Taller sesión 9

Aprendizaje de máquina

Para empezar, veamos unas definiciones importantes

¿Qué es el aprendizaje de máquina supervisado?

Los sistemas de aprendizaje de máquina supervisado aprenden a partir de datos cómo combinar entradas (comunmete llamados características) para producir predicciones útiles sobre datos nunca antes vistos.

Exploremos la terminología básica del aprendizaje automático.

```
[]
Ninguna = None # Celda de ayuda para evitar errores
```

---

1. Etiquetas

Una etiqueta es el valor que estamos prediciendo, por ejemplo, la variable `Y` en el caso de una regresión lineal simple dado el modelo:

$$mX+b=Y$$

La etiqueta podría ser el precio futuro de un producto, el tipo de animal que se muestra en una imagen, el significado de un clip de audio o simplemente cualquier cosa que sea de interes de estudio.

2. Atributos

Un atributo es una variable de entrada, por ejemplo, la variable `X` en la regresión lineal simple, dado el mismo modelo:

$$mX+b=Y$$

Un proyecto de aprendizaje automático simple podría usar un solo atributo, mientras que otro más sofisticado podría usar millones de atributos, especificados como:

En el ejemplo del detector de spam, los atributos podrían incluir los siguientes:

- palabras en el texto del correo electrónico
- dirección del remitente
- hora del día a la que se envió
- presencia de la frase “un truco increíble” en el correo electrónico

3. Ejemplos

Un ejemplo es una instancia de datos en particular, `x`. (La `x` se coloca en negrita para indicar que es un vector). Los ejemplos se dividen en dos categorías:

- ejemplos etiquetados
- ejemplos sin etiqueta

Un ejemplo etiquetado incluye tanto atributos como etiqueta. Esto significa lo siguiente:

Ejemplos con etiqueta: {características, etiqueta}: (x, y)

Los ejemplos etiquetados se usan para entrenar el modelo. En un ejemplo de detector de spam, los ejemplos etiquetados serían los correos electrónicos individuales que los usuarios marcaron explícitamente como “es spam” o “no es spam”.

Por ejemplo, en la siguiente tabla se muestran 5 ejemplos etiquetados de un conjunto de datos que contiene información sobre los precios de vivienda en California:

Años de la casa	**Total de habitaciones**	**Total de cuartos**	**Valor medio**
Atributo	Atributo	Atributo	Etiqueta
15	5,612	1,283	66,900
19	7,650	1,901	80,100
17	720	174	85,700
14	1,501	337	73,400
20	1,454	326	65,500

Un ejemplo sin etiqueta contiene atributos, pero no la etiqueta. Esto significa lo siguiente:

Ejemplo sin etiqueta examples: {etiqueta, ?}: (x, ?)

Años de la casa	**Total de habitaciones**	**Total de cuartos**
Atributo	Atributo	Atributo
15	5,612	1,283
19	7,650	1,901
17	720	174
14	1,501	337
20	1,454	326

Una vez que el modelo se entrena con ejemplos etiquetados, ese modelo se usa para predecir la etiqueta en ejemplos sin etiqueta. En el detector de spam, los ejemplos sin etiqueta son correos electrónicos nuevos que las personas todavía no han etiquetado.

4. Modelos

Un modelo define la relación entre los atributos y la etiqueta. Por ejemplo, un modelo de detección de spam podría asociar de manera muy definida determinados atributos con “es spam”. Destaquemos dos fases en el ciclo de un modelo:

- Entrenamiento significa crear o aprender el modelo. Es decir, le muestras ejemplos etiquetados al modelo y permites que este aprenda gradualmente las relaciones entre los atributos y la etiqueta.
- Inferencia significa aplicar el modelo entrenado a ejemplos sin etiqueta. Es decir, usas el modelo entrenado para realizar predicciones útiles (y). Por ejemplo, durante la inferencia, puedes predecir el **Valor medio** para nuevos ejemplos sin etiqueta.

---

5. Regresión frente a clasificación

Un modelo de regresión predice valores continuos. Por ejemplo, los modelos de regresión hacen predicciones que responden a preguntas como las siguientes:

- ¿Cuál es el valor de una casa en California?
- ¿Cuál es la probabilidad de que un usuario haga clic en este anuncio?

Un modelo de clasificación predice valores discretos. Por ejemplo, los modelos de clasificación hacen predicciones que responden a preguntas como las siguientes:

- ¿Un mensaje de correo electrónico determinado es spam o no es spam?
- ¿Esta imagen es de un perro, un gato o un hámster?

---

6. Ejercicio Aprendizaje supervisado

Exploremos las opciones que aparecen a continuación.

Imagina que quieres desarrollar un modelo de aprendizaje automático supervisado para predecir si un determinado correo electrónico “es spam” o “no es spam”.

¿Cuáles de las siguientes afirmaciones son verdaderas?

---

1. Las palabras del encabezado de asunto serán buenas etiquetas.

[]

```
verdadero_o_falso_1_1 = Ninguna #@param ["Ninguna", "Verdadero", "F
also"]
```

```
print(verdadero_o_falso_1_1)
```

verdadero_o_falso_1_1:

NingunaVerdaderoFalso

None

Pista

Solución

---

2. Los correos electrónicos no marcados como “es spam” o “no es spam” son ejemplos sin etiqueta.

[]

```
verdadero_o_falso_1_2 = Ninguna #@param ["Ninguna", "Verdadero", "F
also"]
print(verdadero_o_falso_1_2)
```

verdadero_o_falso_1_2:

NingunaVerdaderoFalso

None

Pista

Solución

---

3. Algunas etiquetas pueden ser poco confiables.

[]

```
verdadero_o_falso_1_3 = Ninguna #@param ["Ninguna", "Verdadero", "F
also"]
print(verdadero_o_falso_1_3)
```

verdadero_o_falso_1_3:

NingunaVerdaderoFalso

None

Solución

4. Se usarán ejemplos sin etiqueta para entrenar el modelo.

[ ]

```
verdadero_o_falso_1_4 = Ninguna #@param ["Ninguna", "Verdadero", "F
also"]
print(verdadero_o_falso_1_4)
```

verdadero_o_falso_1_4:

NingunaVerdaderoFalso

None

Pista

Solución

7. Ejercicio Atributos y etiquetas

Explora las opciones que aparecen a continuación.

Imagina que una tienda de calzado en línea quiere crear un modelo de aprendizaje de máquina supervisado que proporcione a los usuarios recomendaciones personalizadas sobre zapatos. Esto significa que el modelo recomendará determinados pares de zapatos a Martín y otros pares de zapatos a Juana. ¿Cuáles de las siguientes afirmaciones son verdaderas?

1. La talla del calzado es un atributo útil.

[ ]

```
verdadero_o_falso_2_1 = Ninguna #@param ["Ninguna", "Verdadero", "F
also"]
```

```
print(verdadero_o_falso_2_1)
```

verdadero_o_falso_2_1:

NingunaVerdaderoFalso

None

Pista

Solución

2. Los clics del usuario en la descripción de un zapato son una etiqueta útil.

[]

```
verdadero_o_falso_2_2 = Ninguna #@param ["Ninguna", "Verdadero", "F
also"]
print(verdadero_o_falso_2_2)
```

verdadero_o_falso_2_2:

NingunaVerdaderoFalso

None

Pista

Solución

3. La belleza del calzado es un atributo útil.

[]

```
verdadero_o_falso_2_3 = Ninguna #@param ["Ninguna", "Verdadero", "F
also"]
print(verdadero_o_falso_2_3)
```

verdadero_o_falso_2_3:

NingunaVerdaderoFalso

None

Pista

Solución

4. Los zapatos que le fascinan a un usuario son una etiqueta útil.

[]

```
verdadero_o_falso_2_4 = Ninguna #@param ["Ninguna", "Verdadero", "F
also"]

print(verdadero_o_falso_2_4)
```

verdadero_o_falso_2_4:

NingunaVerdaderoFalso

None

Pista

Solución

Objetivos del machine learning

Exploraremos algunos usos cotidianos del machine learning, y los abordaremos a partir de ejemplos.

Regresión lineal

Se sabe que los grillos cantan con mayor frecuencia en los días de más calor. Durante décadas, entomólogos profesionales y aficionados han catalogado datos sobre la cantidad de cantos por minuto y la temperatura. Para tu cumpleaños, la tía Ruth te regaló su amada base de datos sobre grillos y te invita a que aprendas un modelo para predecir dicha relación.

En primer lugar, es necesario realizar una representación de los datos para examinarlos:

Figura 1. Cantos por minuto contra temperatura

Efectivamente, la representación muestra que la cantidad de cantos aumenta con la temperatura. ¿Es lineal la relación entre los cantos y la temperatura? Sí, ya que es posible dibujar una línea recta como la siguiente para representar dicha relación:

Figura 2. Una relación lineal

Si bien la línea no pasa perfectamente por cada punto, demuestra con claridad la relación entre la temperatura y los cantos por minuto para dichos puntos. Si aplicamos un poco de álgebra, podemos determinar esta relación de la siguiente manera:

donde:

$$Y=mX+b$$

- Y: es la temperatura en grados centígrados, correspondiente al valor que intentamos predecir.
- m: es la pendiente de la línea.
- X: es la cantidad de cantos por minuto, correspondiente al valor de nuestro atributo de entrada.
- b: es la intersección en Y.

Según las convenciones del aprendizaje automático, la ecuación para un modelo se escribirá de una forma un poco diferente:

donde:

$$Y=b+wX$$

- Y: es la etiqueta predicha (un resultado deseado).
- b: es la ordenada al origen (la intersección en y).
- W: es la ponderación del atributo X. La ponderación es el mismo concepto de la “pendiente” , que se indicó anteriormente.
- X: es un atributo (una entrada conocida).

Entrenamiento

Un modelo simplemente necesita aprender (determinar) valores correctos para todas las ponderaciones con base en los ejemplos etiquetados. En un aprendizaje supervisado, el algoritmo de un aprendizaje automático construye un modelo al examinar varios ejemplos e intentar encontrar un modelo que minimice la pérdida. Este proceso se denomina minimización del riesgo empírico.

La pérdida es una penalidad por una predicción incorrecta. Esto quiere decir que la pérdida es un número que indica qué tan incorrecta fue la predicción del modelo en un solo ejemplo. Si la predicción del modelo es perfecta, la pérdida es cero; de lo contrario, la pérdida es mayor. El objetivo de entrenar un modelo es encontrar la combinación de parametros y otros elemetos del modelo para lograr pérdidas bajas. Por ejemplo, la Figura 3 muestra un modelo al lado izquierdo con una pérdida alta, y al lado derecho un modelo con pérdida baja. Ten en cuenta lo siguiente con respecto a la imagen:

- La flecha roja representa la pérdida.
- La línea azul representa las predicciones

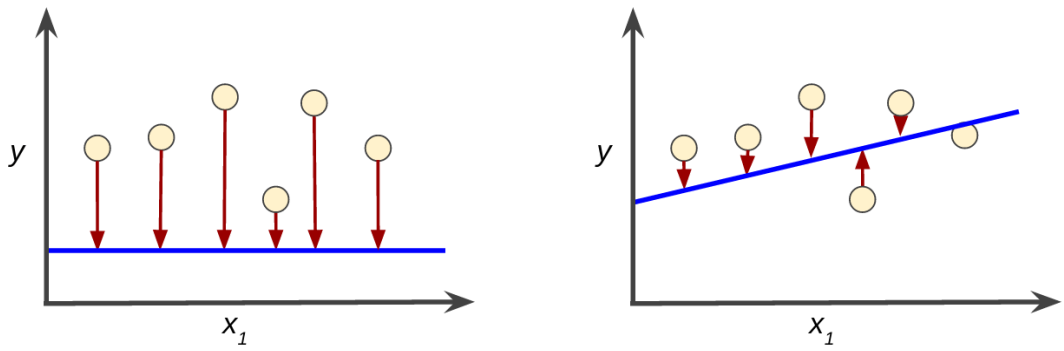


Figura 3. Pérdida alta en el modelo de la izquierda; pérdida baja en el modelo de la derecha. (flechas largas implican mas distancia y por tanto mas pérdida)

Ten en cuenta que las flechas rojas en la figura izquierda son mucho más largas que las de la figura derecha. Claramente, la línea azul en la figura de la derecha es un modelo de predicción mucho más acertado que la línea azul en la figura de la izquierda.

Tal vez te preguntes si puedes crear una función matemática (una función de pérdida) que sume las pérdidas individuales de una forma que tenga sentido.

Pérdida al cuadrado: Una función popular de pérdida

Los modelos de regresión lineal que se examinan aquí usan una función de pérdida llamada pérdida al cuadrado (también conocida como pérdida L2). A continuación, se muestra la pérdida al cuadrado para un único ejemplo:

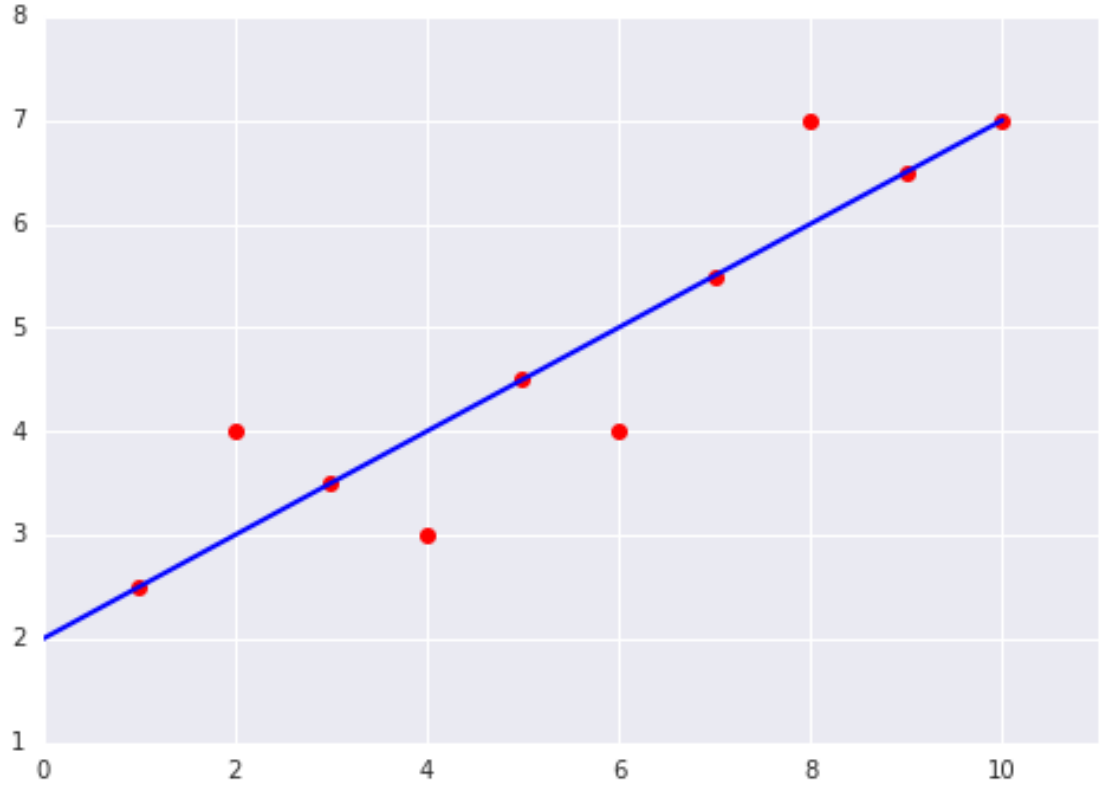
```
= El cuadrado de la diferencia entre la etiqueta real y la predicción
= (etiqueta real - predicción(x)) ^ 2
= (y - y') ^ 2
```

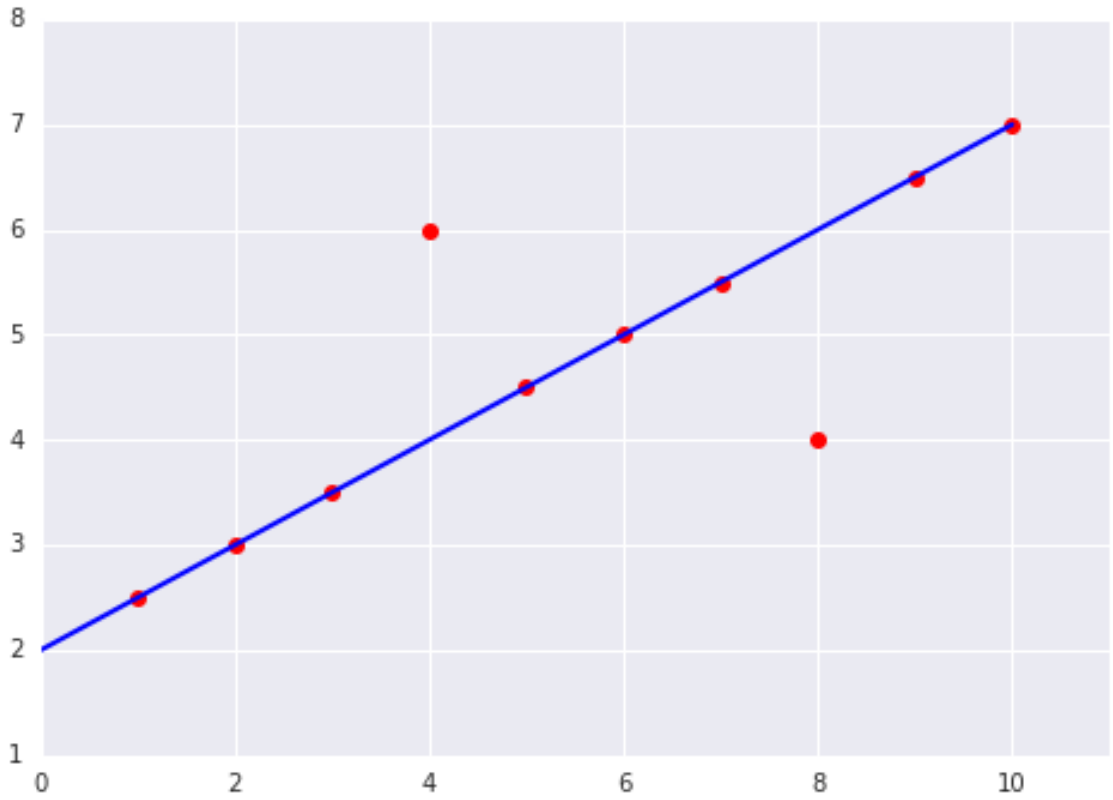
A lo anterior se le conoce como error cuadrático medio (ECM), que es el promedio de la pérdida al cuadrado de cada ejemplo. Para calcular el ECM, sumamos todas las pérdidas al cuadrado de los ejemplos individuales y, luego, lo dividimos por la cantidad de ejemplos:

Si bien ECM se usa comúnmente en el aprendizaje automático, no es la única función de pérdida práctica ni la mejor para todas las circunstancias.

Evaluación

Considera las siguientes representaciones:





¿Cuál de los dos conjuntos de datos que se muestran en las representaciones anteriores tiene el error cuadrático medio (ECM) más alto?

Pista

Solución

Generalización

La generalización hace referencia a la capacidad del modelo para adaptarse de manera adecuada a datos nuevos nunca antes vistos, obtenidos de la misma distribución que aquellos utilizados para crear el modelo.

Riesgos de overfitting

Este módulo se centra en la generalización. Para desarrollar algo de intuición sobre este concepto, observarás tres figuras. Imagina que cada punto en estas figuras representa la posición de un árbol en un bosque. Los dos colores tienen los siguientes significados:

- Los puntos azules representan árboles enfermos.
- Los puntos anaranjados representan árboles sanos.

Con eso en mente, echa un vistazo a la Figura 4:

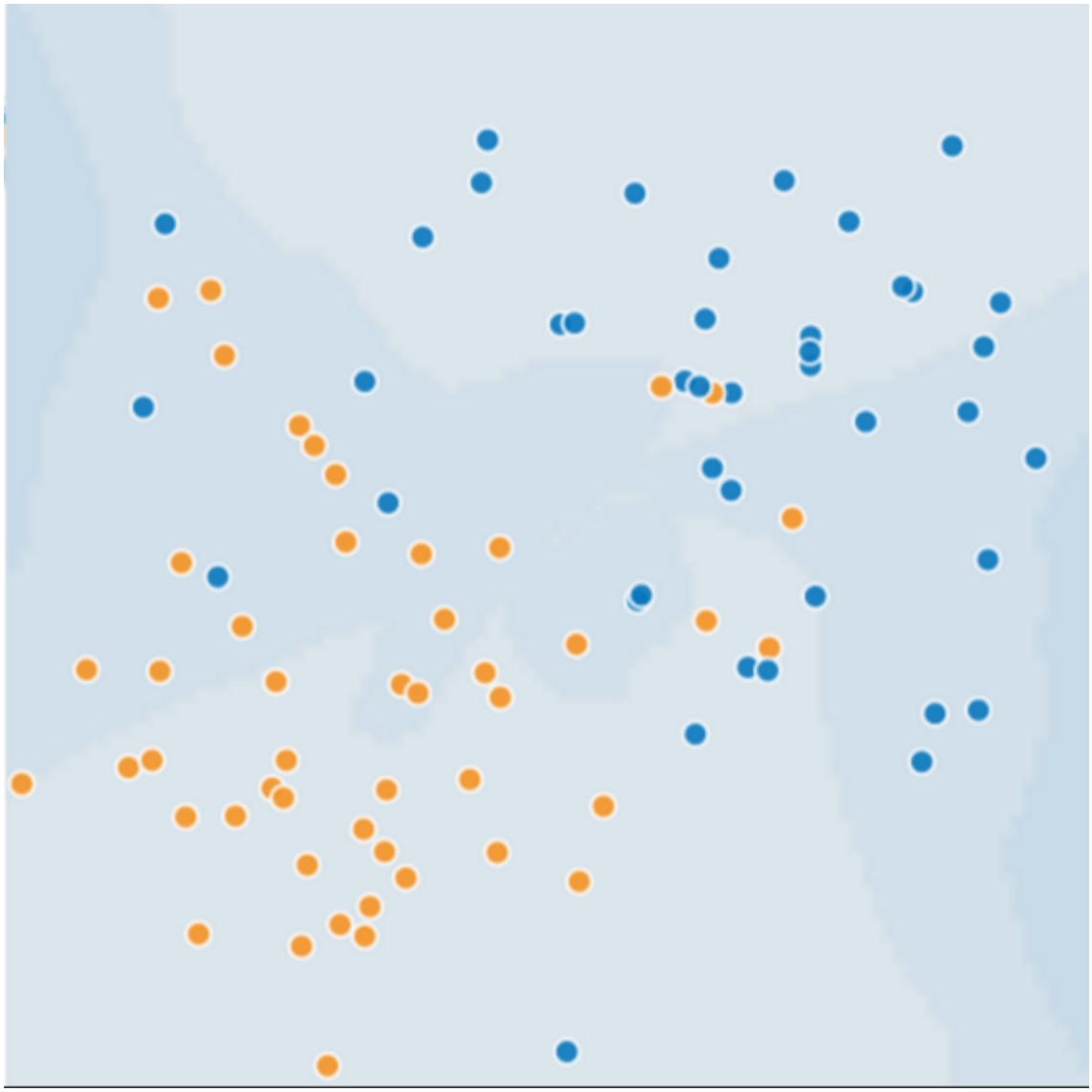


Figura 4. Árboles enfermos (azules) y sanos (anaranjados).

¿Puedes imaginar un buen modelo para predecir los árboles enfermos o sanos subsiguientes? Tómate un momento para dibujar mentalmente un arco que divida los puntos azules de los anaranjados, o enlaza mentalmente un lote de puntos azules o anaranjados.

¿Sigue siendo bueno el modelo cuando le añadimos más información real?

La Figura 5 muestra manchas naranjas y azules, las manchas quieren decir que allí se ubican árboles enfermos o sanos según su color. Luego se agregaron más al modelo y resultó que el modelo se adaptó de manera muy deficiente a los datos nuevos. Observa que el modelo categorizó mal muchos de los datos nuevos (Hay muchos árboles enfermos en zonas de supuestos árboles sanos y viceversa).

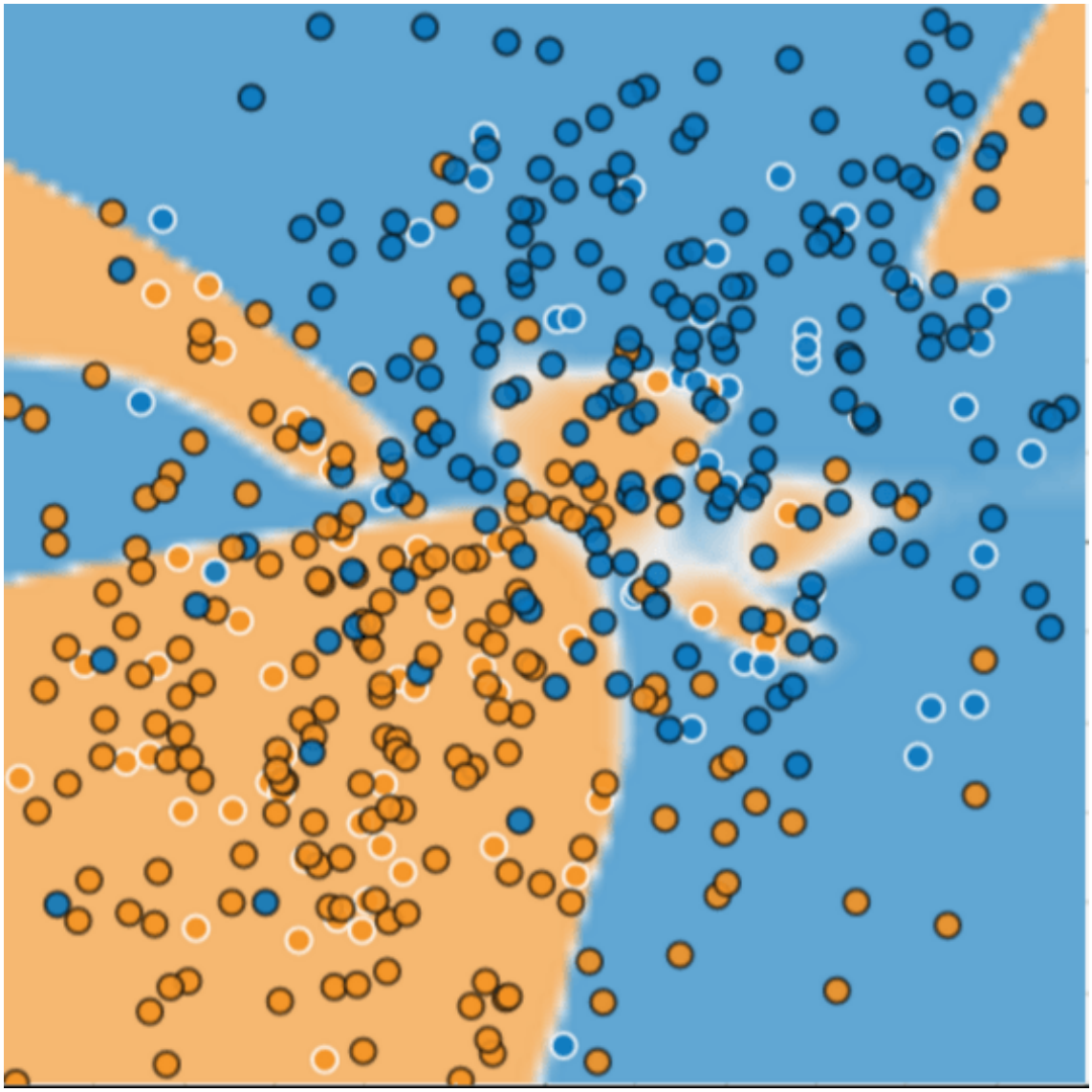


Figura 5. El modelo no se desempeñó bien al predecir datos nuevos.

El modelo que se muestra en la Figuras 3 hace *overfitting* con los datos con los que se entrenó. Un modelo con *overfitting* o sobreajustado obtiene una pérdida baja durante el entrenamiento, pero no se desempeña bien al predecir datos nuevos. Si un modelo se adapta bien a la muestra actual, ¿cómo podemos confiar en que realizará buenas predicciones sobre los datos nuevos? Como verás más adelante, el sobreajuste se genera al desarrollar un modelo más complejo que lo necesario. La presión fundamental del aprendizaje automático está en el ajuste correcto de nuestros datos, pero también en el ajuste de los datos de la manera más simple posible.

El objetivo del aprendizaje automático es realizar buenas predicciones sobre datos nuevos obtenidos de una distribución probablemente verdadera (oculta). Lamentablemente, el modelo no puede ver toda la verdad; este solo puede tomar una muestra de un conjunto de datos de entrenamiento. Si un modelo se adapta bien a los ejemplos actuales, ¿cómo podemos confiar en que también realizará buenas predicciones sobre los ejemplos nunca antes vistos?

Guillermo de Ockham, un fraile y filósofo del siglo XIV, amaba la simplicidad. Creía que los científicos debían preferir las fórmulas o teorías más simples en lugar de aquellas más complejas. Para expresar la navaja de Ockham en términos de aprendizaje automático:

"Cuanto menos complejo sea un modelo de AA, más probable será que un buen resultado empírico no se deba simplemente a las peculiaridades de la muestra."

En la actualidad, hemos formalizado la navaja de Ockham en los campos de la teoría del aprendizaje estadístico y la teoría del aprendizaje computacional. Estos campos han desarrollado límites de generalización, es decir, una descripción estadística de la capacidad de un modelo para generalizar sobre datos nuevos en función de factores como los siguientes:

- la complejidad del modelo
- el rendimiento del modelo con respecto a los datos de entrenamiento

Si bien el análisis teórico ofrece garantías formales en supuestos idealizados, esos límites pueden ser difíciles de aplicar en la práctica.

Un modelo de aprendizaje automático tiene como objetivo realizar buenas predicciones sobre datos nuevos nunca antes vistos. Pero, si desarrollas un modelo a partir de tu conjunto de datos, ¿cómo obtendrías los datos nunca antes vistos? Una forma es dividir el conjunto de datos en dos subconjuntos:

- Conjunto de entrenamiento: Un subconjunto para entrenar un modelo.
- Conjunto de prueba: Un subconjunto para probar el modelo.

Un buen rendimiento en el conjunto de prueba es un indicador útil de buen rendimiento en los datos nuevos en general, suponiendo lo siguiente:

- El conjunto de prueba es lo suficientemente grande.
- No haces trampa usando el mismo conjunto de prueba una y otra vez.

Las condiciones del machine lerarning supervisado

Las siguientes tres suposiciones básicas guían la generalización:

- Los ejemplos se obtienen independiente e idénticamente (i.i.d) de manera aleatoria de la distribución. En otras palabras, los ejemplos no se influyen entre sí. (Una explicación alternativa: i.i.d. es una forma de hacer referencia a la aleatoriedad de las variables).
- La distribución es estacionaria, es decir, no cambia dentro del conjunto de datos.
- Los ejemplos se obtienen de particiones de la misma distribución.

En la práctica, a veces infringimos estas suposiciones. Por ejemplo:

- Considera un modelo que elige los anuncios para mostrar. La suposición de i.i.d. se infringiría si, en parte, el modelo basara su elección en función de los anuncios que el usuario visualizó anteriormente.
- Considera un conjunto de datos que contenga la información de ventas minoristas de un año. Las compras de los usuarios cambian todas las temporadas, lo cual infringiría la estacionariedad.

Cuando sabemos que se infringe alguna de las tres suposiciones básicas anteriores, debemos prestar mucha atención a las métricas.

Conjuntos de entrenamiento y prueba

Un conjunto de prueba es un conjunto de datos que se usa para evaluar un modelo desarrollado a partir de un conjunto de entrenamiento.

Separación de datos

En el módulo anterior, se presentó la idea de dividir el conjunto de datos en dos subconjuntos:

- Conjunto de entrenamiento: Un subconjunto para entrenar un modelo.
- Conjunto de prueba: Un subconjunto para probar el modelo entrenado.

Imagina dividir el único conjunto de datos de la siguiente manera:

Figura 6. División de un único conjunto de datos en un conjunto de entrenamiento y uno de prueba.

Asegúrate de que tu conjunto de prueba reúna las siguientes dos condiciones:

- Que sea lo suficientemente grande como para generar resultados significativos desde el punto de vista estadístico.
- Que sea representativo de todo el conjunto de datos. En otras palabras, no elijas un conjunto de prueba con características diferentes al del conjunto de entrenamiento.

Si suponemos que el conjunto de prueba reúne estas dos condiciones, tu objetivo es crear un modelo que generalice los datos nuevos de forma correcta. Nuestro conjunto de prueba sirve como proxy para los datos nuevos. Por ejemplo, considera la siguiente figura. Observa que el modelo aprendido para los datos de entrenamiento es muy simple. Este modelo no hace un trabajo perfecto. Algunas predicciones son incorrectas. Sin embargo, este modelo funciona de la misma manera tanto en los datos de prueba como en los de entrenamiento. En otras palabras, este modelo simple no sobreajusta los datos de entrenamiento.

Figura 7. Validación del modelo entrenado con los datos de prueba.

Nunca uses los datos de prueba para el entrenamiento. Si ves resultados sorpresivamente positivos en tus métricas de evaluación, es posible que estés usando los datos de prueba para el entrenamiento. Por ejemplo, tener una precisión alta puede ser un indicativo de que se filtraron datos de prueba en los de entrenamiento.

Por ejemplo, imagina un modelo que prediga si un correo electrónico es spam, tomando como atributos el asunto, el cuerpo y la dirección de correo electrónico del remitente. Distribuimos los datos en conjuntos de entrenamiento y prueba en una proporción de 80 a 20. Después del entrenamiento, el modelo alcanza el 99% de precisión en ambos conjuntos. Esperamos una precisión menor en el conjunto de prueba, por lo que volvemos a analizar los datos y descubrimos que muchos de los ejemplos en este conjunto están duplicados en el conjunto de entrenamiento (arrastramos entradas duplicadas para el mismo spam de una base de datos de entrada antes de separar los datos). Involuntariamente, usamos algunos de los datos de prueba para el entrenamiento y, como resultado, no logramos medir de forma precisa de qué manera el modelo generaliza los datos nuevos.

Validación del modelo

Particionar un conjunto de datos en un conjunto de entrenamiento y uno de prueba te permite juzgar si un modelo determinado realizará generalizaciones eficaces sobre los datos nuevos. Sin embargo, usar solo dos particiones puede no ser suficiente cuando se realizan varias series de ajustes de los hiperparámetros.

En el módulo anterior, se presentó la partición de un conjunto de datos en un conjunto de entrenamiento y otro de prueba. Esta partición permite entrenar un conjunto de ejemplos y luego probar el modelo con un conjunto de ejemplos diferente. Con dos particiones, el flujo de trabajo podría verse de la siguiente manera:

Figura 8. ¿Un flujo de trabajo posible?

En la figura 8, “Ajustar el modelo” significa modificar cualquier aspecto que puedas imaginar del modelo, desde cambiar la tasa de aprendizaje hasta agregar o quitar atributos, o diseñar un modelo completamente nuevo desde cero. Al final de este flujo de trabajo, elijes el modelo que mejor se desempeñe con respecto al conjunto de prueba.

La división del conjunto de datos en dos conjuntos es una buena idea, pero no constituye una panacea. Puedes reducir en gran medida las posibilidades de sobreajuste al particionar el conjunto de datos en los tres subconjuntos que se muestran en la siguiente figura:

Figura 9. División de un único conjunto de datos en tres subconjuntos.

Usa el conjunto de validación para evaluar los resultados del conjunto de entrenamiento. A continuación, usa el conjunto de prueba para verificar la evaluación después de que el modelo haya “pasado” el conjunto de validación.

En este flujo de trabajo mejorado, realiza lo siguiente:

1. Selecciona el modelo que mejor se desempeñe con el conjunto de validación.
2. Verifica el modelo con respecto al conjunto de prueba.

Este flujo de trabajo es más eficaz porque crea menos exposiciones al conjunto de prueba.

Sugerencia: Los conjuntos de prueba y de validación se “desgastan” con el uso repetido. Esto significa que cuanto más usas los mismos datos para tomar decisiones sobre la configuración de hiperparámetros u otras mejoras del modelo, menos seguridad tendrás de que esos resultados puedan realizar generalizaciones sobre datos nuevos nunca antes vistos. Ten en cuenta que los conjuntos de validación suelen desgastarse más lentamente que los conjuntos de prueba. Si es posible, una buena idea es recopilar más datos para “actualizar” el conjunto de prueba y el de validación. Comenzar desde cero es un gran restablecimiento.

Exploración de datos y aprendizaje

Clasificación, agrupación y predicciónes sobre los juegos olímpicos

---

0. Librerías y funciones

A continuación se cargan cada una de las librerías necesarias para este taller

---

[ ]

```
import numpy as np
from numpy import array
from numpy import argmax
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
import pylab as pl
from sklearn.neural_network import MLPClassifier, MLPRegressor
from sklearn.model_selection import TimeSeriesSplit
```

```
from sklearn.metrics import mean_absolute_error, mean_squared_error,
mean_squared_log_error
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestClassifier, RandomForestRegr
essor
from sklearn.model_selection import ShuffleSplit, GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform as sp_randfloat
from scipy.stats import randint as sp_randint
import scipy
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import silhouette_score
def configure_plotly_browser_state():
  import IPython
  display(IPython.core.display.HTML('''
        <script src="/static/components/requirejs/require.js"></scr
ipt>
        <script>
          requirejs.config({
            paths: {
              base: '/static/base',
              plotly: 'https://cdn.plot.ly/plotly-
latest.min.js?noext',
            },
          });
        </script>
        '''))
```

1. Datos

En esta sección se cargan los datos con los que vamos a trabajar

[]

```
#Datos de https://www.kaggle.com/heesoo37/120-years-of-olympic-
history-athletes-and-results
#atletas
url1 = "https://drive.google.com/uc?export=download&id=1kFJ5ZKCxLSW
U9v-_p7T25-t1-azpANTL"
#codigo NOC National Olimpic Comité
url2 = "https://drive.google.com/uc?export=download&id=1UDqJWKAzpQf
782X83w9jLoQksUpf91G1"
```

```
df1 = pd.read_csv(url1)
df2 = pd.read_csv(url2)
```

1.1. Preguntas

- ¿Es posible identificar qué deporte practica una persona basado en sus características como peso, talla o nacionalidad?
- Analisis de regresión, ¿es posible predecir por país el número de medallas que ganarán?
- ¿Dadas las características del atleta se le puede catalogar como ganador de una medalla olímpica?

1.2. Descripción de los conjuntos de datos.

Tenemos un par de conjuntos de datos:

- `df1` contiene una base de los atletas que han participado en los juegos olímpicos, incluyendo si ganaron medalla y la comisión olímpica por la cual participaron.
- `df2` contiene para cada comisión la respectiva región/país al cual pertenecen.

[ ]

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 271116 entries, 0 to 271115
Data columns (total 15 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   ID      271116 non-null  int64  
 1   Name    271116 non-null  object 
 2   Sex     271116 non-null  object 
 3   Age     261642 non-null  float64
 4   Height  210945 non-null  float64
 5   Weight  208241 non-null  float64
 6   Team    271116 non-null  object 
 7   NOC     271116 non-null  object 
 8   Games   271116 non-null  object 
 9   Year    271116 non-null  int64  
 10  Season  271116 non-null  object 
 11  City    271116 non-null  object 
 12  Sport   271116 non-null  object 
 13  Event   271116 non-null  object 
```

```
 14  Medal   39783 non-null   object
dtypes: float64(3), int64(2), object(10)
memory usage: 31.0+ MB
```

[ ]

```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 230 entries, 0 to 229
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   NOC     230 non-null    object
 1   region  227 non-null    object
 2   notes   21 non-null     object
dtypes: object(3)
memory usage: 5.5+ KB
```

1.3. Limpieza de datos

Se revisan variables nulas y se detectan posibles errores

[ ]

```
print("datos nulos")

print(df1.isnull().sum())
```

```
datos nulos
ID          0
Name        0
Sex         0
Age      9474
Height  60171
Weight  62875
Team        0
NOC         0
Games       0
Year        0
Season      0
City        0
Sport       0
Event       0
Medal  231333
dtype: int64
```

[ ]

```
#En la columna medalla. se llenarán con "NGM" No Ganó Medalla para
poder trabajar con un valor fijo en esos casos
df1['Medal'].fillna('NGM', inplace = True)
```

1.4 Completar el conjunto de datos y codificar variables categóricas

Debido a que los comités olímpicos nacionales (NOC) pueden registrar a los atletas en diferentes equipos, se genera un DataFrame (llamado `df_merge`) maestro sobre el cual se generan codificaciones que se utilizarán en el resto del taller:

1. Se codificaron las medallas como binarias (1 si gana, 0 si no por tipo y en general)
2. Se codificó medallas como Enteros
3. Se codificó deportes como Enteros
4. Se codificó temporada como enteros y por consiguiente binario

---

[]

```
#Se hace el merge de los dataframes para obtener la columna región
(País)
df_merge = df1.merge(df2, left_on = 'NOC', right_on = 'NOC', how =
'left')
#se tratan las regiones vacias
df_merge['region'] = np.where(df_merge['NOC']=='SGP', 'Singapore',
df_merge['region'])
df_merge['region'] = np.where(df_merge['NOC']=='ROT', 'Refugee Olym
pic Athletes', df_merge['region'])
df_merge['region'] = np.where(df_merge['NOC']=='UNK', 'Unknown', df
_merge['region'])
df_merge['region'] = np.where(df_merge['NOC']=='TUV', 'Tuvalu', df_
merge['region'])
#se introduce unas binarias para descrbibir si ganó medalla 1 si no
 0
df_merge['Medalla_num'] = np.where(df_merge.loc[:,'Medal'] == 'NGM'
, 0, 1)
df_merge['Oro_num'] = np.where(df_merge.loc[:,'Medal'] == 'Gold', 1
, 0)
df_merge['Plata_num'] = np.where(df_merge.loc[:,'Medal'] == 'Silver
', 1, 0)
df_merge['Bronce_num'] = np.where(df_merge.loc[:,'Medal'] == 'Bronz
e', 1, 0)
# encode Medallas
medal_encoder_int = LabelEncoder()
```

```
medal_encoded = medal_encoder_int.fit_transform(array(df_merge.Medal))
df_merge['Medal']= medal_encoded
#print(np.unique(medal_encoded))
#Encode Sport
sport_encoder_int = LabelEncoder()
sport_encoded = sport_encoder_int.fit_transform(array(df_merge.Sport))
df_merge['Sport']= sport_encoded
#print(np.unique(sport_encoded)) #hay 65 deportes codificados
#print(sport_encoder_int.inverse_transform([1,2])) #por ejemplo los valores 1 y 2 fueron Alpine skiing y Aplinism :)
#Encode Season
season_encoder_int = LabelEncoder()
season_encoded = season_encoder_int.fit_transform(array(df_merge.Season))
df_merge['Season'] = season_encoded
#print(np.unique(Season_encoded)) #hay 2 temporadas ok
#print(Season_encoder_int.inverse_transform([0,1])) #los valores 0 y 1 son verano e invierno respectivamente
```

[ ]

```
sport_encoded
```

array([8, 32, 24, ..., 50, 12, 12])

Separación del conjunto de datos por país

Se quiso hacer una selección de datos agrupada por país para dar respuesta a las preguntas planteadas, posteriomente se hará énfasis solo en Colombia.

[ ]

```
df_paises = df_merge.drop(['Age','Weight','Height','ID','Sport'],axis =1 ).groupby(by=(['Year','region','Season'])).sum().reset_index()
#usa = df_paises[(df_paises['region']=='USA')].sort_values(by='Year')
#se Usará luego en la parte de descripcion de datos y el modelo de regresión
df_paises.head()
```

Separación del conjunto de datos por atleta

Se quiso obtener una separacion por atleta (ID) en la cual podamos observar, entre otras, el número de medallas y cuántas participaciones lleva a la última edición de los juegos.

[]

```
#recordemos que Medal está codificada, entonces la suma no tiene se
ntido
df_atletas_med = df_merge.drop(['Age','Weight','Height','Season','S
port','Year','Medal'],axis =1 ).groupby(by=('ID')).sum().reset_inde
x()
df_atletas_med.head()
```

[]

```
t = df_merge.drop_duplicates(subset='ID',keep='last') # datos unico
s de los atletas a la última participación, así sabremos cuanto pes
aba y su edad en los ultimos juegos
df_atletas = df_atletas_med.merge(t, left_on = 'ID', right_on = 'ID
', how = 'left') # se hace merge con el acumulado de medallería
df_atletas.head() # ahora sabemos el nombre de cada persona
# este df se utilizará luego para describir los mejores atletas en
terminos de su rendimiento
```

Separación del conjunto de datos por sexo

Debido a que los hombres y las mujeres no compiten juntos, se separará el conjunto de datos por sexo en:

1. df1_male
2. df1_female

[ ]
```
df1_male = df_merge[df_merge['Sex'] == 'M']

print(df1_male.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 196594 entries, 0 to 271115
Data columns (total 21 columns):
 #   Column      Non-Null Count   Dtype  
---  ------      --------------   -----  
 0   ID          196594 non-null  int64  
 1   Name        196594 non-null  object 
 2   Sex         196594 non-null  object 
 3   Age         187544 non-null  float64
 4   Height      143567 non-null  float64
 5   Weight      141470 non-null  float64
 6   Team        196594 non-null  object 
 7   NOC         196594 non-null  object 
 8   Games       196594 non-null  object 
 9   Year        196594 non-null  int64  
 10  Season      196594 non-null  int64  
 11  City        196594 non-null  object 
 12  Sport       196594 non-null  int64  
 13  Event       196594 non-null  object 
 14  Medal       196594 non-null  int64  
 15  region      196594 non-null  object 
 16  notes       4138 non-null    object 
 17  Medalla_num  196594 non-null  int64  
 18  Oro_num     196594 non-null  int64  
 19  Plata_num   196594 non-null  int64  
 20  Bronce_num  196594 non-null  int64  
dtypes: float64(3), int64(9), object(9)
memory usage: 33.0+ MB
None
```

[ ]
```
df1_female = df_merge[df_merge['Sex'] == 'F']

print(df1_female.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 74522 entries, 4 to 271110
Data columns (total 21 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   ID          74522 non-null  int64  
 1   Name        74522 non-null  object 
 2   Sex         74522 non-null  object 
 3   Age         74098 non-null  float64
```

```
 4  Height      67378 non-null  float64
 5  Weight      66771 non-null  float64
 6  Team        74522 non-null  object
 7  NOC         74522 non-null  object
 8  Games       74522 non-null  object
 9  Year        74522 non-null  int64
 10 Season      74522 non-null  int64
 11 City        74522 non-null  object
 12 Sport       74522 non-null  int64
 13 Event       74522 non-null  object
 14 Medal       74522 non-null  int64
 15 region      74522 non-null  object
 16 notes       901 non-null    object
 17 Medalla_num 74522 non-null  int64
 18 Oro_num     74522 non-null  int64
 19 Plata_num   74522 non-null  int64
 20 Bronce_num  74522 non-null  int64
dtypes: float64(3), int64(9), object(9)
memory usage: 12.5+ MB
None
```

Llenado de datos faltantes

Al obtener la información del conjunto de datos se observa que en las columnas o variables *Age*, *Height*, *Weight* y *Medal* existen valores nulos. Las variables se llenarán de la siguiente manera:

1. Los valores nulos de ***Age*** se llenarán con el valor aproximado de la media del conjunto de datos.
2. Los valores nulos de ***Height y Weight*** se llenarán con el valor de la media del conjunto de datos respectivamente.
3. Los valores nulos de ***Medal*** se llenarán con la variable categorica 'NGM' referente a 'No Gano Medalla' ya que no es un dato nulo realmente.

[ ]

```
#Para el conjunto de hombres
df1_male['Age'].fillna(np.round(df1_male['Age'].mean()), inplace= True)
df1_male['Height'].fillna(df1_male['Height'].mean(), inplace = True)
df1_male['Weight'].fillna(df1_male['Weight'].mean(), inplace = True)
df1_male.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 196594 entries, 0 to 271115
Data columns (total 21 columns):
```

```
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   ID          196594 non-null  int64
 1   Name        196594 non-null  object
 2   Sex         196594 non-null  object
 3   Age         196594 non-null  float64
 4   Height      196594 non-null  float64
 5   Weight      196594 non-null  float64
 6   Team        196594 non-null  object
 7   NOC         196594 non-null  object
 8   Games       196594 non-null  object
 9   Year        196594 non-null  int64
 10  Season      196594 non-null  int64
 11  City        196594 non-null  object
 12  Sport       196594 non-null  int64
 13  Event       196594 non-null  object
 14  Medal       196594 non-null  int64
 15  region      196594 non-null  object
 16  notes       4138 non-null    object
 17  Medalla_num  196594 non-null  int64
 18  Oro_num     196594 non-null  int64
 19  Plata_num   196594 non-null  int64
 20  Bronce_num  196594 non-null  int64
dtypes: float64(3), int64(9), object(9)
memory usage: 33.0+ MB
/usr/local/lib/python3.7/dist-packages/pandas/core/series.py:4536: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  downcast=downcast,
```

[ ]

```
#Para el conjunto de mujeres

df1_female['Age'].fillna(np.round(df1_female['Age'].mean()), inplace = True)

df1_female['Height'].fillna(np.round(df1_female['Height'].mean()), inplace = True)

df1_female['Weight'].fillna(np.round(df1_female['Weight'].mean()), inplace = True)

df1_female.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 74522 entries, 4 to 271110
Data columns (total 21 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   ID          74522 non-null  int64
 1   Name        74522 non-null  object
 2   Sex         74522 non-null  object
 3   Age         74522 non-null  float64
```

```
 4  Height     74522 non-null  float64
 5  Weight     74522 non-null  float64
 6  Team       74522 non-null  object
 7  NOC        74522 non-null  object
 8  Games      74522 non-null  object
 9  Year       74522 non-null  int64
 10 Season     74522 non-null  int64
 11 City       74522 non-null  object
 12 Sport      74522 non-null  int64
 13 Event      74522 non-null  object
 14 Medal      74522 non-null  int64
 15 region     74522 non-null  object
 16 notes      901 non-null    object
 17 Medalla_num 74522 non-null  int64
 18 Oro_num    74522 non-null  int64
 19 Plata_num  74522 non-null  int64
 20 Bronce_num  74522 non-null  int64
dtypes: float64(3), int64(9), object(9)
memory usage: 12.5+ MB
/usr/local/lib/python3.7/dist-packages/pandas/core/series.py:4536: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  downcast=downcast,
```

Se observa que el conjunto de datos se encuentra completo

---

2. Muestre los datos de forma descriptiva

Usaremos funciones de pandas para mostrar las características de los datos

---

[]

```
print ("Tenemos información disponible de", df_merge['Year'].nuniqu
e(),"versiones de los juegos, desde", df_merge['Year'].min(), "hast
a",df_merge['Year'].max())
```

```
Tenemos información disponible de 35 versiones de los juegos, desde 1896 hasta 2016
```

[]

```
print('Descripción para el conjunto de hombres: \n')

df1_male.describe()
```

[ ]

```
print('Descripción para el conjunto de mujeres: \n')

df1_female.describe()
```

[ ]

```
print('GRUPO DE HOMBRES')

print(df1_male.groupby('Medal')['Medal'].count())

print('\nEn el conjunto de datos de hombres existen:', df1_male['Name'].count(), 'registros de los cuales', df1_male.nunique()['Name'], 'son unicos')

print('\nGRUPO DE MUJERES')

print(df1_female.groupby('Medal')['Medal'].count())

print('\nEn el conjunto de datos existen de mujeres:', df1_female['Name'].count(), 'registros de los cuales', df1_female.nunique()['Name'], 'son unicos')
```

GRUPO DE HOMBRES
Medal
0 9524
1 9625
2 168064
3 9381
Name: Medal, dtype: int64

En el conjunto de datos de hombres existen: 196594 registros de los cuales 100979 son unicos

GRUPO DE MUJERES
Medal
0 3771
1 3747
2 63269
3 3735
Name: Medal, dtype: int64

En el conjunto de datos existen de mujeres: 74522 registros de los cuales 33808 son unicos

---

De todos los registros de ***hombres*** se tiene que:

1. Participaron 100979 Atletas
2. 168064 registros no ganaron medallas
3. 9526 registros ganaron medallas de bronce
4. 9381 registros ganaron medallas de plata

5. 9625 registros ganaron medallas de oro

De todos los registros de ***mujeres*** se tiene que:

1. Participarón 33808 Atletas
2. 63269 registros no ganarón medallas
3. 3771 registros ganarón medallas de Bronze
4. 3735 registros ganarón medallas de plata
5. 3747 registros ganarón medallas de oro

2.1. Países en el Top 10 y medallería histórica

[]

```
top10_pais = df_paises.groupby('region').sum().sort_values('Medalla
_num',ascending=False).head(10).drop(['Year','Medal','Season'],axis
=1)

top10_pais
```

[]

```
# Convirtamos los datos de la anterior tabla, en una interesante gr
áfica

g = top10_pais[['Oro_num', 'Plata_num','Bronce_num']].plot(kind = '
bar',stacked = True, figsize = (8,6), rot = 0)

g.set_xlabel('País')

g.set_ylabel('Medallas')
```

[]

```
# Veamos otra gráfica con la medallería histórica de los países en
el top 6 histórico

temporada_grafico = 0

print(season_encoder_int.inverse_transform([temporada_grafico]))

fig, ax = plt.subplots(figsize=(8,6))

for label in top10_pais.head(6).index:

  serie = df_paises[(df_paises['region']==label)&(df_paises['Season
']==temporada_grafico) ]
```

```
  serie.plot.line(x='Year',y = 'Medalla_num',label=label,ax=ax)
plt.legend()
#los picos mas abruptos en la grafica se deben a dos grandes Boicot
s que ocurrieron en la guerra fria años 80, 84
```

2.2 Top de Atletas

[]

```
#atletas TOP
print('Atletas con mayor número de medallas')
df_atletas.sort_values(by='Medalla_num_x',ascending=False).head(5)
```

[]

```
# el record mundial es Michael Phelps quien a sus 31 años ya habia
acumulado 28 medallas en los olimpicos de Rio de Janeiro 2016
# le sigue la ex atleta Larysa Semenivna, a los 29 acumuló 18 medal
las participando paor la URSS se retiró (de los olímpicos) en Tokio
 64
g = df_atletas.sort_values(by='Medalla_num_x',ascending=False)[['Br
once_num_x','Plata_num_x','Oro_num_x']].head(5).plot(kind = 'bar',s
tacked = True, figsize = (8,6), rot = 0)
g.set_xlabel('Participante')
g.set_ylabel('Medallas')
```

Realice el escalamiento o normalización que requieran los datos

[]

```
y_male = df1_male['Medal']
```

```
y_female = df1_female['Medal']
```

Particione los datos en entrenamiento y prueba

[]

```
#Conjunto de entrenamiento y prueba 70 - 30 para el grupo de hombre
s
df1_male_train, df1_male_test, y_male_sport_train, y_male_sport_tes
t = train_test_split(df1_male, y_male, test_size= 0.3, stratify= y_
male)
print('Tamaño del conjunto de entrenamiento y prueba respectivament
e del conjunto de hombres: ', df1_male_train.shape, df1_male_test.s
hape)
```

Tamaño del conjunto de entrenamiento y prueba respectivamente del conjunto de hombres: (137615, 21) (58979, 21)

[]

```
#Conjunto de entrenamiento y prueba 70 - 30 para el grupo de mujere
s
df1_female_train, df1_female_test, y_female_sport_train, y_female_s
port_test = train_test_split(df1_female, y_female, test_size= 0.3,
stratify = y_female)
print('Tamaño del conjunto de entrenamiento y prueba respectivament
e del conjunto de mujeres: ', df1_female_train.shape, y_female_spor
t_test.shape)
```

Tamaño del conjunto de entrenamiento y prueba respectivamente del conjunto de mujeres: (52165, 21) (22357,)

3. Métodos de aprendizaje

A continuacion encontrara distintos tipos de aprendizaje. Primero exploraremos el aprendizaje no supervisado haciendo un trabajo de agrupamiento o clustering. Luego revisaremos algunos ejemplo de aprendizaje supervisado como clasificación con redes neuronales y árboles aleatorios.

3.1 Clustering (Agrupamiento)

A continuación se muestra el uso de la metrica de la inercia del cluster y el coeficiente de silueta. Luego se comparan los dos para saber el número optimo de grupos (clusters)

Este paso se debe realizar incluso si sus datos estan supervisados para descubrir patrones dentro de sus datos.

Como puede observar en la siguiente celda se utilizar la altura *Height* y el peso *Weight* para mostrar las medallas.

[]

```
# Explorando que conjunto de datos se podria trabajar un cluster

from sklearn.cluster import KMeans, spectral_clustering, AgglomerativeClustering, DBSCAN

pl.figure(figsize= (10,6))

sns.scatterplot(data= df1_male, x='Height', y= 'Weight', hue= 'Medal')
```

[]

```
pl.figure(figsize= (10,6))

sns.scatterplot(data= df1_male, x= 'Height', y= 'Weight', hue= 'Medal')
```

Ajuste del modelo de k-means

A continuacion ajustaremos un modelo de k-means (k-medias) sobre los datos anteriormente mencionados

La celda siguiente define algunas funciones basicas pero puede saltarla.

Funciones

[]

```
# FUNCIONES NECESARIAS PARA DIBUJAR LOS CLUSTERS

def plot_cluster(X, cluster, z, km):
```

```
  cmap = plt.cm.plasma
  cmap((z*255./(cluster-1)).astype(int))

  for i in np.unique(z):
    cmap = plt.cm.bwr
    col = cmap((i*255./(cluster-1)).astype(int))
    Xr = X[z==i]
    plt.scatter(Xr[:,0], Xr[:,1], color=col, label="cluster %d"%i,
alpha=.5)
    plt.scatter(km.cluster_centers_[:,0], km.cluster_centers_[:,1],
marker="x", lw=5, s=200, color="black")
    plt.legend()
    plt.xlabel("Height")
    plt.ylabel("Weight")
def experiment_number_of_clusters(X, clustering, show_metric=True,
                                  plot_data=True, plot_centers=True
, plot_boundaries=False):
    plt.figure(figsize=(15,6))
    for n_clusters in range(2,10):
        clustering.n_clusters = n_clusters
        y = clustering.fit_predict(X)
        cm = plt.cm.plasma
        plt.subplot(2,4,n_clusters-1)
        plot_cluster_predictions(clustering, X, n_clusters, cm,
                                 plot_data, plot_centers, show_metr
ic)
def plot_cluster_predictions(clustering, X, n_clusters = None, cmap
 = plt.cm.plasma,
                             plot_data=True, plot_centers=True, sho
w_metric=False,
                             title_str=""):
    assert not hasattr(clustering, "n_clusters") or \
           (hasattr(clustering, "n_clusters") and n_clusters is not
 None), "must specify `n_clusters` for "+str(clustering)
    if n_clusters is not None:
        clustering.n_clusters = n_clusters
    y = clustering.fit_predict(X)
    # remove elements tagged as noise (cluster nb<0)
    X = X[y>=0]
    y = y[y>=0]
```

```
    if n_clusters is None:
        n_clusters = len(np.unique(y))
    if plot_data:
        plt.scatter(X[:,0], X[:,1], color=cmap((y*255./(n_clusters-
1)).astype(int)), alpha=.5)
    if plot_centers and hasattr(clustering, "cluster_centers_"):
        plt.scatter(clustering.cluster_centers_[:,0], clustering.cl
uster_centers_[:,1], s=150,  lw=3,
                    facecolor=cmap((np.arange(n_clusters)*255./(n_c
lusters-1)).astype(int)),
                    edgecolor="black")
    if show_metric:
        if hasattr(clustering, 'inertia_'):
          inertia = clustering.inertia_
        else:
          inertia = 0
        sc = silhouette_score(X, y) if len(np.unique(y))>1 else 0
        plt.title("k=%d, inertia=%.0f sc=%.3f"%(n_clusters, inertia
, sc)+title_str)
    else:
        plt.title("k=%d"%n_clusters+title_str)
    plt.axis("off")
    return
```

3.2.1 Para el conjunto de hombres

Vamos a crear un dataframe nuevo que tenga solo los datos de *Height* y *Weight*.

[]

```
#Separamos las variables que usaremos para el clustering
df1_male_train_sub = df1_male_train[['Height', 'Weight']].to_numpy(
)
df1_male_test_sub = df1_male_test[['Height', 'Weight']]. to_numpy()
```

Vamos a utilizar los datos y trataremos de encontrar tres grupos.

[]

```
#Entrenando el modelo para el conjunto de hombres con los datos de
entrenamiento
clusters = 3
```

```
km = KMeans(n_clusters = clusters)
km.fit(df1_male_train_sub)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

A continuación se muestra como se puede usar k-means para encontrar los distintos grupos dentro de los datos.

[ ]

```
#Imprimiendo el gráfico de clusters para el conjunto de hombres con
 los datos de prueba
z = km.predict(df1_male_test_sub)
print(z, z.shape)
print(pd.Series(z).value_counts())
print(km.cluster_centers_)
plot_cluster(df1_male_test_sub, clusters, z, km)
```

3.2.1.1 Inercia o distancia intracluster

En este punto vamos a calcular la medida de desempeño del modelo para diferentes cantidades de clusteres. Se puede observar que entre el mayor número de clusters, menor es la ***inercia*** del modelo

---

[ ]

```
#para distintos valores de cluster
sum_squared_distances = []
K= range(2, 20)
for k in K:
  km = KMeans(n_clusters= k)
  km.fit(df1_male_train_sub)
  sum_squared_distances.append(km.inertia_)
plt.plot(K, sum_squared_distances, 'bx-')
```

[ ]
```
print('Inercia del Modelo: ', km.inertia_)
```


Inercia del Modelo: 1620294.8961243234

3.2.2 Para el conjunto de mujeres

A continuación haremos el mismo ejercicio anterior pero para los datos de mujeres.

[ ]
```
#Separamos las variables que usaremos para el clustering
df1_female_train_sub = df1_female_train[['Height', 'Weight']].to_numpy()
df1_female_test_sub = df1_female_test[['Height', 'Weight']]. to_numpy()
```

[ ]
```
clusters = 3
km_female = KMeans(n_clusters = clusters)
km_female.fit(df1_female_train_sub)
```


KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)

[ ]
```
#Imprimendo el grafico de clusters para el conjunto de mujeres con los datos de prueba
z = km_female.predict(df1_female_test_sub)
print(z, z.shape)
print(pd.Series(z).value_counts())
print(km_female.cluster_centers_)
```

```
plot_cluster(df1_female_test_sub, clusters, z, km_female)
```

3.2.1.1 Inercia o distancia intracluster

Vamos a calcular ahora la inercia. Esto nos dice que tan de cerca los puntos dentro de un grupo.

[]

```
#para distintos valores de cluster
sum_squared_distances = []
K= range(2, 20)
for k in K:
  km_female = KMeans(n_clusters= k)
  km_female.fit(df1_female_train_sub)
  sum_squared_distances.append(km_female.inertia_)
plt.plot(K, sum_squared_distances, 'bx-')
```

[]

```
print('Inercia del Modelo: ', km_female.inertia_)
```

Inercia del Modelo: 558866.0286591987

5.2.2.2 EXPERIMENTO CON NUMERO DE CLUSTERS

Por favor no ejecute la siguiente linea.

La siguiente linea prueba distintos números de cluster, esta linea se demora mas de 1 hora entrenando.

[]

```
experiment_number_of_clusters(df1_male_train_sub, KMeans())
```

3.3 Etiquetas RED NEURONAL

A continuación usaremos una red neuronal para predecir el deporte al que pertenece un deportista dado

[]

```
# Función para visualizar de la superficie de decisión de un clasif
icador

#Función para visualizar la superficie de decisión de nuestro algor
itmo.

def plot_decision_region2(X, pred_fun):      #Función para visualiz
ar la superficie de decisión de nuestro algoritmo.
    min_x = np.min(X[:, 0])
    max_x = np.max(X[:, 0])
    min_y = np.min(X[:, 1])
    max_y = np.max(X[:, 1])
    min_x = min_x - (max_x - min_x) * 0.05
    max_x = max_x + (max_x - min_x) * 0.05
    min_y = min_y - (max_y - min_y) * 0.05
    max_y = max_y + (max_y - min_y) * 0.05
    x_vals = np.linspace(min_x, max_x, 100)
    y_vals = np.linspace(min_y, max_y, 100)
    XX, YY = np.meshgrid(x_vals, y_vals)
    grid_r, grid_c = XX.shape
    ZZ = np.zeros((grid_r, grid_c))
    for i in range(grid_r):
        for j in range(grid_c):
            ZZ[i, j] = pred_fun(XX[i, j], YY[i, j])
    pl.contourf(XX, YY, ZZ, 100, cmap = pl.cm.coolwarm, vmin= 1, vm
ax=3)
    pl.colorbar()
    pl.xlabel("x")
    pl.ylabel("y")


def gen_pred_fun(clf):
```

```
    def pred_fun(x1, x2):
        x = np.array([[x1, x2]])
        return clf.predict(x)[0]
    return pred_fun
def list_cm(cm,classes):     #función para generar de una forma más
 visual la matriz de confusión
    if len(cm)==2:
      cm.astype(int)
      row_0 =['','Valor','Verdadero']
      row_1 =['-',classes[0],classes[1]]
      row_2 =[classes[0],cm[0,0],cm[1,0]]
      row_3 =[classes[1],cm[0,1],cm[1,1]]
      table = zip(row_0,row_1, row_2, row_3)
      headers = ['', '', 'Valor', 'Predicho']
      return print(tabulate(table, headers=headers, floatfmt=".0f"))
    else:
      cm.astype(int)
      row_0 =['','Valor','Verdadero','']
      row_1 =['-',np.int(classes[0]),classes[1],classes[2]]
      row_2 =[classes[0],cm[0,0],cm[1,0],cm[2,0]]
      row_3 =[classes[1],cm[0,1],cm[1,1],cm[2,1]]
      row_4 =[classes[2],cm[0,2],cm[1,2],cm[2,2]]
      table = zip(row_0,row_1, row_2, row_3, row_4)
      headers = ['', '', 'Valor', 'Predicho', '']
      return print(tabulate(table, headers=headers, floatfmt=".0f"))
```

En la celda siguiente puede ver como creamos un modelo de red neuronal con cuatro capas. La primera capa cuenta con 300 neuronas, la segunda con 400, la tercera con 400, y la última capa con 80 neuronas. Se utiliza el optimizador **adam** en este caso.

Por favor no ejecute la siguiente linea.

[]

```
#entrenando el modelo
nn1= MLPClassifier(hidden_layer_sizes= (300, 300, 80), solver= 'ada
m', max_iter= 50)
nn1.fit(df1_male_train_sub, y_male_sport_train)
nn1.score(df1_male_train_sub, y_male_sport_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (50) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
0.8548777386186099
```

3.4 Etiquetas RANDOM FOREST

A continuación usaremos un modelo de Random Forest para tratar de clasificar el deporte de un deportista dado, según su altura y peso.

3.4.1 PARA EL CONJUNTO DE HOMBRES

[]

```
#Entrando el modelo random forest para predecir el deporte que jueg
a una persona con la caracteristica de peso y altura
```

[]

```
rf1_male = RandomForestClassifier(n_estimators= 1000, max_features=
 0.5)
rf1_male.fit(df1_male_train_sub, y_male_sport_train)
rf1_male.score(df1_male_train_sub, y_male_sport_train)
```

```
0.8565127348036188
```

El modelo de random forest permite obtener cual es la importancia de cada una de las características. En este caso las caracaterísticas son la altura y el peso.

[]

```
#importancia de las caracterisiticas
importance = rf1_male.feature_importances_
indices = np.argsort(importance)[::-1]
print("Importancia de características:")
columns = ['Height', 'Weight']
for f in range(df1_male_train_sub.shape[1]):
```

```
    print("Característica %s (%f)" % (columns[int(indices[f])], imp
ortance[indices[f]]))
```

Importancia de características:
Característica Weight (0.620585)
Característica Height (0.379415)

[]

```
plt.figure()
plt.title("Importancia de las características")
plt.bar(range(df1_male_train_sub.shape[1]), importance[indices],
       color="r", align="center")
xticks_labels = [columns[i] for i in indices]
plt.xticks(range(df1_male_train_sub.shape[1]), xticks_labels, rotat
ion=45)
plt.xlim([-1, df1_male_train_sub.shape[1]])
plt.show()
```

Para realizar el proceso de validación cruzada de una manera sistematica se puede usar RandomizerSearchCv que es un método que nos entrega Sklearn. En este caso el buscar las mejores combinaciones posibles de n_estimators y max_features

---

[]

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform
from scipy.stats import randint
```

[]

```
#Optimizacion de parametros
rf_opt = RandomForestClassifier()
param_dist = {'n_estimators': randint(4,1000), 'max_features': unif
orm()}
itera = 10
random_searh= RandomizedSearchCV(rf_opt, param_distributions= param
_dist, n_iter= itera, cv= 4)
```

En la siguiente celda ajustamos el modelo

[ ]

```
random_searh.fit(df1_male_train_sub, y_male_sport_train)
```

```
RandomizedSearchCV(cv=4, error_score=nan,
                   estimator=RandomForestClassifier(bootstrap=True,
                                                    ccp_alpha=0.0,
                                                    class_weight=None,
                                                    criterion='gini',
                                                    max_depth=None,
                                                    max_features='auto',
                                                    max_leaf_nodes=None,
                                                    max_samples=None,
                                                    min_impurity_decrease=0.0,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
                                                    min_weight_fraction_leaf=0.0,
                                                    n_estimators=100,
                                                    n_jobs...
                                                    verbose=0,
                                                    warm_start=False),
                   iid='deprecated', n_iter=10, n_jobs=None,
                   param_distributions={'max_features': <scipy.stats._distn_infrastructure.rv_frozen object
at 0x7fac40e69510>,
                                        'n_estimators': <scipy.stats._distn_infrastructure.rv_frozen object at
0x7fac41063e10>},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=0)
```

[ ]

```
results = pd.DataFrame(random_searh.cv_results_)

results

results = results[['param_n_estimators', 'param_max_features', 'mea
n_test_score']]

print(results.sort_values(by= 'mean_test_score', ascending= True ).
head(10))
```

```
  param_n_estimators param_max_features  mean_test_score
9                786          0.0973501         0.852901
0                322           0.215245         0.853017
6                244           0.988067         0.853025
3                472           0.843848         0.853039
8                576           0.441762         0.853105
5                848           0.616571         0.853119
```

4	887	0.417086	0.853126
2	768	0.718027	0.853163
1	854	0.977258	0.853192
7	996	0.933161	0.853235

[ ]

```
print('El mejor resultado para la clasificación para el conjunto de
 hombres se da con', random_searh.best_params_)
```

El mejor resultado para la clasificación para el conjunto de hombres se da con {'max_features': 0.9331607521366939, 'n_estimators': 996}

[ ]

```
#Usando los mejores parametros con el conjunto de prueba
rf1_male = RandomForestClassifier(n_estimators=94, max_features= 0.
55994423)
rf1_male.fit(df1_male_train_sub, y_male_sport_train)
```

RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
criterion='gini', max_depth=None,
max_features=0.55994423, max_leaf_nodes=None,
max_samples=None, min_impurity_decrease=0.0,
min_impurity_split=None, min_samples_leaf=1,
min_samples_split=2, min_weight_fraction_leaf=0.0,
n_estimators=94, n_jobs=None, oob_score=False,
random_state=None, verbose=0, warm_start=False)

[ ]

```
from sklearn.metrics import precision_score, recall_score, confusion_
matrix
print('Accuracy de clasificación usando ', random_searh.best_params
_, ': {}'.format(rf1_male.score(df1_male_test_sub, y_male_sport_tes
t)))
print('Error de clasificación usando ', random_searh.best_params_,
': {}'.format(1- rf1_male.score(df1_male_test_sub, y_male_sport_tes
t)))
print('Precision Macro usando ', random_searh.best_params_, ': {}'.
format(precision_score(y_male_sport_test, rf1_male.predict(df1_male
_test_sub), average= 'macro')))
print('Recall Score usando ', random_searh.best_params_, ': {}'.for
mat(recall_score(y_male_sport_test, rf1_male.predict(df1_male_test_
sub), average= 'macro')))
```

```
prediction_1 = rf1_male.predict(df1_male_test_sub)
cnf_matrix = confusion_matrix(y_male_sport_test, prediction_1)
print(cnf_matrix)
```

Accuracy de clasificación usando {'max_features': 0.9331607521366939, 'n_estimators': 996} : 0.8535919564590787
Error de clasificación usando {'max_features': 0.9331607521366939, 'n_estimators': 996} : 0.1464080435409213
Precision Macro usando {'max_features': 0.9331607521366939, 'n_estimators': 996} : 0.34192017107019534
Recall Score usando {'max_features': 0.9331607521366939, 'n_estimators': 996} : 0.2524208460531234
[[10 11 2829 7]
[5 18 2858 7]
[43 46 50310 21]
[6 11 2791 6]]

3.4.2 PARA EL CONJUNTO DE MUJERES

[]

```
rf1_female = RandomForestClassifier(n_estimators= 100, max_features
= 0.1)
rf1_female.fit(df1_female_train_sub, y_female_sport_train)
rf1_female.score(df1_female_train_sub, y_female_sport_train)
```

0.8520655612000383

[]

```
#importancia de las caracterisiticas
importance = rf1_female.feature_importances_
indices = np.argsort(importance)[::-1]
print("Importancia de características:")
columns = ['Height', 'Weight']
for f in range(df1_female_train_sub.shape[1]):
    print("Característica %s (%f)" % (columns[int(indices[f])], imp
ortance[indices[f]]))
```

Importancia de características:
Característica Weight (0.568839)
Característica Height (0.431161)

[]

```
plt.figure()
plt.title("Importancia de las características")
plt.bar(range(df1_female_train_sub.shape[1]), importance[indices],
       color="r", align="center")
xticks_labels = [columns[i] for i in indices]
plt.xticks(range(df1_female_train_sub.shape[1]), xticks_labels, rot
ation=45)
plt.xlim([-1, df1_female_train_sub.shape[1]])
plt.show()
```

[]

```
#Optimizacion de parametros
random_searh.fit(df1_female_train_sub, y_female_sport_train)
```

RandomizedSearchCV(cv=4, error_score=nan,
estimator=RandomForestClassifier(bootstrap=True,
ccp_alpha=0.0,
class_weight=None,
criterion='gini',
max_depth=None,
max_features='auto',
max_leaf_nodes=None,
max_samples=None,
min_impurity_decrease=0.0,
min_impurity_split=None,
min_samples_leaf=1,
min_samples_split=2,
min_weight_fraction_leaf=0.0,
n_estimators=100,
n_jobs...
verbose=0,
warm_start=False),

iid='deprecated', n_iter=10, n_jobs=None,
param_distributions={'max_features': <scipy.stats._distn_infrastructure.rv_frozen object at 0x7fac40e69510>,
'n_estimators': <scipy.stats._distn_infrastructure.rv_frozen object at 0x7fac41063e10>},
pre_dispatch='2*n_jobs', random_state=None, refit=True,
return_train_score=False, scoring=None, verbose=0)

[ ]

```
results = pd.DataFrame(random_searh.cv_results_)

results

results = results[['param_n_estimators', 'param_max_features', 'mean_test_score']]

print(results.sort_values(by= 'mean_test_score', ascending= True ).head(10))
```

	param_n_estimators	param_max_features	mean_test_score
0	27	0.466809	0.845299
9	572	0.715601	0.845950
2	301	0.821205	0.846065
8	261	0.428919	0.846161
5	855	0.448668	0.846180
3	778	0.746121	0.846200
7	855	0.628783	0.846200
6	741	0.319368	0.846276
1	868	0.851465	0.846315
4	486	0.44057	0.846315

[ ]

```
print('El mejor resultado para la clasificación en el conjunto de mujeres se da con', random_searh.best_params_)
```

El mejor resultado para la clasificación en el conjunto de mujeres se da con {'max_features': 0.44057040264758307, 'n_estimators': 486}

[ ]

```
#Usando los mejores parametros con el conjunto de prueba

rf1_female = RandomForestClassifier(n_estimators=94, max_features= 0.55994423)

rf1_female.fit(df1_female_train_sub, y_female_sport_train)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None,
                       max_features=0.55994423, max_leaf_nodes=None,
                       max_samples=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       n_estimators=94, n_jobs=None, oob_score=False,
                       random_state=None, verbose=0, warm_start=False)
```

[]

```
print('Accuracy de clasificación usando ', random_searh.best_params
_, ': {}'.format(rf1_female.score(df1_female_test_sub, y_female_spo
rt_test)))
print('Error de clasificación usando ', random_searh.best_params_,
': {}'.format(1- rf1_female.score(df1_female_test_sub, y_female_spo
rt_test)))
print('Precision Macro usando ', random_searh.best_params_, ': {}'.
format(precision_score(y_female_sport_test, rf1_female.predict(df1_
female_test_sub), average= 'macro')))
print('Recall Score usando ', random_searh.best_params_, ': {}'.for
mat(recall_score(y_female_sport_test, rf1_female.predict(df1_female
_test_sub), average= 'macro')))
prediction_1 = rf1.predict(df1_female_test_sub)
cnf_matrix = confusion_matrix(y_female_sport_test, prediction_1)
print(cnf_matrix)
```

4. CONCLUSIONES

CLUSTERING

Se realizo una clasificación de los atletas hombres y mujeres respectivamente en funcion de su peso y altura, se generarón 3 clusters para la clasificación de los atletas es decir:

Para el caso de los hombres

1. El primer cluster tendra centro en: ***Altura= 179.73 y Peso= 75.94***
2. El segundo cluster tendra centro en: ***Altura= 189.89 y Peso= 94.68***
3. El tercer cluster tendra centro en: ***Altura= 169.21 y peso= 62.73***

[]

```
print(km.cluster_centers_)
```

```
plot_cluster(df1_male_test_sub, clusters, z)
```

En el caso de las mujeres:

1. El primer cluster tendra centro en: ***Altura= 157.67 y Peso= 49.15***
2. El segundo cluster tendra centro en: ***Altura= 168.48 y Peso= 59.96***
3. El tercer cluster tendra centro en: ***Altura= 178.61 y peso= 73.73***

La ***Inercia*** de los modelos es mucho menor en el caso de las mujeres que de los hombres

---

[]
```
print(km_female.cluster_centers_)
```
```
plot_cluster(df1_female_test_sub, clusters, z)
```

RANDOM FOREST

En el caso del ***RANDOM FOREST*** se tiene que es dificil predecir el deporte que juega una persona basado en sus caracteristicas de peso y altura ya que usando estos dos para el conjunto de hombres se obtiene
un ***Accuracy*** de **85%** y para las mujeres del **84%.** Usando estos modelos se tiene que la importacia de las caracteristicas es

1. 'Peso' = 55%
2. 'Altura'= 45%
   Esto para el conjunto tanto de hombres como mujeres respectivamente

---

¿Qué otras técnicas cree que puede utilizar para mejorar su algoritmo?

Existen una gran cantidad de técnicas que no se abordaron en este taller.

Creación

Desarrollado por:

- Joseph Alejandro Gallego Mejia
- Daniela Martin
- Nelson Ospina
- Dennis Rodriguez

Dirigido y ajustado por:

- Joseph Alejandro Gallego Mejia

Bibliografía

1. Explicación tomada de: https://www.juanbarrios.com/google-machine-learning/