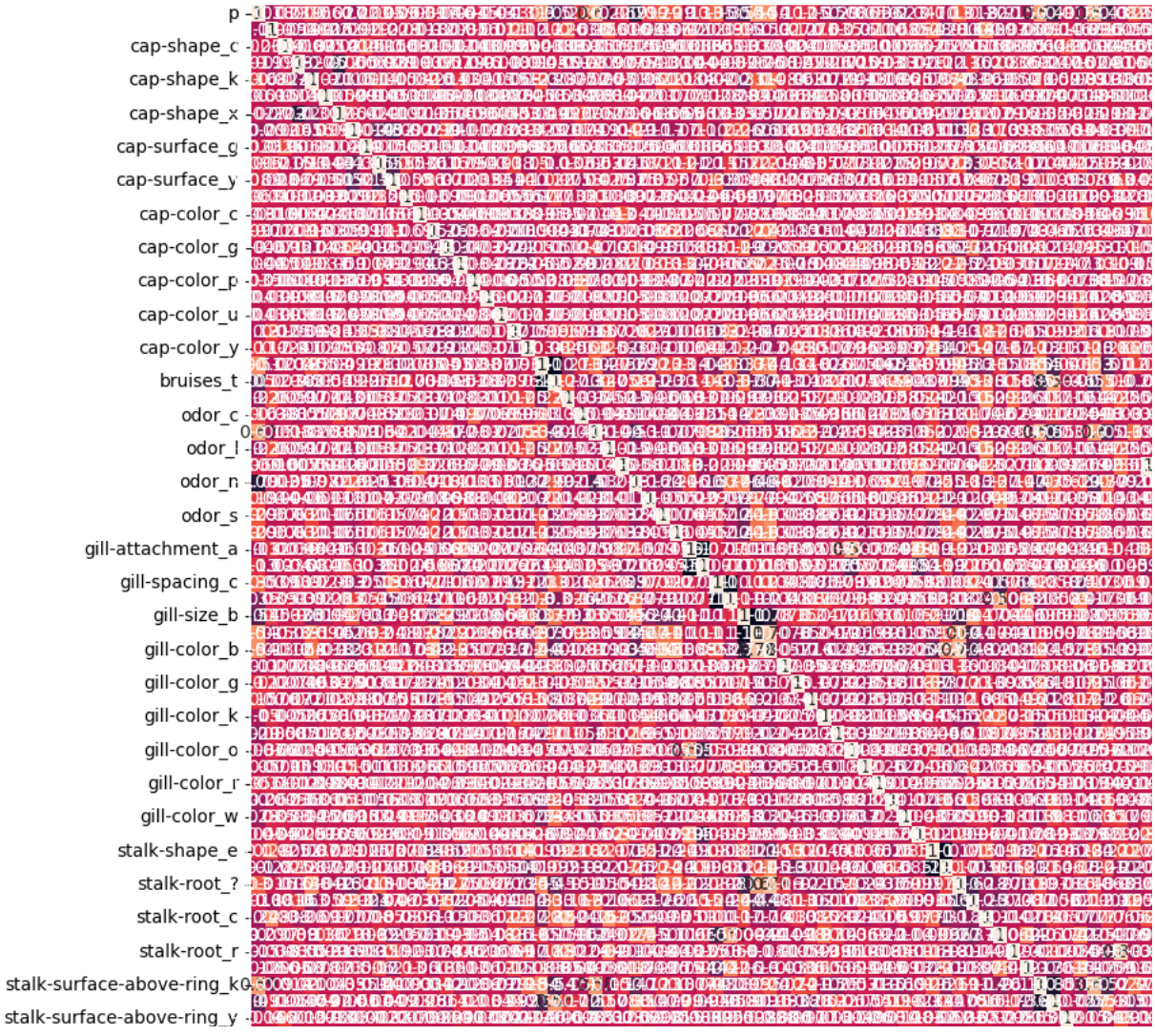
```
# Cuantificando la variable dependiente
mushroom_y = pd.get_dummies(mushroom_y, drop_first=True)['p']

# Ahora toca cuantificar las variables indpendientes

# Cuantificamos cap-shape
dummy_cap_shape = pd.get_dummies(mushroom_x['cap-shape'], prefix='cap-shape')

# Cuantificamos cap-surface
dummy_cap_surface = pd.get_dummies(mushroom_x['cap-surface'], prefix='cap-surface')

# Cuantificamos cap-color
dummy_cap_color = pd.get_dummies(mushroom_x['cap-color'], prefix='cap-color')

# Cuantificamos bruises
dummy_bruises = pd.get_dummies(mushroom_x['bruises'], prefix='bruises')

# Cuantificamos odor
dummy_odor = pd.get_dummies(mushroom_x['odor'], prefix='odor')

# Cuantificamos gill-attachment
dummy_gill_attachment = pd.get_dummies(mushroom_x['gill-attachment'], prefix='gill-attachment')

# Cuantificamos gill-spacing
dummy_gill_spacing = pd.get_dummies(mushroom_x['gill-spacing'], prefix='gill-spacing')

# Cuantificamos gill-size
dummy_gill_size = pd.get_dummies(mushroom_x['gill-size'], prefix='gill-size')

# Cuantificamos gill-color
dummy_gill_color = pd.get_dummies(mushroom_x['gill-color'], prefix='gill-color')

# Cuantificamos stalk-shape
dummy_stalk_shape = pd.get_dummies(mushroom_x['stalk-shape'], prefix='stalk-shape')

# Cuantificamos stalk-root
dummy_stalk_root = pd.get_dummies(mushroom_x['stalk-root'], prefix='stalk-root')

# Cuantificamos stalk-surface-above-ring
dummy_stalk_surface_above_ring = pd.get_dummies(mushroom_x['stalk-surface-above-ring'], prefix='stalk-surface-above-ring')

# Cuantificamos stalk-surface-below-ring
dummy_stalk_surface_below_ring = pd.get_dummies(mushroom_x['stalk-surface-below-ring'], prefix='stalk-surface-below-ring')

# Cuantificamos stalk-color-above-ring
dummy_stalk_color_above_ring = pd.get_dummies(mushroom_x['stalk-color-above-ring'], prefix='stalk-color-above-ring')

# Cuantificamos stalk-color-below-ring
dummy_stalk_color_below_ring = pd.get_dummies(mushroom_x['stalk-color-below-ring'], prefix='stalk-color-below-ring')

# Cuantificamos veil-color
dummy_veil_color = pd.get_dummies(mushroom_x['veil-color'], prefix='veil-color')

# Cuantificamos ring-number
dummy_ring_number = pd.get_dummies(mushroom_x['ring-number'], prefix='ring-number')

# Cuantificamos ring-type
dummy_ring_type = pd.get_dummies(mushroom_x['ring-type'], prefix='ring-type')

# Cuantificamos spore-print-color
dummy_spore_print_color = pd.get_dummies(mushroom_x['spore-print-color'], prefix='spore-print-color')

# Cuantificamos population
dummy_population = pd.get_dummies(mushroom_x['population'], prefix='population')

# Cuantificamos habitat
dummy_habitat = pd.get_dummies(mushroom_x['habitat'], prefix='habitat')
```

```
# Concatenamos todos los dummies
mushroom_x_all = pd.concat([dummy_cap_shape, dummy_cap_surface, dummy_cap_color,
                                                    dummy_bruises,
                                                    dummy_odor,
                                                    dummy_gill_attachment, dummy_gill_spacing, dummy_gill_size, dummy_g
                                                    dummy_stalk_shape, dummy_stalk_root,
                                                    dummy_stalk_surface_above_ring, dummy_stalk_surface_below_ring,
                                                    dummy_stalk_color_above_ring, dummy_stalk_color_below_ring,
                                                    dummy_veil_color,
                                                    dummy_ring_number, dummy_ring_type,
                                                    dummy_spore_print_color,
                                                    dummy_population,
                                                    dummy_habitat], axis=1)

# Como se puede observar, quitamos la caracteristica de veil-type.
```

Con los datos cuantificados, ahora podemos hacer nuestro analisis estadistico y encontrar correlaciones entre nuestras caracteristicas y nuestra variable dependiente.

```
correlation = pd.concat([mushroom_y, mushroom_x_all], axis=1).corr()
f, ax = plt.subplots(figsize=(20, 20))
sns.heatmap(correlation, annot=True)
plt.show()
```

:arrow_down: :arrow_down: :arrow_down: :arrow_down:



Debido a la monstruosidad de matriz de correlacion que obtuvimos, investigue en internet si es posible obtener los pares de valores con mayor correlacion.
El siguiente pedazo de codigo NO ES MIO aunque lo modifique para que se adaptara a este problema
Creditos van para:
HYRY: https://stackoverflow.com/users/772649/hyry (autor de la respuesta)
Michel de Ruiter: https://stackoverflow.com/users/357313/michel-de-ruiter (mantuvo la respues actualizada)
Link de la pregunta en StackOverflow: https://stackoverflow.com/questions/17778394/list-highest-correlation-pairs-from-a-large-correlation-matrix-in-pandas

```
c = correlation.abs()

s = c.unstack()
mayor_correlacion = s.sort_values(kind="quicksort")
```

El codigo nos da esta lista:

```
# Correlaciones con la variable dependiente ==========
p  stalk-surface-above-ring_y    0.016198
        stalk-root_b                 0.017712
        cap-shape_f                  0.018526
        cap-surface_g                0.023007
        cap-shape_c                  0.023007
        cap-shape_x                  0.026886
        cap-color_c                  0.030910
        stalk-color-above-ring_y     0.032545
        veil-color_y                 0.032545
        ...
        bruises_f                    0.501530
        bruises_t                    0.501530
        gill-color_b                 0.538808
```

```
gill-size_b                  0.540024
gill-size_n                  0.540024
ring-type_p                  0.540469
stalk-surface-below-ring_k   0.573524
stalk-surface-above-ring_k   0.587658
odor_f                       0.623842
odor_n                       0.785557
p                            1.000000
```

Podemos observar que hay una gran variedad de correlaciones que van desde 0.016 hasta 0.78, lo cual es una correlacion fuerte. Debido a la gran cantidad de correlaciones fuertes haremos lo siguiente.
Para el preprocesamiento de datos, vamos a quedarnos con 2 data-sets de variables independientes:

- Data-set con solamente las caracteristicas mas correlacionadas (0.5 o mas en indice de correlacion)
- Data-set con todas las caracteristicas

Como se hizo con el ejercicio anterior, esto nos va a dar mas espacio de experimentacion cuando empecemos a afinar nuestro modelo.

Despues toca escalar todos los datos para empezar a experimentar con los modelos

```
escalador_all = StandardScaler()
escalador_all.fit(mushroom_x_all)
mushroom_x_all_scaled = escalador_all.transform(mushroom_x_all)

escalador_corr = StandardScaler()
escalador_corr.fit(mushroom_x_corr)
mushroom_x_corr_scaled = escalador_corr.transform(mushroom_x_corr)
```

Por ultimo toca modularizar los datos para tener 3 bloques de datos, aunque el bloque de pruebas sea un subconjunto del bloque de validacion:

```
train_x_all, test_x_all, train_y_all, test_y_all = train_test_split(mushroom_x_all_scaled, mushroom_y, random_state=0)

train_x_corr, test_x_corr, train_y_corr, test_y_corr = train_test_split(mushroom_x_corr_scaled, mushroom_y, random_state=0)
```

Prueba con modelo predisenado

Empezemos probando el arbol de decision que fue preparado para el reto pasado (El alfa fue copiado de forma manual ya que al fin y al cabo, el mejor arbol de decision que se utilizo fue ese)

```
# Arbol con todos los datos
decision_tree_all = DecisionTreeClassifier(random_state=0,
                                                                                    max_depth=12,
                                                                                    ccp_alpha=0.00088584501076195)
decision_tree_all.fit(train_x_all, train_y_all)

# Arbol con datos correlacionados
decision_tree_corr = DecisionTreeClassifier(random_state=0,
                                            max_depth=12,
                                            ccp_alpha=0.00088584501076195)
decision_tree_corr.fit(train_x_corr, train_y_corr)
```

```
Arbol de decision podado (todos los datos) ======================================
Puntaje de entrenamiento: 0.9993435089446906
Puntaje de validacion: 1.0
Alfa: 0.00088584501076195
=========================================================


Arbol de decision podado (datos correlacionados) ======================================
Puntaje de entrenamiento: 0.9789922862301002
Puntaje de validacion: 0.9763663220088626
Alfa: 0.00088584501076195
=========================================================
```

Bueno, eso no lo esperaba... Tener un modelo con un 100% de precision siempre es bienvenido, sin embargo, esta se supone que era la oportunidad para explicar porque aplicar siempre lo mismo a todos los problemas era malo y poder explicar las diferentes metricas para afinar un modelo, como sesgo, varianza, etc, pero esto es un dato atipico.

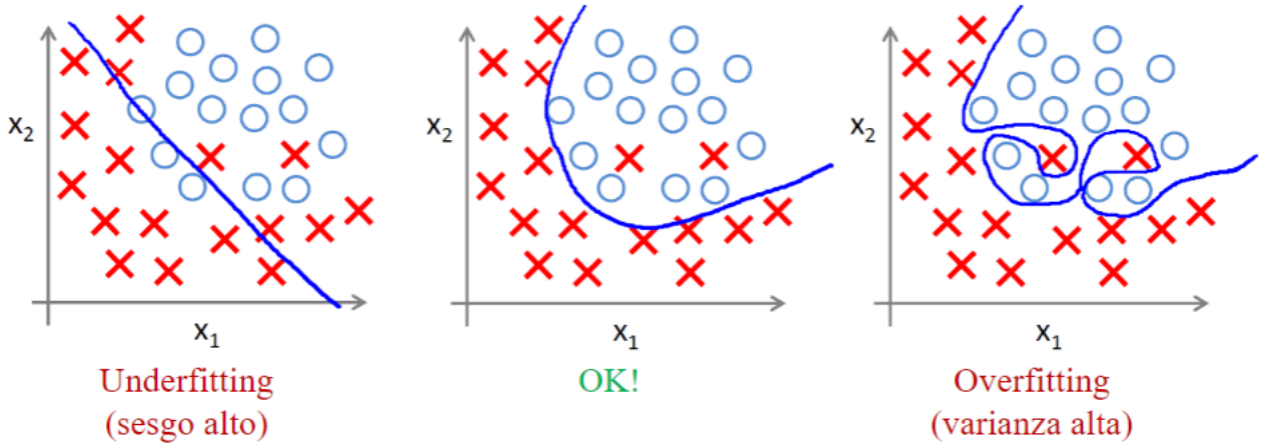
Hagamos esto, haremos un arbol de decision que tenga un puntaje muy malo y de ahi lo empezaremos a afinar...

```
# Arbol con todos los datos
decision_tree_2_all = DecisionTreeClassifier(random_state=0,
                                                                                    ccp_alpha=10)
decision_tree_2_all.fit(train_x_all, train_y_all)

# Arbol con datos correlacionados
decision_tree_2_corr = DecisionTreeClassifier(random_state=0,
                                                                                     ccp_alpha=10)
decision_tree_2_corr.fit(train_x_corr, train_y_corr)
```

```
Arbol de decision podado (todos los datos) ====================================
Puntaje de entrenamiento: 0.5164943377646479
Puntaje de validacion: 0.5224027572624323
==================================================

Arbol de decision podado (datos correlacionados) ==================================
Puntaje de entrenamiento: 0.5164943377646479
Puntaje de validacion: 0.5224027572624323
==================================================
```

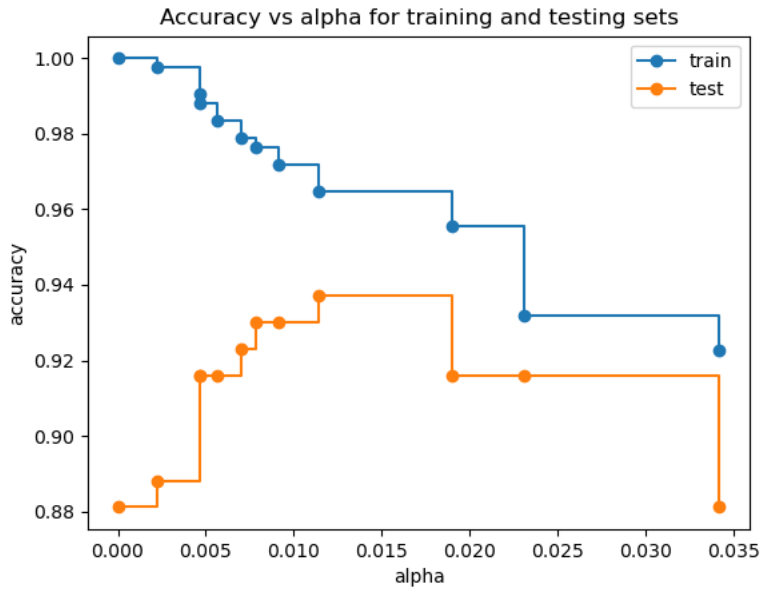
Este arbol de decision tiene un puntaje muy bajo, tanto en el entrenamiento como en la validacion. Esto es por 3 razones:

- Sesgo alto: los valores predecidos estan muy lejos de los valores verdaderos
- Varianza baja: el modelo no se mueve lo suficiente para poder predecir de forma correcta los valores
- Underfitting: el modelo no es capaz de generalizar debido a su baja complejidad
  En realidad, estas 3 caracteristicas se deben a que el modelo es muy simple.
  Resolver esto es muy sencillo, solamente hay que aumentarle la complejidad al modelo



¿Por que nuestro modelo es simple?

La razon principal es porque el valor alpha en un arbol de deicision le indica al modelo cuando *podar* ramas, lo que significa que el modelo no alcanza una convergencia porque no se le permitio crecer mas



¿Cuales son los hiperparametros que puedo utilizar para subir la complejidad del arbol de decision?

Normalmente los arboles de decision tienden a ser tan complejos que datos nuevos que llegan para predecir los predice mal ya que el modelo se *memorizo* los datos de entrenamiento.
Para aumentar la complejidad de este modelo podemos dejar que crezca lo maximo que pueda quitandole el parametro `ccp_alpha`.
De la misma forma, quitandole el hiperparametro `max_depth` puede dejar que el arbol crezca lo que sea posible, lo cual tampoco es muy recomendable debido a los recursos computacionales que necesita.
Asimismo, quitandole el hiperparametro `min_samples_leaf` puede ayudar a que su complejidad aumente ya que este parametro limita la generacion de una hoja si es que no hay suficientes datos en ese nodo, lo cual no deja generar el arbol completo.

En resumen, si ambos puntajes son bajos, indica underfitting con las 3 caracteristicas nombradas, asi que el plan de accion es aumentar la complejidad del modelo.
Si el puntaje de entrenamiento es muy alto y el de validacion muy bajo, esto indica overfitting, haciendo referencia a que el modelo "memoriza" los datos de entrenamiento y cualquier registro nuevo no lo puede predecir de manera correcta ya que va a tender a dar una respuesta del mismo modulo de entrenamiento.
Cuando ocurre overfitting es necesario bajar la complejidad del modelo, asi como usar mas datos de entrenamiento para que pueda generalizar de mejor manera.

```
# Arbol con todos los datos
decision_tree_3_all = DecisionTreeClassifier(random_state=0)
decision_tree_3_all.fit(train_x_all, train_y_all)

# Arbol con datos correlacionados
decision_tree_3_corr = DecisionTreeClassifier(random_state=0)
decision_tree_3_corr.fit(train_x_corr, train_y_corr)
```

```
Arbol de decision completo (todos los datos) ======================================
Puntaje de entrenamiento: 1.0
Puntaje de validacion: 1.0
=========================================================


Arbol de decision completo (datos correlacionados) ======================================
Puntaje de entrenamiento: 0.9789922862301002
Puntaje de validacion: 0.9763663220088626
=========================================================
```

En esta ocasion en especifico, nuestro modelo llego a un puntaje del 100% en ambos modulos, lo cual indica que es basicamente un modelo perfecto No hizo falta afinarlo mas, pero esto no significa que vaya a pasar siempre

Comparacion entre modelos

Para acabar, vamos a compararlo con nuestro "primer" modelo de arbol de decision utilizando metricas de rendimiento y la matriz de confusion

```
# Funcion para sacar metricas de rendimiento
def metricas_rendimiento(matriz_confusion):
        exactitud = (matriz_confusion[0][0] + matriz_confusion[1][1]) / (
                                matriz_confusion[0][0] + matriz_confusion[0][1] + matriz_confusion[1][0] + matriz_confusion

        try:
                precision = matriz_confusion[0][0] / (matriz_confusion[0][0] + matriz_confusion[1][0])
        except:
                precision = 0

        exhaustividad = matriz_confusion[0][0] / (matriz_confusion[0][0] + matriz_confusion[0][1])

        try:
                puntaje_F1 = (2 * precision * exhaustividad) / (precision + exhaustividad)
        except:
                puntaje_F1 = 0

        return exactitud, precision, exhaustividad, puntaje_F1

# Listado de modelos para ciclar la obtencion de metricas y matrices de confusion

models = [['Arbol de decision 1 (Todos los datos)', 'all', decision_tree_2_all],
                  ['Arbol de decision 1 (Datos correlacionados)', 'corr', decision_tree_2_corr],
                  ['Arbol de decision 2 (Todos los datos)', 'all', decision_tree_3_all],
                  ['Arbol de decision 2 (Datos correlacionados)', 'corr', decision_tree_3_corr]]

for trio in models:
        if trio[1] == 'all':
                conf_matrix = confusion_matrix(test_y_all, trio[2].predict(test_x_all))
        else:
                conf_matrix = confusion_matrix(test_y_corr, trio[2].predict(test_x_corr))

        acc, prec, recall, F1_score = metricas_rendimiento(conf_matrix)

        print("=============================================")
        print(f"Metricas de rendimiento para modelo de {trio[0]}")
        print("Matriz de confusion:")
        print(conf_matrix)
        print(f"Exactitud     : {acc}")
        print(f"Precision     : {prec}")
        print(f"Exhaustividad : {recall}")
```

```
        print(f"Puntaje F1    : {F1_score}")
        print("==========================================\n")
```

```
==========================================
Metricas de rendimiento para modelo de Arbol de decision 1 (Todos los datos)
Matriz de confusion:
[[1061    0]
 [ 970    0]]
Exactitud     : 0.5224027572624323
Precision     : 0.5224027572624323
Exhaustividad : 1.0
Puntaje F1    : 0.6862871927554981
==========================================


==========================================
Metricas de rendimiento para modelo de Arbol de decision 1 (Datos correlacionados)
Matriz de confusion:
[[1061    0]
 [ 970    0]]
Exactitud     : 0.5224027572624323
Precision     : 0.5224027572624323
Exhaustividad : 1.0
Puntaje F1    : 0.6862871927554981
==========================================
```

```
==========================================
Metricas de rendimiento para modelo de Arbol de decision 2 (Todos los datos)
Matriz de confusion:
[[1061    0]
 [   0  970]]
Exactitud     : 1.0
Precision     : 1.0
Exhaustividad : 1.0
Puntaje F1    : 1.0
==========================================


==========================================
Metricas de rendimiento para modelo de Arbol de decision 2 (Datos correlacionados)
Matriz de confusion:
[[1032   29]
 [  19  951]]
Exactitud     : 0.9763663220088626
Precision     : 0.9819219790675547
Exhaustividad : 0.9726672950047125
Puntaje F1    : 0.9772727272727273
==========================================
```

Aqui se puede observar que el mejor modelo sin duda es el Arbol de deicision 2 utilizando el data-set completo Considerando que la gran mayoria de las variables independientes aportaban informacion importante debido a su alta correlacion promedio, es normal que al utilizar todas las caracteristicas se obtenga un resultado mas preciso.

Conclusion

Aunque en este proyecto haya sido asi, es muy importante entender la idea de que no todos los modelos son para todas las circunstancias y que es importante saber como afinar y reparar modelos cuyo rendimiento no es el que queremos. Conceptos como *sesgo*, *overfitting* y *afinacion de modelos* son cosas que tenemos que tener muy presentes para ser un verdadero cientifico de datos o ingeniero en Machine Learning.

Mejoras a partir de la retroalimentacion

- Creacion de documentacion en README.md
- Creacion de documentacion en .pdf