ARTIFICIAL INTELLIGENCE
PROFESSIONAL CERTIFICATE

I.6 Validación Cruzada

CAIPC™ Version 062021

CertiProf®

Concepto

En una misión anterior, aprendimos sobre la validación de entrenamiento/prueba, una técnica simple para probar la precisión de un modelo de aprendizaje automático en nuevos datos en los que el modelo no fue entrenado. En esta misión, nos centraremos en técnicas más robustas.

Para empezar, nos centraremos en la técnica de validación de retención (**holdout validation**), que implica:

- Dividir el conjunto de datos completo en 2 particiones:
  - Un conjunto de entrenamiento
  - Un conjunto de pruebas
  - Entrenar el modelo en el conjunto de entrenamiento
- Utilizar el modelo entrenado para predecir etiquetas en el conjunto de pruebas
- Calcular una métrica de error para comprender la eficacia del modelo
- Cambiar los conjuntos de entrenamiento y de prueba y repetirlo
- Promediar los errores

En la validación por retención (**holdout validation**), solemos utilizar una división 50/50 en lugar de la división 75/25 de la validación de entrenamiento/prueba. De este modo, eliminamos el número de observaciones como fuente potencial de variación en el rendimiento de nuestro modelo.

Figura 4. Descomposición de la prueba y el entreno

Empecemos por dividir el conjunto de datos en 2 mitades casi equivalentes.

Al dividir el conjunto de datos, no olvides establecer una copia del mismo utilizando **.copy()** para asegurarte de que no obtienes resultados inesperados más adelante. Si ejecutas el código localmente en Jupyter Notebook o Jupyter Lab sin **.copy()**, notarás lo que se conoce como una advertencia SettingWithCopy. Esto no impedirá que tu código se ejecute correctamente, pero te está haciendo saber que cualquier operación que estés haciendo está tratando de establecerse en una copia de un slice de un dataframe. Para asegurarse de no ver esta advertencia, asegúrese de incluir .copy() siempre que realice operaciones en un marco de datos.

Instrucciones

- Usar la función **numpy.random.permutation()** para barajar el orden de las filas en **dc_listings**
- Seleccionar las primeras 1862 filas y asignarlas a **split_one**
- Seleccionar las 1861 filas restantes y asignarlas a **split_two**

```
import numpy as np
import pandas as pd

dc_listings = pd.read_csv("dc_airbnb.csv")
stripped_commas = dc_listings['price'].str.replace(',',
'')
stripped_dollars = stripped_commas.str.replace('$', '')
dc_listings['price'] = stripped_dollars.astype('float')
shuffled_index =
np.random.permutation(dc_listings.index)
dc_listings = dc_listings.reindex(shuffled_index)

split_one = dc_listings.iloc[0:1862].copy()
split_two = dc_listings.iloc[1862:].copy()
```

Validación de la Retención

Ahora que hemos dividido nuestro conjunto de datos en 2 marcos de datos (dataframes), vamos a:

- Entrenar un modelo de k-nearest neighbors en la primera mitad
- Probar este modelo en la segunda mitad
- Entrenar un modelo de k-nearest neighbors en lsegunda mitad
- Probar este modelo en la segunda mitad

Instrucciones

- Entrenar un modelo de k-nearest neighbors model utilizando el algoritmo por defecto (**auto**) y utilizar el numero por defecto (**5**) que:
  - Utilize la columna de **accommodates** de **train_one** para entrenamiento
  - Pruebe en **test_one**
- Asigna el valor de RMSE resultante a **iteration_one_rmse.**
- Entrenar un modelo de k-nearest neighbors model utilizando el algoritmo por defecto (**auto**) y utilizar el numero por defecto (**5**) que:
  - Utilize la columna de **accommodates** de **train_two** para entrenamiento
  - Pruebe en **test_two**
- Asigna el valor de RMSE resultante a **iteration_two_rmse**
- Utiliza **numpy.mean()** para calcular la media de los 2 valores de RMSE y la asigna a **avg_rmse**

Soluciones

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error

train_one = split_one
test_one = split_two
train_two = split_two
test_two = split_one
# First half
model = KNeighborsRegressor()
model.fit(train_one[["accommodates"]],
train_one["price"])
test_one["predicted_price"] =
model.predict(test_one[["accommodates"]])
iteration_one_rmse =
mean_squared_error(test_one["price"],
test_one["predicted_price"])**(1/2)
```

```
# Second half
model.fit(train_two[["accommodates"]],
train_two["price"])
test_two["predicted_price"] =
model.predict(test_two[["accommodates"]])
iteration_two_rmse =
mean_squared_error(test_two["price"],
test_two["predicted_price"])**(1/2)

avg_rmse = np.mean([iteration_two_rmse,
iteration_one_rmse])

print(iteration_one_rmse, iteration_two_rmse, avg_rmse)
```

Validación Cruzada K-Fold

Si promediamos los dos valores de RMSE del último paso, obtenemos un valor de RMSE de aproximadamente **128,96**. La validación holdout es en realidad un ejemplo específico de una clase más amplia de técnicas de validación llamada **validación cruzada k-fold (K-fold cross-validation)**. Mientras que la validación holdout es mejor que la validación de entrenamiento/prueba porque el modelo no está sesgado repetidamente hacia un subconjunto específico de los datos, ambos modelos que se entrenan sólo utilizan la mitad de los datos disponibles. **La validación cruzada K-fold,** por otro lado, aprovecha una mayor proporción de los datos durante el entrenamiento mientras sigue rotando a través de diferentes subconjuntos de los datos para evitar los problemas de la validación entrenamiento/prueba.

Este es el algoritmo de la validación cruzada k-fold:

- Dividir el conjunto de datos en **k** particiones de igual longitud
  - Seleccionar **k-1** particiones como conjunto de entrenamiento y
  - Seleccionar la partición restante como conjunto de pruebas
- Entrenamiento del modelo en el conjunto de entrenamiento
- Utilizar el modelo entrenado para predecir etiquetas en el pliegue de prueba
- Calcular la métrica de error del pliegue de prueba
- Repetir todos los pasos anteriores **k-1** veces, hasta que cada partición se haya utilizado como conjunto de prueba para una iteración
- Calcular la media de los **k** valores de error

La validación cruzada es esencialmente una versión de la validación cruzada de k pliegues cuando k es igual a 2. Generalmente, se utilizan 5 o 10 pliegues para la validación cruzada de k pliegues. Este es un diagrama que describe cada iteración de la validación cruzada de 5 pliegues:

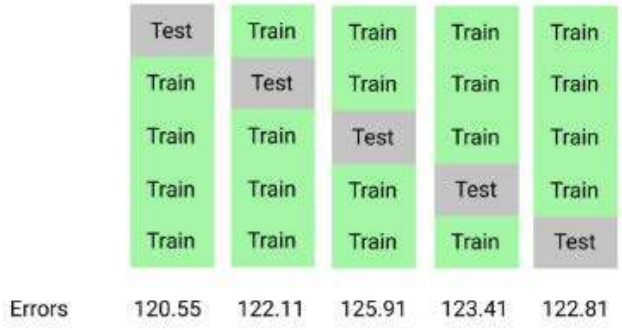


Figura 5. Diagrama que describe cada iteración de la validación cruzada de 5 veces

A medida que aumenta el número de pliegues, el número de observaciones en cada pliegue disminuye y la varianza de los errores por pliegue aumenta. Empecemos por dividir manualmente el conjunto de datos en 5 pliegues. En lugar de dividir en 5 marcos de datos, añadamos una columna que especifique a qué pliegue pertenece la fila. De este modo, podemos seleccionar fácilmente nuestro conjunto de entrenamiento y nuestro conjunto de prueba.

Instrucciones

- Agregue una nueva columna a **dc_listings** llamada **fold** que contenga el número de pliegue al que cada fila pertenece:
- El pliegue **1** debe tener filas desde el índice **0** hasta el **745**, sin incluir el **745**
- El pliegue **2** debe tener filas desde el índice 745 hasta el **1490**, sin incluir el **1490**
- El pliegue **3** debe tener filas desde el índice 1490 hasta el **2234**, sin incluir el **2234**
- El pliegue **4** debe tener filas desde el índice 2234 hasta el **2978**, sin incluir el **2978**
- El pliegue **5** debe tener filas desde el índice 2978 hasta el **3723**, sin incluir el **3723**
- Muestre los recuentos de valores únicos para la columna de **fold** para confirmar que cada pliegue tiene aproximadamente el mismo número de elementos
- Muestre el número de valores perdidos en la columna **fold** para confirmar que no perdimos ninguna fila

Soluciones

```
dc_listings.loc[dc_listings.index[0:745], "fold"] = 1
dc_listings.loc[dc_listings.index[745:1490], "fold"] = 2
dc_listings.loc[dc_listings.index[1490:2234], "fold"] =
3
dc_listings.loc[dc_listings.index[2234:2978], "fold"] =
4
dc_listings.loc[dc_listings.index[2978:3723], "fold"] =
5

print(dc_listings['fold'].value_counts())
print("\n Num of missing values: ",
dc_listings['fold'].isnull().sum())
```

Hasta ahora, hemos trabajado bajo el supuesto de que un RMSE más bajo siempre significa que un modelo es más preciso. Por desgracia, esto no es del todo cierto. Un modelo tiene dos fuentes de error, el sesgo y la varianza.

El sesgo describe el error que resulta de las malas suposiciones sobre el algoritmo de aprendizaje. Por ejemplo, suponer que sólo una característica, como el peso de un coche, está relacionada con la eficiencia del combustible de un coche le llevará a ajustar un modelo de regresión simple y univariante que dará lugar a un alto sesgo. La tasa de error será alta, ya que la eficiencia de combustible de un coche se ve afectada por muchos otros factores además de su peso.

La varianza describe el error que se produce debido a la variabilidad de los valores predichos de un modelo. Si nos dieran un conjunto de datos con 1.000 características de cada coche y utilizáramos cada una de ellas para entrenar un modelo de regresión multivariante increíblemente complicado, tendríamos un sesgo bajo pero una varianza alta. En un mundo ideal, queremos un sesgo bajo y una varianza baja, pero en la realidad, siempre hay una compensación.

La desviación estándar de los valores de RMSE puede ser un indicador de la varianza de un modelo, mientras que el RMSE medio es un indicador del sesgo de un modelo. El sesgo y la varianza son las dos fuentes de error observables en un modelo que podemos controlar indirectamente.

ARTIFICIAL INTELLIGENCE
PROFESSIONAL CERTIFICATE

I.7 Proyecto Guiado: Predicción de los Precios de Automóviles

CAIPC™ Version 062021

CertiProf®

Proyecto Guiado: Predicción de los Precios de Automóviles

En este curso, hemos explorado los fundamentos del aprendizaje automático utilizando el algoritmo de k- vecinos más cercanos. En este proyecto guiado, practicarás el flujo de trabajo de aprendizaje automático que has aprendido hasta ahora para predecir el precio de mercado de un coche utilizando sus atributos. El conjunto de datos con el que trabajaremos contiene información sobre varios coches. Para cada coche tenemos información sobre los aspectos técnicos del vehículo, como la cilindrada del motor, el peso del coche, los kilómetros por galón, la velocidad de aceleración del coche, etc. Puedes leer más sobre el conjunto de datos aquí y puedes descargarlo directamente desde aquí. Aquí tienes un avance del conjunto de datos:

https://archive.ics.uci.edu/ml/datasets/automobile

symboling	normalized_losses	make	fuel_type	aspiration	num_doors
3	?	alfa-romero	gas	std	two
3	?	alfa-romero	gas	std	two
1	?	alfa-romero	gas	std	two
2	164	audi	gas	std	four
2	164	audi	gas	std	four

Instrucciones

- Lee **imports-85.data** en un dataframe llamado **cars**. Si lees el archivo usando **pandas.read_csv()** sin especificar ningún valor de parámetro adicional, notarás que los nombres de las columnas no coinciden con los de la documentación del conjunto de datos (dataset´s documentation) ¿Por qué cree que esto es así y cómo puede solucionarlo?
- Determine qué columnas son numéricas y pueden utilizarse como características y qué columna es la columna de destino
- Muestre las primeras filas del marco de datos y asegúrese de que se parece a la vista previa del conjunto de datos

Soluciones

Puede encontrar las soluciones para este proyecto guiado aquí.

ARTIFICIAL INTELLIGENCE
PROFESSIONAL CERTIFICATE

II Cálculo para el Aprendizaje Automático

CAIPC™ Version 062021

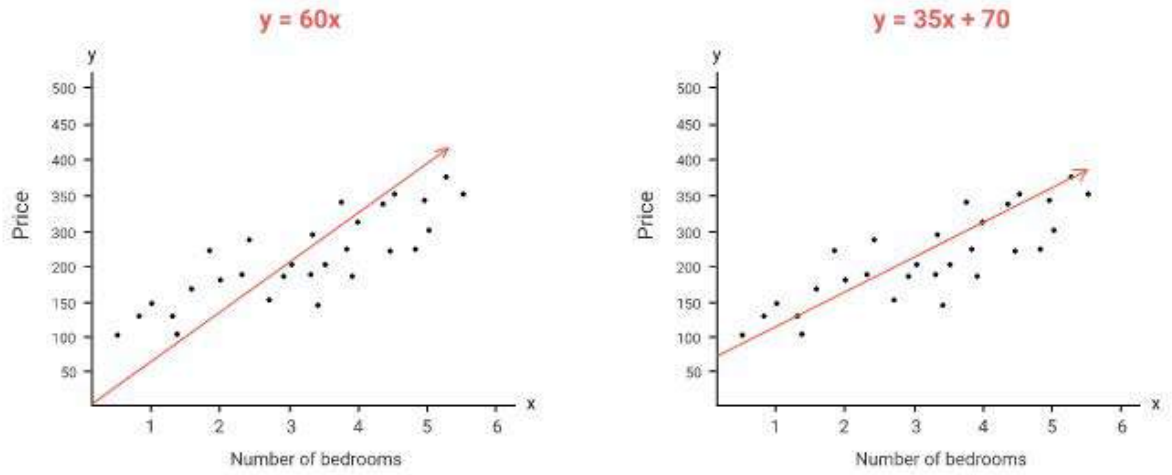
CertiProf®

Cálculo para el Aprendizaje Automático

En el curso anterior, exploramos el flujo de trabajo del aprendizaje automático utilizando el algoritmo de k-próximos más cercanos. Elegimos el algoritmo de k-próximos más cercanos porque construir la intuición de cómo funciona el algoritmo no requiere ninguna matemática. Aunque el algoritmo es fácil de entender, no podemos utilizarlo para conjuntos de datos más grandes porque el propio modelo se representa utilizando todo el conjunto de entrenamiento. Cada vez que queremos hacer una predicción sobre una nueva observación, tenemos que calcular la distancia entre cada observación de nuestro conjunto de entrenamiento y nuestra nueva observación, y luego ordenar por distancia ascendente. Se trata de una técnica muy intensiva desde el punto de vista informático.

En la mayoría de las técnicas de aprendizaje automático que veremos a continuación, el modelo se representa como una función matemática. Esta función matemática se aproxima a la función subyacente que describe cómo se relacionan las características con el atributo objetivo. Una vez que derivamos esta función matemática utilizando el conjunto de datos de entrenamiento, hacer predicciones en el conjunto de datos de prueba (o en un futuro conjunto de datos) es computacionalmente barato. El siguiente diagrama muestra dos funciones de regresión lineal diferentes que se aproximan al conjunto de datos (nótese que los valores de este conjunto de datos son aleatorios).

https://app.dataquest.io/course/calculus-for-machine-learning



Antes de que podamos sumergirnos en el uso de modelos de regresión lineal para el aprendizaje automático, tendremos que entender algunas ideas clave del cálculo. El cálculo proporciona un marco para entender cómo se comportan las funciones matemáticas. El cálculo nos ayuda a:

- Entender la inclinación en varios puntos
- Encontrar los puntos extremos de una función
- Determinar la función óptima que mejor representa un conjunto de datos

Empecemos por plantear un problema motivador, al que nos referiremos a lo largo de este curso. Supongamos que se nos da la siguiente ecuación, que describe la trayectoria de un balón después de ser pateado por un jugador de fútbol: y= −(x2)+3x−1

X es el tiempo en segundos mientras que **Y** es la posición vertical de la pelota. Naturalmente, nos gustaría saber la posición más alta que alcanzó la pelota y a qué hora ocurrió. Aunque podemos graficar la ecuación y estimar el resultado visualmente, si queremos el tiempo y la posición vertical precisos tendremos que usar el cálculo. En este curso, exploraremos los diferentes conceptos de cálculo necesarios para poder encontrar este punto preciso.

Empecemos por visualizar esta función.

Instrucciones

- Usar **numpy.linspace()** para generar un array NumPy que contenga **100** valores de **0** a **3** y asignarlo a **x**
- Transformar **x** aplicando la función: y=-(x2)+3x-1 Asignar el array resultante de valores transformados a **y**
- Utiliza **pyplot.plot()** para generar un gráfico de líneas con **x** en el eje-x e **y** en el eje-y
- Haz una lluvia de ideas sobre cómo calcular la altura máxima y encontrar el momento exacto en que se produjo

Soluciones

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 3, 100)
y = -1*(x**2) + x*3 - 1
plt.plot(x,y)
```

Comprender las Funciones Lineales y No Lineales

Antes de entrar en el análisis de la curva de la altura de una pelota, tendremos que entender primero algunas ideas clave. Exploraremos estos conceptos utilizando primero las líneas rectas simples y, a continuación, aplicaremos estos conceptos a las curvas. Una línea recta simple se define más claramente como una función lineal. Todas las **funciones lineales** se pueden escribir de la siguiente forma:

y=mx+b

En el caso de una función lineal concreta, m y b son valores constantes, mientras que x e y son variables. y=3x+1 e y=5 son ejemplos de funciones lineales.

Centrémonos por ahora en la función y=3x+1. Esta función multiplica por 3 cualquier valor de x que le pasemos y luego le suma 1.

Empecemos por adquirir una comprensión geométrica de las funciones lineales. A continuación, encontrarás una imagen que te ayudará a entender cómo se desplaza o cambia la recta al alterar los valores de **m** y/o **b.**

- ¿Cómo cambia la línea cuando mantienes **m** fijo pero varías **b**?
- ¿Cómo cambia la línea cuando mantienes **b** fijo pero varías **m**?
- ¿Qué valor controla la inclinación de la recta?
- ¿Qué ocurre con la recta cuando **m** se fija en **0**?

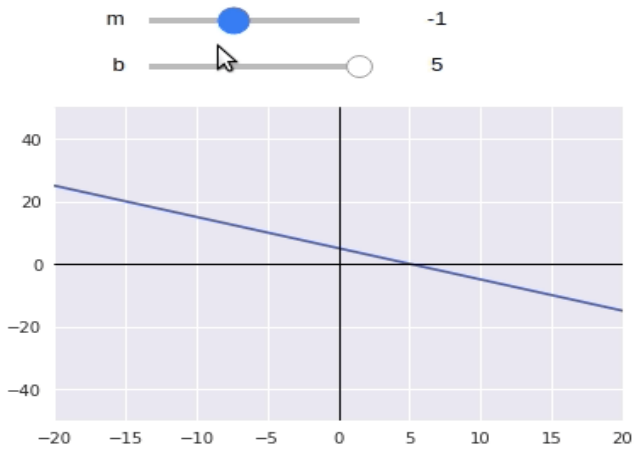


Figura 6. Ejemplo de función lineal

Hasta ahora hemos trabajado con funciones lineales, en las que podemos determinar la pendiente de la función a partir de la propia ecuación. Sin embargo, si volvemos a nuestra ecuación de la trayectoria de la pelota, te darás cuenta de que no coincide con la forma y=mx+b:

y=−(x2)+3x−1

Esto se debe a que esta función es una función **no lineal**. Las funciones no lineales no representan líneas rectas, sino curvas como la que hemos trazado en el primer paso de esta misión. Las salidas y de una función no lineal no son proporcionales a los valores de entrada **x**. Un incremento en **x** no resulta en un incremento constante en **y**.

Siempre que x se eleve a una potencia no igual a **1**, tenemos una función no lineal. He aquí algunos ejemplos más de funciones no lineales:

$$y = x^3$$

$$y = x^3 + 3x^2 + 2x - 1$$

$$y = \frac{1}{-x^2}$$

$$y = \sqrt{x}$$

En la siguiente imagen, observa cómo cambia la pendiente con diferentes valores de x1 y x2.

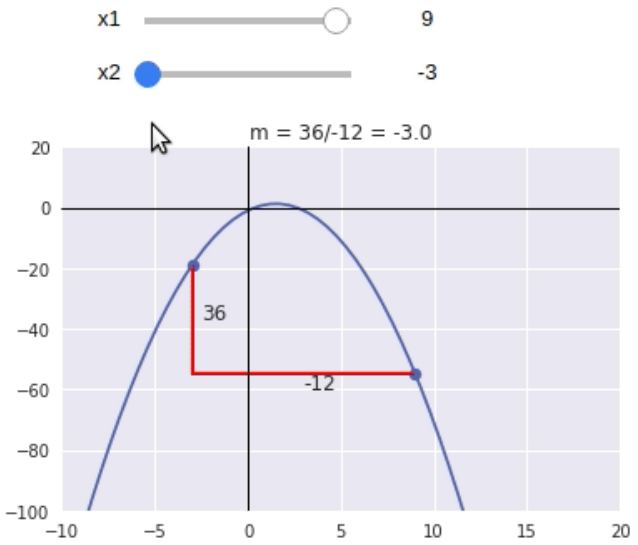


Figura 7.Ejemplo de función no lineal

Comprensión de los Límites

Al final de la última misión, fijamos un primer punto en nuestra curva, dibujamos una línea secante entre ese primer punto y un segundo punto, y observamos lo que ocurría cuando acercábamos el segundo punto al primero a lo largo de la curva. Cuanto mayor era el intervalo entre los dos puntos en el eje x, más divergía la inclinación de la línea secante de la inclinación de la curva. Cuanto más cercano es el intervalo, más se ajusta la línea secante a la pendiente del primer punto de la curva.

En esta misión, formalizaremos más la idea de la pendiente y aprenderemos a calcular la pendiente de las ecuaciones no lineales en cualquier punto. **A medida que avanzas en el resto de este curso, te recomendamos encarecidamente que sigas las matemáticas que presentamos utilizando lápiz y papel.** Empezaremos introduciendo una notación matemática que formaliza la observación que hicimos al final de la última misión. Si intentamos enunciar la observación introduciendo valores en la ecuación de la pendiente, nos encontraremos con el problema de la división por cero:

$$m = \frac{f(x_2)-f(x_1)}{x_2-x_1} = 0/0$$

Aunque la pendiente es indefinida cuando x1 y x2 son equivalentes, queremos ser capaces de afirmar y razonar sobre el valor al que se aproxima la pendiente cuando x2 se acerca a x1. Para ello, tenemos que replantear el problema como un **límite**. Un límite describe el valor al que se aproxima una función cuando la variable de entrada a la función se aproxima a un valor específico. En nuestro caso, la variable de entrada es x2 y nuestra función es m=f(x2)-f(x1)x2-x1. La siguiente notación matemática formaliza la afirmación "A medida que x2 se aproxima a 3, la pendiente entre x1 y x2 se aproxima a -3" utilizando un límite:

$$\lim_{x_2 \to 3} \frac{f(x_2)-f(x_1)}{x_2-x_1} = -3$$

limx2-->3 es otra forma de decir "A medida que x2 se acerca a 3". Como hemos fijado x1 en 3, podemos sustituir x1 por 3 en la función:

$$\lim_{x_2 \to 3} \frac{f(x_2)-f(3)}{x_2-3} = \lim_{x_2 \to 3} \frac{f(x_2)+1}{x_2-3} = -3$$

Encontrar Puntos Extremos

En la última misión, aprendimos a utilizar los límites para calcular el punto al que se aproxima una función cuando el valor de entrada se acerca a un valor específico. Hemos aplicado esta técnica para calcular la pendiente de la recta tangente en un punto concreto de nuestra función no lineal:

$$y = -(x^2) + 3x - 1$$

Si recuerdas la primera misión de este curso, nos interesa determinar el punto más alto de esta curva.

Si alguna vez has hecho senderismo en una montaña, estarás familiarizado con la forma en que el sendero se inclina hacia arriba hasta llegar a la cima. Sin embargo, una vez que se llega a la cima, todos los caminos de vuelta descienden. Entender cómo varía la pendiente a lo largo de una curva proporciona una lente útil para determinar el punto máximo de una curva.

Empezaremos construyendo una intuición visual sobre cómo se relacionan la pendiente de una función y su punto máximo. En la imagen siguiente, hemos generado dos gráficos. A medida que xx varía, el gráfico de la izquierda visualiza **cómo cambia la línea tangente de la curva**, mientras que el gráfico de la derecha **visualiza cómo cambia la pendiente de esta línea tangente.**

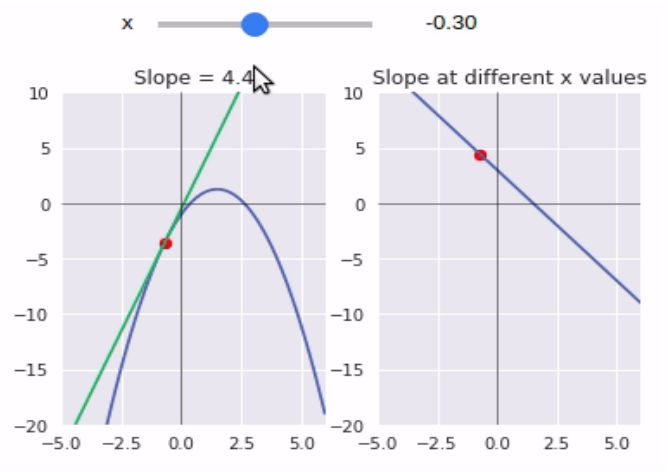


Figura 8. La línea tangente de la curva cambia

ARTIFICIAL INTELLIGENCE
PROFESSIONAL CERTIFICATE

III Álgebra Lineal para el Aprendizaje Automático

CAIPC™ Version 062021

CertiProf®

Álgebra Lineal para el Aprendizaje Automático

El álgebra lineal es un pilar del aprendizaje automático. No se puede desarrollar una profunda comprensión y aplicación del aprendizaje automático sin ella. Mediante explicaciones claras, bibliotecas estándar de Python y lecciones tutoriales paso a paso, descubrirá qué es el álgebra lineal, la importancia del álgebra lineal para el aprendizaje automático, las operaciones vectoriales y matriciales, y mucho más.

https://mml-book.github.io/book/mml-book.pdf

Sistemas Lineales

En el último curso, exploramos el marco del cálculo y lo utilizamos para:

- Comprender la pendiente de las funciones lineales
- Comprender la derivada (la pendiente como función) de las funciones no lineales
- Encontrar valores extremos en funciones no lineales

Aunque aprendimos los fundamentos de la pendiente a través de las funciones lineales, en el último curso nos centramos principalmente en las funciones no lineales.

En este curso, nos centraremos en la comprensión de las funciones lineales. Específicamente, exploraremos el marco del álgebra lineal, que proporciona una manera de representar y entender las soluciones a los sistemas de ecuaciones lineales. Un sistema de ecuaciones lineales consiste en múltiples funciones relacionadas con un conjunto común de variables. La palabra ecuación lineal se utiliza a menudo indistintamente con función lineal. Muchos procesos del mundo real pueden modelarse utilizando múltiples ecuaciones lineales relacionadas. Empezaremos explorando un ejemplo concreto de sistema lineal, otra palabra para sistema de ecuaciones lineales, antes de profundizar en el álgebra lineal.

Problema del Salario Óptimo

Supongamos que tenemos que elegir entre dos ofertas de trabajo diferentes. La primera oferta de trabajo tiene un salario semanal base de 1000 dólares y paga 30 dólares por hora. Podemos representar esta oferta como **y=1000+30x**, donde y representa los dólares ganados esa semana y xx representa las horas trabajadas esa semana. La segunda oferta de trabajo tiene un salario semanal base de 100 dólares y paga 50 dólares por hora. Podemos representar esta oferta como **y=100+50x**, donde y también representa los dólares ganados esa semana y xx también representa las horas trabajadas esa semana.

Queremos saber qué oferta de trabajo es mejor. Si sabemos exactamente la cantidad de dinero que queremos ganar cada semana (y), podemos sustituir ese valor en ambas ecuaciones y resolver para x para identificar qué trabajo nos exigirá menos horas. Si sabemos exactamente el número de horas que queremos trabajar cada semana (x), podemos sustituir ese valor en ambas ecuaciones y resolver para y para identificar qué trabajo nos hará ganar más dinero por la misma cantidad de horas trabajadas. En cambio, si queremos entender:

- ¿A partir de qué número de horas trabajadas podemos esperar ganar la misma cantidad de dinero en cualquiera de los dos trabajos?
- ¿Cuántas horas tenemos que trabajar para ganar más dinero en el primer trabajo que en el segundo?

Para responder a la primera pregunta, tenemos que encontrar el valor x en el que los dos valores y son equivalentes. Una vez que sepamos dónde se cruzan, podremos averiguar fácilmente la respuesta a la segunda pregunta.

Instrucciones

- Usar **numpy.linspace()** para generar 1000 valores espaciados uniformemente entre **0** y **50** y asignarlos a **x**
- Transforme **x** usando la ecuación y=30x+1000 y asigna el resultado a **y1**
- Transforme **x** utilizando la ecuación y=50x+100 y asigne el resultado a **y2**
- Genera 2 gráficos de líneas en la misma gráfica secundaria:
  - Uno con **x** en el eje x y y1 en el eje y. Establezca el color de la línea en **"orange"**
  - Uno con **x** en el eje x e y2 en el eje y. Establezca el color de la línea en **"blue"**
- Omitir la selección de un rango de valores para los ejes x e y, y en su lugar dejar que matplotlib seleccione automáticamente en base a los datos

Soluciones

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 50, 1000)
y1 = 30*x + 1000
y2 = 50*x + 100

plt.plot(x, y1, c='orange')
plt.plot(x, y2, c='blue')
```

Vectores

En la última misión, aprendimos a utilizar una matriz aumentada y las operaciones de fila que preservan las relaciones en un sistema para resolver un sistema de funciones lineales. En esencia, una matriz es una forma de representar una tabla de números. Todas las matrices con las que trabajamos en la última misión contenían 2 filas y 3 columnas. Aquí está la primera que configura nuestro sistema lineal:

$$\left[\begin{array}{cc|c} 30 & -1 & -1000 \\ 50 & -1 & -100 \end{array}\right]$$

Esto se conoce como una matriz de 2x3 (se pronuncia "matriz de dos por tres"). La convención en álgebra lineal es especificar primero el número de filas (2) y luego el número de columnas (3). Cada una de las filas y columnas de esta matriz se representa como una lista de números.

Una lista de números se conoce como **vector**. Una fila de una matriz se conoce como **vector fila**, mientras que una columna se conoce como **vector columna**. Estos son los vectores fila de la matriz:

$$[30 \quad -1 \quad -1000]$$

$$[50 \quad -1 \quad -100]$$

Estos son los vectores columna de la matriz:

$$\begin{bmatrix} 30 \\ 50 \end{bmatrix}$$

$$\begin{bmatrix} -1 \\ -1 \end{bmatrix}$$

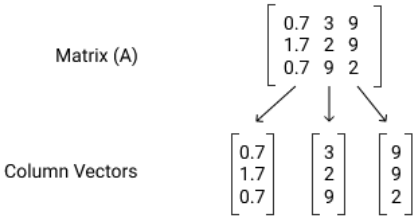
$$\begin{bmatrix} -1000 \\ -100 \end{bmatrix}$$

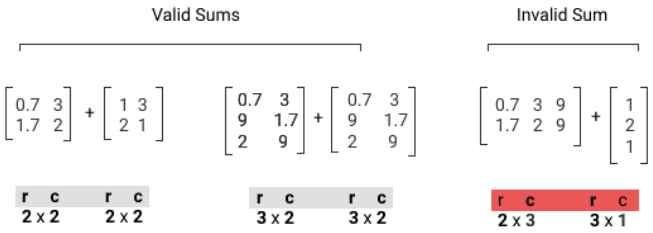
En esta misión, aprenderemos más sobre los vectores columna y sus operaciones asociadas para ayudarnos a entender ciertas propiedades de los sistemas lineales. Terminaremos esta misión justificando el enfoque que utilizamos en la última misión para resolver el sistema lineal conectando algunas ideas clave de las matrices y los vectores. Empezaremos construyendo alguna intuición geométrica de los vectores. Generalmente, la palabra vector se refiere al vector columna (lista ordenada de elementos en una sola columna) y nos referiremos al vector columna de esa manera durante el resto de este curso.

Álgebra Matricial

Al igual que los vectores, las matrices tienen su propio conjunto de operaciones algebraicas. En esta misión, aprenderemos las principales operaciones matriciales y llegaremos a utilizar algunas de ellas para resolver la ecuación matricial. Empecemos por la suma y la resta de matrices. Si recuerdas la misión anterior, una matriz está formada por uno o más vectores columna.

Por ello, las operaciones de los vectores también se trasladan a las matrices. Podemos realizar sumas y restas vectoriales entre vectores con el mismo número de filas. Podemos realizar sumas y restas de matrices entre matrices que contengan el mismo número de filas y columnas.





Al igual que con los vectores, la suma y la resta de matrices funciona distribuyendo las operaciones entre los elementos específicos y combinándolos.

$$A + B = \overset{A}{\begin{bmatrix} 0.7 & 3 & 9 \\ 1.7 & 2 & 9 \\ 0.7 & 9 & 2 \end{bmatrix}} + \overset{B}{\begin{bmatrix} 0 & 3 & 3 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}} = \begin{bmatrix} 0.7 & 6 & 12 \\ 2.7 & 4 & 10 \\ 0.7 & 10 & 4 \end{bmatrix}$$

Por último, también podemos multiplicar una matriz por un valor escalar, al igual que podemos hacerlo con un vector.

$$5A = 5 \overset{A}{\begin{bmatrix} 0.7 & 3 & 9 \\ 1.7 & 2 & 9 \\ 0.7 & 9 & 2 \end{bmatrix}} = \begin{bmatrix} 3.5 & 15 & 45 \\ 8.5 & 10 & 45 \\ 3.5 & 45 & 10 \end{bmatrix}$$

Conjuntos de Soluciones

En este curso, hemos explorado dos formas diferentes de encontrar la solución de $A\vec{x}=\vec{b}$ cuando bb no es un vector que contiene todos los ceros ($b \neq 0$). La primera forma que exploramos fue la eliminación gaussiana, que consiste en utilizar las operaciones de fila para transformar la representación aumentada de un sistema lineal a la forma escalonada y, finalmente, a la forma escalonada reducida.

La segunda forma que exploramos fue calcular la matriz inversa de A y multiplicar por la izquierda ambos lados de la ecuación para encontrar ${}^{t}x$.

Aunque podemos usar estas técnicas para resolver la mayoría de los sistemas lineales que encontraremos, necesitamos aprender qué hacer cuando:

- **El conjunto de soluciones de un sistema lineal no existe**
- **El conjunto de soluciones de un sistema lineal no es un único vector**
- **b es igual a $\vec{0}$**

En esta misión, terminaremos este curso explorando estas tres situaciones.

ARTIFICIAL INTELLIGENCE
PROFESSIONAL CERTIFICATE

IV Regresión Lineal para el Aprendizaje Automático

CAIPC™ Version 062021

CertiProf®

Regresión Lineal para el Aprendizaje Automático

La regresión lineal es uno de los algoritmos de aprendizaje automático más fáciles y populares. Es un método estadístico que se utiliza para el análisis predictivo. En esta sección aprenderás sobre los algoritmos paramétricos de aprendizaje automático y los fundamentos del modelo de regresión lineal.

Modelo de Regresión Lineal

En el primer curso de este paso, Fundamentos del Aprendizaje Automático, recorrimos el flujo de trabajo completo del aprendizaje automático utilizando el algoritmo de K-nearest neighbors . El algoritmo de K-nearest neighbors funciona encontrando ejemplos etiquetados similares del conjunto de entrenamiento para cada instancia del conjunto de prueba y los utiliza para predecir la etiqueta. El algoritmo de K-nearest neighbors se conoce como un algoritmo de aprendizaje basado en instancias porque se basa completamente en instancias anteriores para hacer predicciones. El algoritmo de K-nearest neighbors no trata de entender o capturar la relación entre las columnas de características y la columna objetivo.

Dado que todo el conjunto de datos de entrenamiento se utiliza para encontrar los vecinos más cercanos de una nueva instancia para hacer predicciones de etiquetas, este algoritmo no se adapta bien a conjuntos de datos medianos y grandes. Si tenemos un millón de instancias en nuestro conjunto de datos de entrenamiento y queremos hacer predicciones para cien mil nuevas instancias, tendríamos que ordenar el millón de instancias del conjunto de entrenamiento por distancia euclidiana para cada instancia. El siguiente diagrama proporciona una visión general de la complejidad de K-nearest neighbors.

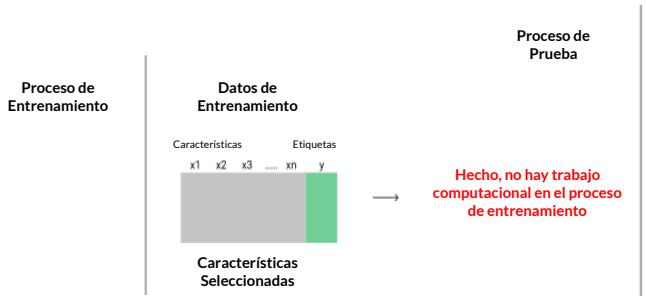


Figura 9. Proceso de entrenamiento

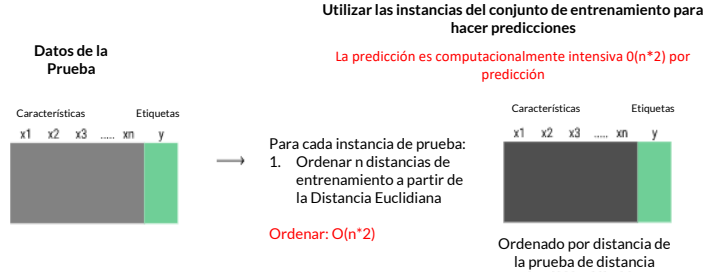


Figura 10. Proceso de prueba - Paso 1

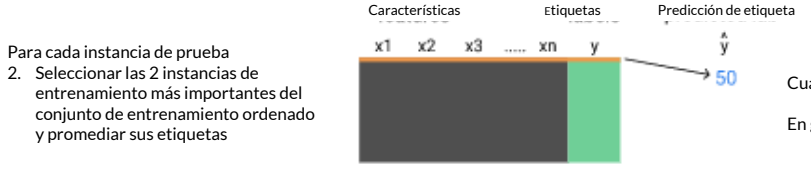


Figura 11. Proceso de prueba - Paso 2

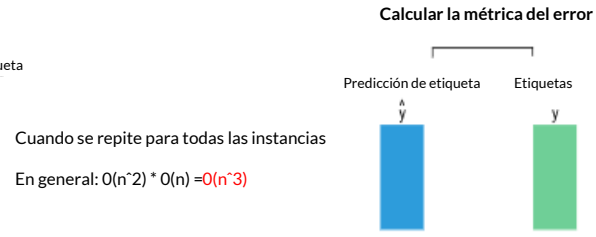


Figura 12. Proceso de prueba - Paso 3

En su lugar, debemos aprender sobre los enfoques de aprendizaje automático paramétrico, como la regresión lineal y la regresión logística. A diferencia del algoritmo k-próximo, el resultado del proceso de entrenamiento de estos algoritmos de aprendizaje automático es una función matemática que se aproxima lo mejor posible a los patrones del conjunto de entrenamiento. En el aprendizaje automático, esta función suele denominarse modelo.

En este curso, exploraremos el modelo de aprendizaje automático más utilizado: el modelo de regresión lineal. Los enfoques paramétricos de aprendizaje automático funcionan haciendo suposiciones sobre la relación entre las características y la columna objetivo. En la regresión lineal, la relación aproximada entre las columnas de características y la columna objetivo se expresa como una ecuación de regresión lineal:

$$y = a_1x_1 + a_2x_2 + a_3x_3 + \ldots + a_nx_n$$

El siguiente diagrama ofrece una visión general del proceso de aprendizaje automático para la regresión lineal. Por ahora, el objetivo no es entender todo el proceso, sino más bien comparar y contrastar con el enfoque no paramétrico **k-nearest neighbors.**

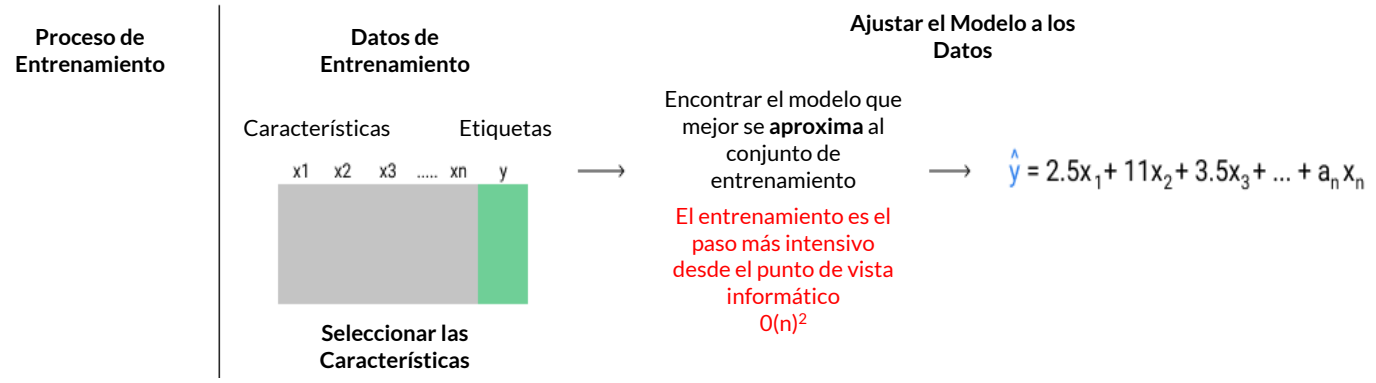


Figura 13. Proceso de aprendizaje automático para la regresión lineal-Proceso de entrenamiento

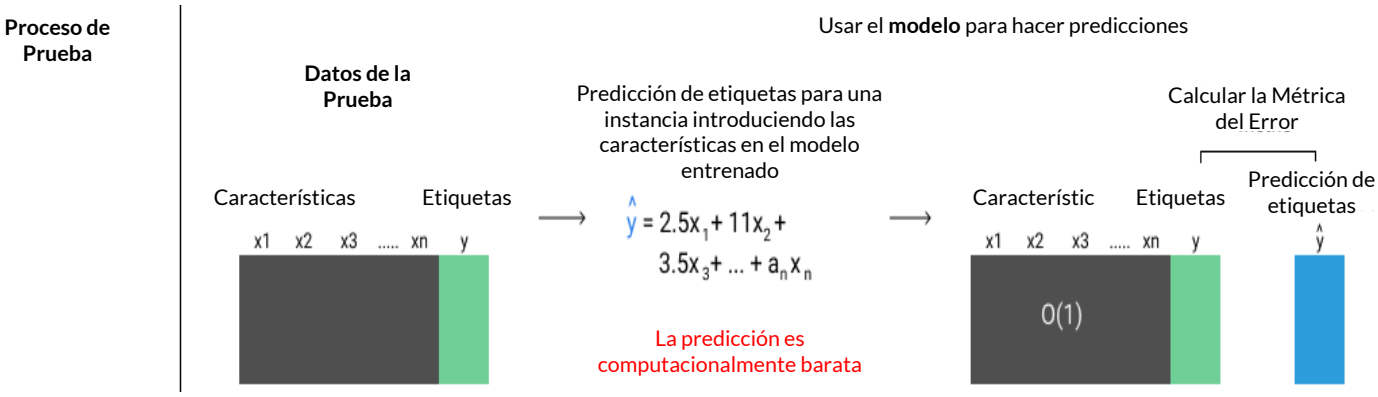


Figura 14.Proceso de aprendizaje automático para la regresión lineal-Proceso de prueba

En esta misión, vamos a proporcionar una visión general de cómo utilizamos un modelo de regresión lineal para hacer predicciones. Utilizaremos scikit-learn para el proceso de entrenamiento del modelo, de modo que podamos centrarnos en ganar intuición para el enfoque de aprendizaje basado en modelos para el aprendizaje automático.

En misiones posteriores de este curso, nos adentraremos en las matemáticas que hay detrás de cómo se ajusta un modelo al conjunto de datos, cómo seleccionar y transformar las características, y mucho más.

Selección de Características

En el flujo de trabajo del aprendizaje automático, una vez que hemos seleccionado el modelo que queremos utilizar, la selección de las características adecuadas para ese modelo es el siguiente paso importante. En esta misión, exploraremos cómo utilizar la correlación entre las características y la columna objetivo, la correlación entre las características y la varianza de las características para seleccionarlas. Seguiremos trabajando con el mismo conjunto de datos(dataset) de viviendas de la última misión.

Nos centraremos específicamente en la selección de las columnas de características que no tienen valores perdidos o que no necesitan ser transformadas para ser útiles (por ejemplo, columnas como Año de Construcción (Year Built) y Año de Remodelación (Year Remod/Add). Exploraremos cómo tratar ambos aspectos en una misión posterior de este curso.

Para empezar, veamos qué columnas entran en alguna de estas dos categorías.

Instrucciones

- Lea **AmesHousing.txt** en un dataframe llamado **data**. Asegurese de separar el delimitador **\t**
- Cree un dataframe llamado **train**, el cual contenga las primeras 1460 filas de **data**
- Cree un dataframe llamado **test**, el cual contenga el resto de las filas de **data**
- Seleccione las columnas integer y float de **train** y asignelas a las variables **numerical_train**
- Elimine las siguientes columnas de **numerical_train:**
  - **PID (Place ID no es útil para modelar)**
  - **Year Built**
  - **Year Remod/Add**
  - **Garage Yr Blt**
  - **Mo Sold**
  - **Yr Sold**
- Calcule el número de valores perdidos de cada columna en **numerical_train**. Cree un objeto Series donde el index esté hecho de columnas de nombres y los valores asociados sean los números de los valores faltantes
- Asigne este objeto Series a **null_series**. Seleccione el subconjunto **null_series** para dejar solo las columnas con sin valores faltantes, y asigne la objeto Series resultante a **full_cols_series**
- Muestre **full_cols_series** usando la función **print()**

Soluciones

```
import pandas as pd
data = pd.read_csv('AmesHousing.txt', delimiter="\t")
train = data[0:1460]
test = data[1460:]
# Default code
import pandas as pd
data = pd.read_csv('AmesHousing.txt', delimiter="\t")
train = data[0:1460]
test = data[1460:]
```

```

# Solution code
numerical_train = train.select_dtypes(include=['int', 'float'])
numerical_train = numerical_train.drop(['PID', 'Year Built', 'Year Remod/Add', 'Garage Yr Blt', 'Mo Sold', 'Yr Sold'], axis=1)
null_series = numerical_train.isnull().sum()
full_cols_series = null_series[null_series == 0]
print(full_cols_series)
```

Descenso de Gradientes

En las misiones anteriores, aprendimos cómo el modelo de regresión lineal estima la relación entre las columnas de características y la columna objetivo y cómo podemos utilizarlo para hacer predicciones. En esta misión y en la siguiente, discutiremos las dos formas más comunes de encontrar los valores óptimos de los parámetros para un modelo de regresión lineal. Cada combinación de valores de parámetros únicos forma un modelo de regresión lineal único, y el proceso de encontrar estos valores óptimos se conoce como ajuste del modelo.

En ambos enfoques del ajuste del modelo, trataremos de minimizar la siguiente función:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

Esta función es el error cuadrático medio entre las etiquetas predichas realizadas con un modelo determinado y las etiquetas verdaderas. El problema de elegir un conjunto de valores que minimicen o maximicen otra función se conoce como problema de optimización (optimization problem). Para intuir el proceso de optimización, empecemos con un modelo de regresión lineal de un solo parámetro:

$$\hat{y} = a_1 x_1$$

Nótese que esto es diferente de un modelo de regresión lineal simple, que en realidad tiene dos parámetros: a_0 y a_1.

$$\hat{y} = a_1 x_1 + a_0$$

Utilicemos la columna Gr Liv Area para el parámetro único:

$$\hat{SalePrice} = a_1 * GrLivArea$$

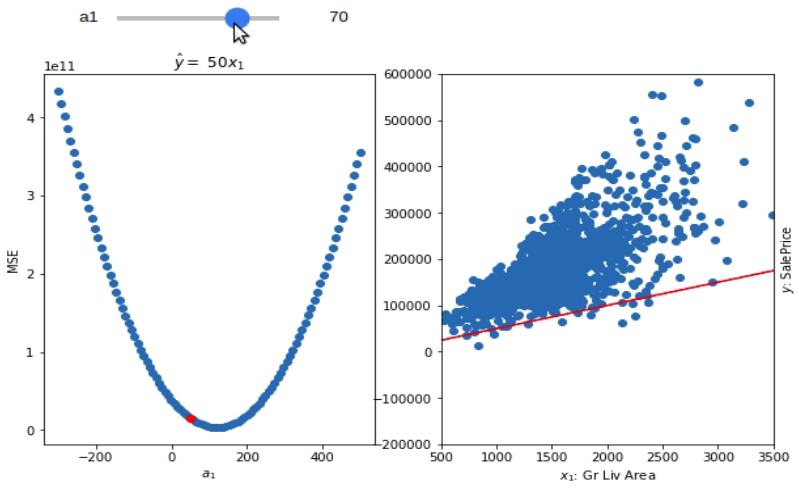


Figura 15. Modelo de regresión lineal simple para el precio de venta

Mínimos Cuadrados Ordinarios

En la última misión, exploramos una técnica iterativa para el ajuste de modelos denominada descenso de gradiente. El algoritmo de descenso de gradiente requiere múltiples iteraciones para converger en los valores óptimos de los parámetros y el número de iteraciones depende en gran medida de los valores iniciales de los parámetros y de la tasa de aprendizaje que seleccionemos.

En esta misión, exploraremos una técnica llamada estimación por **mínimos cuadrados ordinarios** o estimación OLS para abreviar. A diferencia del descenso de gradiente, la estimación OLS proporciona una fórmula clara para calcular directamente los valores óptimos de los parámetros que minimizan la función de coste. Para entender la estimación OLS, necesitamos primero enmarcar nuestro problema de regresión lineal en forma de matriz. La mayoría de las veces hemos trabajado con la siguiente forma del modelo de regresión lineal:

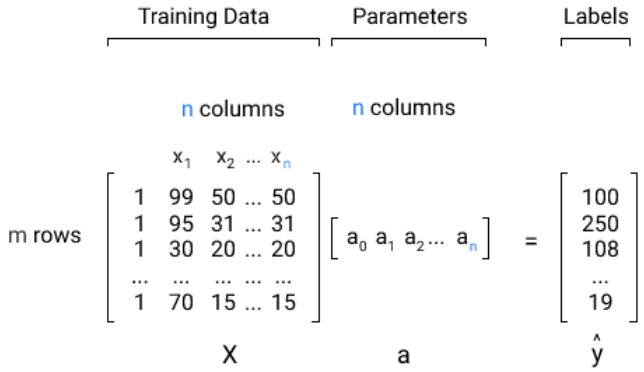
$$\hat{y} = a_0 + a_1 x_1 + a_2 x_2 + \ldots + a_n x_n$$

https://app.dataquest.io/login?target-url=%2Fm%2F238%2Fordinary-least-squares

Mientras que esta forma representa la relación entre las características (x1x1 a xnxn) y la columna objetivo (yy) bien cuando hay sólo unos pocos valores de los parámetros, no escala bien cuando tenemos cientos de parámetros. Si recuerdas el curso de Álgebra Lineal para el Aprendizaje Automático, exploramos cómo la notación matricial nos permite representar y razonar mejor sobre un sistema lineal con muchas variables. Con esto en mente, aquí está la forma matricial de nuestro modelo de regresión lineal:

$$Xa = \hat{y}$$

Donde X es una matriz que representa las columnas del conjunto de entrenamiento que utiliza nuestro modelo, aa es un vector que representa los valores de los parámetros, y $\hat{y}$ es el vector de predicciones. Aquí hay un diagrama con algunos valores de muestra para cada uno:



Ahora que hemos comprendido la representación matricial del modelo de regresión lineal, echemos un vistazo a la fórmula de estimación OLS que da como resultado el vector óptimo a:

$$a = (X^T X)^{-1} X^T y$$

Empecemos por calcular la estimación OLS para encontrar los mejores parámetros de un modelo con las siguientes características:

```
features = ['Wood Deck SF', 'Fireplaces', 'Full Bath',
'1st Flr SF', 'Garage Area',
       'Gr Liv Area', 'Overall Qual']
```

En las siguientes pantallas, nos sumergiremos en la derivación matemática de la técnica de estimación OLS. Es importante tener en cuenta que lo más probable es que nunca se implemente esta técnica en un papel de ciencia de datos y en su lugar se utilizará una implementación existente y eficiente (scikit-learn utiliza OLS bajo el capó cuando se llama fit() en una instancia LinearRegression).

Instrucciones

- Cree un dataframe, **X,** donde:
  - Su número de filas es el mismo que el del **train** (definido en el código de visualización)
  - La primera columna se llama **bias** y está poblada de **1**s en todo momento
  - Las siguientes columnas son las de las **feautures** del **train**, en el mismo orden
- Seleccionar la columna **SalePrice** (Precio de venta) del conjunto de entrenamiento y asignarla a y
- Utilice la fórmula de estimación OLS para obtener los valores óptimos de los parámetros. Guarde la estimación en la variable **ols_estimation**

Soluciones

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

data = pd.read_csv('AmesHousing.txt', delimiter="\t")
train = data[0:1460]
test = data[1460:]

features = ['Wood Deck SF', 'Fireplaces', 'Full Bath',
'1st Flr SF', 'Garage Area',
       'Gr Liv Area', 'Overall Qual']
X = train[features]
X['bias'] = 1
X = X[['bias']+features]
y = train['SalePrice']
```

```

first_term = np.linalg.inv(
        np.dot(
            np.transpose(X),
            X
        )
    )
second_term = np.dot(
        np.transpose(X),
        y
    )
ols_estimation = np.dot(first_term, second_term)
print(ols_estimation)
```

Características de Procesamiento y Transformación

Para entender cómo funciona la regresión lineal, nos hemos atascado utilizando o descartando algunas características del conjunto de datos de entrenamiento que no contenían valores perdidos y ya estaban en una representación numérica conveniente. En esta misión, exploraremos cómo transformar algunas de las características restantes para poder utilizarlas en nuestro modelo. En términos generales, el proceso de procesamiento y creación de nuevas características se conoce como ingeniería de características (feauture enginering). La ingeniería de características es un poco un arte y tener conocimientos en el dominio específico (en este caso inmobiliario) puede ayudarle a crear mejores características. En esta misión, nos centraremos en algunas estrategias independientes del dominio que funcionan para todos los problemas.

En la primera mitad de esta misión, nos centraremos sólo en las columnas que no contienen valores perdidos, pero que aún no tienen el formato adecuado para ser utilizadas en un modelo de regresión lineal. En la segunda mitad de esta misión, exploraremos algunas formas de tratar los valores perdidos.

Entre las columnas que no contienen valores perdidos, algunos de los problemas más comunes son:

- La columna no es numérica (por ejemplo, un código de zona representado mediante texto)
- La columna es numérica pero no ordinal (por ejemplo, los valores del código postal)
- La columna es numérica pero no es representativa del tipo de relación con la columna objetivo (por ejemplo, los valores del año)

Comencemos por filtrar el conjunto de entrenamiento solo a las columnas que no contienen valores perdidos.

Instrucciones

- Seleccione sólo las columnas del marco de datos de **train** (entrenamiento) que no contengan valores perdidos
- Asigne el marco de datos resultante, que contiene sólo estas columnas, a **df_no_mv**
- Utilice la pantalla de variables para familiarizarse con estas columnas

Solucciones

```
import pandas as pd

data = pd.read_csv('AmesHousing.txt', delimiter="\t")
train = data[0:1460]
test = data[1460:]

train_null_counts = train.isnull().sum()
print(train_null_counts)
df_no_mv =
train[train_null_counts[train_null_counts==0].index]
```

Proyecto Guiado: Pronóstico de los Precios de Venta de la Vivienda

En este curso, comenzamos construyendo la intuición para el aprendizaje basado en modelos, exploramos cómo funcionaba el modelo de regresión lineal, comprendimos cómo funcionaban los dos enfoques diferentes para el ajuste de modelos y algunas técnicas para limpiar, transformar y seleccionar características. En este proyecto guiado, podrás practicar lo que has aprendido en este curso explorando formas de mejorar los modelos que hemos construido. Trabajarás con datos de vivienda de la ciudad de Ames, Iowa, Estados Unidos, desde 2006 hasta 2010. Puedes leer más sobre por qué se recogieron los datos aquí. También puedes leer sobre las diferentes columnas de los datos aquí.

https://app.dataquest.io/login?target-url=%2Fm%2F240%2Fguided-project%253A-predicting-house-sale-prices

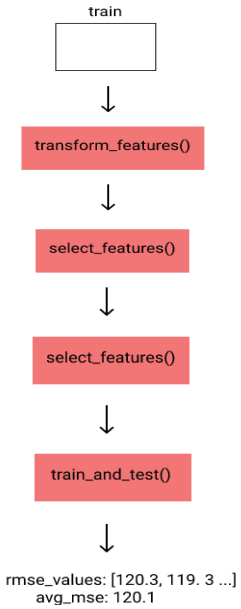


Figura 16. Cadena de funciones para la regresión lineal

Instrucciones

- Importar pandas, matplotlib y numpy en el entorno. Importa también las clases que necesites de scikit-learn
- Lee **AmesHousing.tsv** en un marco de datos de pandas
- Para las siguientes funciones, recomendamos crearlas en las primeras celdas del cuaderno. De esta manera, puedes añadir celdas al final del cuaderno para hacer experimentos y actualizar las funciones en estas celdas.
  - Cree una función llamada **transform_features()** que, por ahora, sólo devuelve el marco de datos de **train**
  - Crea una función llamada **select_features()** que, por ahora, sólo devuelve las columnas **Gr Liv Area** y **SalePrice** del marco de datos de **train**
  - Cree una función llamada **train_and_test()** que, por ahora:
    - Selecciona las primeras **1460** filas de los **datos** y las asigna al **train**
    - Seleccione las filas restantes de los **datos** y las asigne a **test**
    - Entrene un modelo utilizando todas las columnas numéricas excepto la columna **SalePrice** - Precio de venta - (la columna objetivo) del marco de datos devuelto por **select_features()**
    - Prueba el modelo en el conjunto de pruebas y devuelve el valor de RMSE

Soluciones

Puede encontrar el cuaderno de soluciones para este proyecto guiado aquí.

ARTIFICIAL INTELLIGENCE
PROFESSIONAL CERTIFICATE

V Aprendizaje Automático en Python

CAIPC™ Version 062021

CertiProf®

Regresión Logística

La regresión lineal es una técnica de aprendizaje automático supervisado que funciona bien cuando la columna objetivo que intentamos predecir, la variable dependiente, está ordenada y es continua. En cambio, si la columna objetivo contiene valores discretos, la regresión lineal no es apropiada.

En esta misión, exploraremos cómo construir un modelo de predicción para este tipo de problemas, que se conocen como **problemas de clasificación**. En la clasificación, nuestra columna objetivo tiene un conjunto limitado de valores posibles que representan diferentes categorías para una fila. Utilizamos números enteros para representar las diferentes categorías para seguir utilizando funciones matemáticas para describir cómo las variables independientes se corresponden con la variable dependiente.

He aquí algunos ejemplos de problemas de clasificación:

Problema	Ejemplos de Características	Tipo	Categorías	Categorías Numéricas
¿Debemos aceptar a este estudiante sobre la base de su solicitud de posgrado?	GPA de la Universidad, Puntuación SAT, Calidad de las Recomendaciones	Binaria	No Aceptar, Aceptar	0,1
¿Cuál es el tipo de sangre más probable de los hijos de 2 padres?	Tipo de sangre del padre 1, Tipo de sangre del padre 2	Multi-clase	A, B, AB, O	1, 2, 3, 4

Por ahora nos centraremos en la clasificación binaria, donde las únicas dos opciones de valores son:

- **0 para la condición de Falso**
- **1 para la condición Verdadera**

Antes de aprender más sobre la clasificación, vamos a entender los datos.

Introducción a la Evaluación de Clasificadores Binarios

En la lección anterior, aprendimos sobre la clasificación, la regresión logística y cómo utilizar scikit-learn para ajustar un modelo de regresión logística a un conjunto de datos sobre admisiones a escuelas de posgrado. Seguiremos trabajando con el conjunto de datos, que contiene datos sobre 644 solicitudes con las siguientes columnas:

- **GRE - Puntuación del solicitante en el Graduate Record Exam, una prueba generalizada para futuros estudiantes de posgrado**
  - La puntuación oscila entre 200 y 800
- **GPA - Promedio de notas de la universidad**
  - Continuo entre 0,0 y 4,0
- **Admit - Valor binario**
  - Valor binario, 0 o 1, donde 1 significa que el solicitante fue admitido en el programa y 0 significa que fue rechazado

Aquí tiene una vista previa del conjunto de datos:

admit	gpa	gre
0	3.177277	594.102992
0	3.412655	631.528607
0	2.728097	553.714399
0	3.093559	551.089985
0	3.141923	537.184894

Utilicemos el modelo de regresión logística de la última misión para predecir las etiquetas de clase para cada observación en el conjunto de datos y añadamos estas etiquetas al marco de datos en una columna separada.

Instrucciones

- Utilice el método LogisticRegression **predict** para devolver la etiqueta para cada observación en el conjunto de datos, las **admissions**. Asigne la lista devuelta a labels
- Añada una nueva columna al marco de datos de **admissions** denominada etiqueta predicha que contenga los valores de las etiquetas
- Utilice el método Series **value_counts** y la función **print** para mostrar la distribución de los valores en la columna **predicted_label**
- Utilice el método dataframe **head** y la función de impresión para mostrar las cinco primeras filas de **admissions**

Solutions

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression

admissions = pd.read_csv("admissions.csv")
model = LogisticRegression()
model.fit(admissions[["gpa"]], admissions["admit"])
admissions = pd.read_csv("admissions.csv")
model = LogisticRegression()
model.fit(admissions[["gpa"]], admissions["admit"])

labels = model.predict(admissions[["gpa"]])
admissions["predicted_label"] = labels
print(admissions["predicted_label"].value_counts())
print(admissions.head())
```

Clasificación Multiclase

El conjunto de datos con el que vamos a trabajar contiene información sobre varios coches. Para cada coche tenemos información sobre los aspectos técnicos del vehículo, como la cilindrada del motor, el coche, las millas por galón y la aceleración del coche. Esta información permite predecir el origen del vehículo, ya sea Norteamérica, Europa o Asia. A diferencia de nuestros anteriores conjuntos de datos de clasificación, tenemos tres categorías entre las que elegir, lo que hace que nuestra tarea sea más difícil.

He aquí un avance de los datos:

```
18.0   8   307.0      130.0      3504.      12.0   70  1
 "chevrolet chevelle malibu"
15.0   8   350.0      165.0      3693.      11.5   70  1
 "buick skylark 320"
18.0   8   318.0      150.0      3436.      11.0   70  1
 "plymouth satellite"
```

El conjunto de datos está alojado en la Universidad de California Irvine en su repositorio de aprendizaje automático (machine learning repository). El repositorio de aprendizaje automático de la UCI contiene muchos conjuntos de datos pequeños que son útiles cuando se ensucian las manos con el aprendizaje automático.

Verás que la **carpeta de datos (Data Folder)** contiene diferentes archivos. Trabajaremos con auto-mpg.data, que omite las 8 filas que contienen valores perdidos para la eficiencia del combustible (columna mpg). Hemos convertido estos datos en un archivo CSV llamado auto.csv para usted.

Estas son las columnas del conjunto de datos:

- **mpg** -- Millas por galón, Continuo
- **cylinders** -- Número de cilindros en el motor, Entero, Ordinal y Categórico
- **displacement** -- Tamaño del motor, Continuo
- **horsepower** -- Caballos de fuerza producidos, Continuo
- **weight** -- Peso del coche, Continuo
- **acceleration** -- Aceleración, Continuo
- **year** -- Año de construcción del coche, Entero y Categórico
- **origin** -- Entero y categórico. 1: Norteamérica, 2: Europa, 3: Asia

Instrucciones

- Importa la librería Pandas y lee **auto.csv** en un Dataframe llamado **cars**
- Utilice el método **Series.unique()** para asignar los elementos únicos de la columna **origin** a regiones **unique**. A continuación, utilice la función de **print** para mostrar **unique_regions**

Soluciones

```
import pandas as pd
cars = pd.read_csv("auto.csv")
print(cars.head())
unique_regions = cars["origin"].unique()
print(unique_regions)
```

Sobreajustes

Al explorar la regresión, hemos mencionado brevemente el sobreajuste y los problemas que puede causar. En esta lección, exploraremos cómo identificar el sobreajuste y qué se puede hacer para evitarlo. Para explorar el sobreajuste, utilizaremos un conjunto de datos sobre coches que contiene 7 características numéricas que podrían tener un efecto sobre la eficiencia del combustible de un coche:

- **cylinders** -- el número de cilindros en ek motor
- **displacement** -- la cilindrada del motor
- **horsepower** -- los caballos de fuerza del motor
- **weight** -- el peso del coche
- **acceleration** -- la aceleración del coche
- **model year** -- el año en que salió al mercado el modelo del coche (por ejemplo, 70 corresponde a 1970)
- **origin** -- dónde se fabricó el coche (0 si es Norteamérica, 1 si es Europa, 2 si es Asia)

- La columna **mpg** es nuestra columna objetivo y queremos predecir usando las otras características
- El conjunto de datos está alojado en la Universidad de California Irvine en su repositorio de aprendizaje automático (machine learning repository). Trabajaremos con auto-mpg.data, que omite las 8 filas que contienen valores perdidos para la eficiencia del combustible (columna **mpg**)
- El código de inicio importa Pandas, lee los datos en un marco de datos y limpia algunos valores desordenados Explora el conjunto de datos para familiarizarte con él
- Al leer el código de inicio, es posible que descubras una sintaxis diferente. Si ejecutas el código localmente en Jupyter Notebook o Jupyter Lab, notarás una advertencia SettingWithCopy. Esto no impedirá que tu código se ejecute correctamente, pero te avisa de que la operación que estás realizando está intentando establecerse en una copia de un slice de un dataframe. Para resolver esto, se considera una buena práctica incluir **.copy()** siempre que se realicen operaciones en un marco de datos

Fundamentos de Agrupación (Clustering)

Hasta ahora, hemos hablado de la regresión y la clasificación. Ambos son tipos de aprendizaje automático supervisado (supervised machine learning). En el aprendizaje supervisado, se puede entrenar un algoritmo para predecir una variable desconocida a partir de variables conocidas. Otro tipo importante de aprendizaje automático es el llamado aprendizaje no supervisado (unsupervised learning). En el aprendizaje no supervisado, no intentamos predecir nada. En su lugar, tratamos de encontrar patrones en los datos.

Utilizaremos un algoritmo llamado k-means clustering para dividir nuestros datos en clusters. k-means clustering utiliza la distancia euclidiana para formar clusters de senadores similares. En una lección posterior nos adentraremos en la teoría del clustering de k-means y construiremos el algoritmo desde cero. Por ahora, es importante entender la agrupación a un nivel alto, así que aprovecharemos la biblioteca scikit-learn para entrenar un modelo k-means.

El algoritmo k-means agrupa a los senadores que votan de forma similar en los proyectos de ley en clusters. A cada grupo se le asigna un centro y se calcula la distancia euclidiana de cada senador al centro. Los senadores se asignan a los clusters en función de la proximidad. A partir de nuestros conocimientos previos, pensamos que los senadores se agrupan según las líneas de partido.

El algoritmo k-means requiere que especifiquemos el número de agrupaciones (clusters) inicialmente, porque sospechamos que las agrupaciones se producirán según las líneas de los partidos, y la gran mayoría de los senadores son republicanos o demócratas. Elegiremos 2 para nuestro número de agrupaciones.

Utilizaremos la clase KMeans de scikit-learn para realizar la agrupación, porque no estamos prediciendo nada. No hay riesgo de sobreajuste, así que entrenaremos nuestro modelo en todo el conjunto de datos. Después del entrenamiento, podremos determinar las etiquetas de los clusters que indican el cluster de cada senador.

Podemos inicializar el modelo así:

```
kmeans_model = KMeans(n_clusters=2, random_state=1)
```

El código anterior inicializa el modelo k-means con **2** clusters y un estado aleatorio de **1** para permitir que se reproduzcan los mismos resultados cada vez que se ejecute el algoritmo.

Entonces podremos utilizar el método fit transform() para ajustar el modelo a los votos y obtener la distancia de cada senador a cada cluster. El resultado será similar al del ejemplo siguiente:

```
array([[ 3.12141628,  1.3134775 ],
       [ 2.6146248 ,  2.05339992],
       [ 0.33960656,  3.41651746],
       [ 3.42004795,  0.24198446],
       [ 1.43833966,  2.96866004],
       [ 0.33960656,  3.41651746],
       [ 3.42004795,  0.24198446],
       [ 0.33960656,  3.41651746],
       [ 3.42004795,  0.24198446],
       [ 0.31287498,  3.30758755],
       ...
```

Se trata de un array NumPy con dos columnas. La primera columna es la distancia euclidiana de cada senador al primer cluster y la segunda columna es la distancia euclidiana al segundo cluster. Los valores de las columnas indican lo "lejos" que está el senador de cada clúster. Cuanto más lejos esté el grupo, menos se alineará el historial de votos del senador con el del grupo.

Agrupación (Clustering) de K-means

En la cobertura mediática de la NBA, los periodistas deportivos suelen centrarse en unos pocos jugadores y crear historias sobre la singularidad de las estadísticas de estos jugadores. Como científicos de datos, es probable que seamos escépticos en cuanto a la singularidad de cada jugador. En esta lección, utilizaremos la ciencia de los datos para explorar esta idea observando un conjunto de datos de información de jugadores de la temporada 2013-2014. Estas son las columnas con las que trabajaremos:

- **player** - el nombre del jugador
- **pos** - la posición del jugador
- **g** - el número de partidos jugados
- **pts** - el total de puntos anotados por el jugador
- **fg.** - el porcentaje de tiros de campo
- **ft.** - el porcentaje de tiros libres

Consulta el glosario de Basketball Reference para obtener una explicación de cada columna.

Proyecto Guiado: Predicción de la Bolsa

Predicción de la Bolsa de valores. Si no has pasado por un proyecto guiado en esta interfaz, aquí tienes una rápida introducción. Puedes utilizar estas herramientas en línea:

- Google colaboratory
- Cocalc Collaborative Calculation and Data Science

Para obtener una visión más completa, te recomendamos que realices el proyecto guiado Working With Data Downloads. En este proyecto, trabajarás con datos del índice S&P500 Index. El S&P500 es un índice bursátil. Antes de entrar en lo que es un índice, tendremos que empezar con los fundamentos del mercado de valores.

Algunas empresas cotizan en bolsa, lo que significa que cualquiera puede comprar y vender sus acciones en el mercado abierto. Una acción da derecho al propietario a cierto control sobre la dirección de la empresa y a un porcentaje (o parte) de los beneficios de la misma. Cuando se compran o venden acciones, es lo que comúnmente se conoce como negociar una acción.

El precio de una acción se basa en la oferta y la demanda de una acción determinada. Por ejemplo, las acciones de Apple tienen un precio de 120 dólares por acción en diciembre de 2015 - http://www.nasdaq.com/symbol/aapl. Una acción que tiene menos demanda, como Ford Motor Company, tiene un precio más bajo -- http://finance.yahoo.com/q?s=F. . En el precio de las acciones también influyen otros factores, como el número de acciones que ha emitido una empresa.

Las acciones se negocian a diario y el precio puede subir o bajar desde el principio de un día de negociación hasta el final en función de la demanda. Las acciones que tienen más demanda, como Apple, se negocian con más frecuencia que las acciones de empresas más pequeñas.

Los índices reúnen los precios de varios valores y permiten ver la evolución del mercado en su conjunto. Por ejemplo, el Promedio Industrial Dow Jones (the Dow Jones Industrial Average) agrupa los precios de las acciones de 30 grandes empresas estadounidenses. El índice S&P500 agrupa las cotizaciones de 500 grandes empresas. Cuando un fondo índice sube o baja, se puede decir que el mercado primario o el sector que representa está haciendo lo mismo. Por ejemplo, si el precio del Promedio Industrial Dow Jones baja un día, se puede decir que las acciones estadounidenses en general bajaron (es decir, la mayoría de las acciones estadounidenses bajaron de precio).

Utilizará los datos históricos del precio del índice S&P500 para hacer predicciones sobre los precios futuros. Predecir si un índice sube o baja ayuda a pronosticar cómo se comporta el mercado de valores en su conjunto. Dado que las acciones tienden a correlacionarse con el rendimiento de la economía en su conjunto, también puede ayudar a realizar previsiones económicas.

Hay miles de operadores que ganan dinero comprando y vendiendo fondos cotizados (Exchange Traded Funds o ETF). Los **ETF** le permiten comprar y vender índices como si fueran acciones. Esto significa que usted podría "comprar" el **ETF** del índice S&P500 cuando el precio es bajo y vender cuando es alto para obtener un beneficio. La creación de un modelo predictivo podría permitir a los operadores ganar dinero en el mercado de valores.

Las columnas del conjunto de datos son:

- **Date** -- La fecha del registro
- **Open** -- El precio de apertura del día (cuando comienza la negociación)
- **High** -- El precio de negociación más alto del día
- **Low** -- El precio de negociación más bajo del día
- **Close** -- El precio de cierre del día (cuando finaliza la negociación)
- **Volumen** -- El número de acciones negociadas
- **Adj Close** -- El precio de cierre diario, ajustado retroactivamente para incluir cualquier acción corporativa

Utilizará este conjunto de datos para desarrollar un modelo predictivo. Entrenará el modelo con datos de 1950-2020 e intentará hacer predicciones de 2013-2015.

Nota: No deberías hacer operaciones con ningún modelo desarrollado en esta guía. Operar con acciones tiene riesgos y nada en esta guía constituye un consejo para operar con acciones.

En esta lección, trabajará con un archivo csv que contiene los precios de los índices. Cada fila del archivo contiene un registro diario del precio del índice S&P500 desde 1950 hasta 2015. El conjunto de datos se almacena en sphist.csv.

ARTIFICIAL INTELLIGENCE
PROFESSIONAL CERTIFICATE

VI Árbol de Decisiones

CAIPC™ Version 062021

CertiProf®

Árbol de Decisiones

- El árbol de decisión es una técnica de aprendizaje supervisado que puede utilizarse tanto para problemas de clasificación como de regresión, pero se prefiere para resolver problemas de clasificación. Se trata de un clasificador estructurado en forma de árbol, en el que los nodos internos representan las características de un conjunto de datos, las ramas representan las reglas de decisión y cada nodo hoja representa el resultado
- En un árbol de decisión, hay dos nodos, que son el nodo de decisión y el nodo hoja. Los nodos de decisión se utilizan para tomar cualquier decisión y tienen múltiples ramas, mientras que los nodos hoja son el resultado de esas decisiones y no contienen más ramas
- Las decisiones o la prueba se realizan sobre la base de las características del conjunto de datos dado
- Se trata de una representación gráfica para obtener todas las posibles soluciones a un problema/ decisión en función de unas condiciones dadas
- Se denomina árbol de decisión porque, de forma similar a un árbol, comienza con el nodo raíz, que se expande en otras ramas y construye una estructura similar a un árbol
- Para construir un árbol, se utiliza el algoritmo CART, que significa algoritmo de árbol de clasificación y regresión
- Un árbol de decisión simplemente formula una pregunta y, en función de la respuesta (Sí/No), divide el árbol en subárboles
- El siguiente diagrama explica la estructura general de un árbol de decisión:

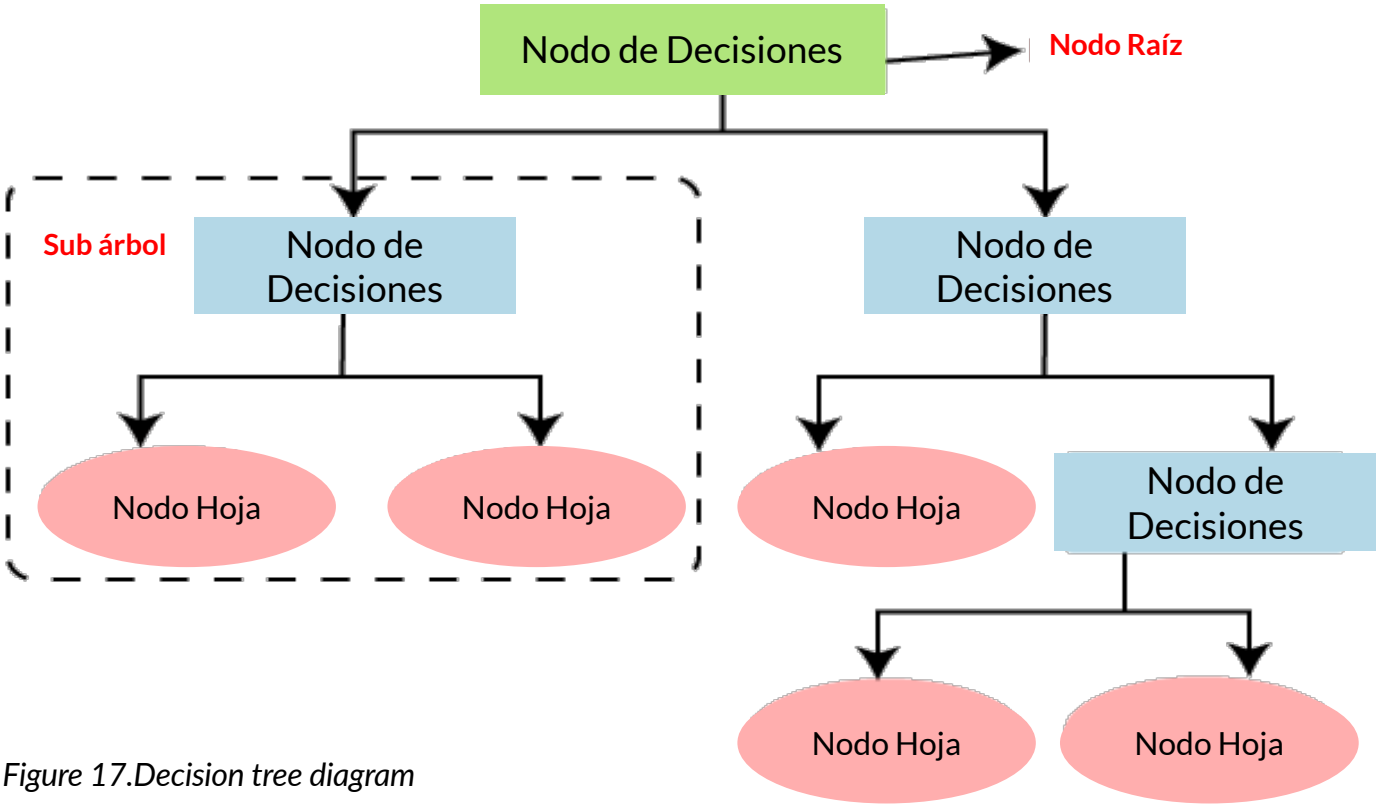


Figure 17.Decision tree diagram

Nota: Un árbol de decisión puede contener datos categóricos (SÍ/NO) así como datos numéricos.

¿Por qué utilizar Árboles de Decisiones?

Hay varios algoritmos en el aprendizaje automático, por lo que la elección del mejor algoritmo para el conjunto de datos y el problema dados es el punto principal que hay que recordar al crear un modelo de aprendizaje automático. A continuación se exponen las dos razones para utilizar el árbol de decisión:

- Los árboles de decisión suelen imitar la capacidad de pensamiento humano al tomar una decisión, por lo que son fáciles de entender
- La lógica detrás del árbol de decisión puede entenderse fácilmente porque muestra una estructura similar a la de un árbol

Terminología de los Árboles de Decisiones

- **Nodo raíz:** El nodo raíz es el punto de partida del árbol de decisión. Representa todo el conjunto de datos, que a su vez se divide en dos o más conjuntos homogéneos
- **Nodo hoja:** Los nodos hoja son el nodo de salida final, y el árbol no se puede segregar más después de obtener un nodo hoja
- **División:** La división es el proceso de dividir el nodo de decisión/nodo raíz en subnodos según las condiciones dadas
- **Rama/Subárbol:** Un árbol formado por la división del árbol
- **Poda:** La poda es el proceso de eliminar las ramas no deseadas del árbol
- **Nodo padre/hijo:** El nodo raíz del árbol se llama nodo padre, y los demás nodos se llaman nodos hijos

Cómo Funciona el Algoritmo del Árbol de Decisiones

En un árbol de decisión, para predecir la clase del conjunto de datos dado, el algoritmo parte del nodo raíz del árbol. Este algoritmo compara los valores del atributo raíz con el atributo del registro (conjunto de datos real) y, basándose en la comparación, sigue la rama y salta al siguiente nodo.

En el siguiente nodo, el algoritmo vuelve a comparar el valor del atributo con los demás subnodos y sigue avanzando. Continúa el proceso hasta llegar al nodo hoja del árbol. El proceso completo puede entenderse mejor utilizando el siguiente algoritmo:

- **Paso 1:** Comienza el árbol con el nodo raíz, llamado S, que contiene el conjunto de datos completo.
- **Paso 2:** Encontrar el mejor atributo en el conjunto de datos utilizando la medida de selección de atributos (ASM)
- **Paso 3:** Dividir S en subconjuntos que contengan los posibles valores de los mejores atributos
- **Paso 4:** Generar el nodo del árbol de decisión que contiene el mejor atributo
- **Paso 5:** Hacer recursivamente nuevos árboles de decisión utilizando los subconjuntos del conjunto de datos creados en el paso 3. Continúe este proceso hasta que se llegue a una etapa en la que no se puedan clasificar más los nodos y se llame al nodo final como nodo hoja

Ejemplo: Supongamos que hay un candidato que tiene una oferta de trabajo y quiere decidir si debe aceptar la oferta o no. Entonces, para resolver este problema, el árbol de decisión comienza con el nodo raíz (atributo Salario por ASM). El nodo raíz se divide en el siguiente nodo de decisión (distancia a la oficina) y un nodo hoja basado en las etiquetas correspondientes. El siguiente nodo de decisión se divide a su vez en un nodo de decisión (instalación de taxi) y un nodo de hoja. Por último, el nodo de decisión se divide en dos nodos de hoja (ofertas aceptadas y ofertas rechazadas). Considere el siguiente diagrama:

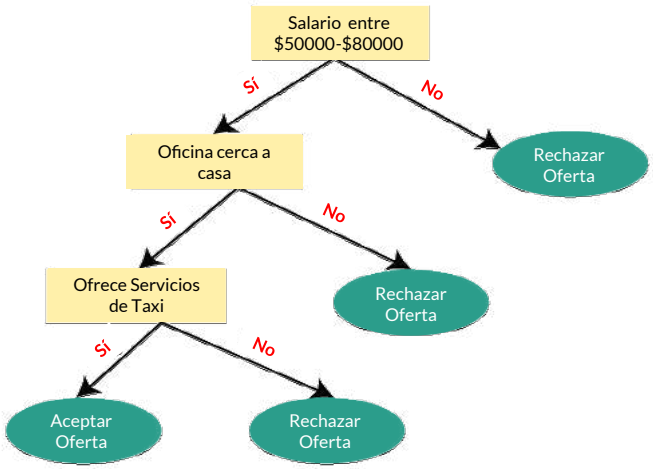


Figura 18. Medidas de Selección de Atributos

Cuando se implementa un árbol de decisión, el principal problema es cómo seleccionar el mejor atributo para el nodo raíz y los subnodos. Por lo tanto, para resolver estos problemas existe una técnica que se denomina medida de selección de atributos o ASM. Con esta medida, podemos seleccionar fácilmente el mejor atributo para los nodos del árbol. Hay dos técnicas populares para ASM, que son:

- Ganancia de información
- Índice de Gini

1. Ganancia de información:

- La ganancia de información es la medida de los cambios en la entropía tras la segmentación de un conjunto de datos basado en un atributo
- Calcula cuánta información nos proporciona una característica sobre una clase
- Según el valor de la ganancia de información, dividimos el nodo y construimos el árbol de decisión
- Un algoritmo de árbol de decisión siempre intenta maximizar el valor de la ganancia de información, y el nodo/atributo que tiene la mayor ganancia de información se divide primero. Se puede calcular mediante la siguiente fórmula: **Ganancia de información= Entropía(S) - [(Media ponderada) * Entropía(cada característica) or Information Gain= Entropy(S) - [(Weighted Avg) * Entropy(each feature)**

Entropía: La entropía es una métrica que mide la impureza de un atributo determinado. Especifica la aleatoriedad de los datos. La entropía puede calcularse como:

```
Information Gain= Entropy(S)- [(Weighted Avg) *Entropy(each feature)
```

Donde,

- **S= Número total de muestras**
- **P(yes)= probabilidades de si**
- **P(no)= probabilidades de no**

2. Índice de Gini:

- El índice de Gini es una medida de impureza o pureza que se utiliza al crear un árbol de decisión en el algoritmo CART (árbol de clasificación y regresión)
- Un atributo con un índice de Gini bajo debería ser preferido en comparación con un índice de Gini alto
- Sólo crea divisiones binarias, y el algoritmo CART utiliza el índice de Gini para crear divisiones binarias
- El índice de Gini se puede calcular mediante la siguiente fórmula:

$$\text{Gini Index} = 1 - \sum_j P_j^2$$

Poda: Obtención de un Árbol de Decisión Óptimo

La poda (pruning) es un proceso que consiste en eliminar los nodos innecesarios de un árbol para obtener el árbol de decisión óptimo.

Un árbol demasiado grande aumenta el riesgo de sobreajuste, y un árbol pequeño puede no capturar todas las características importantes del conjunto de datos. Por lo tanto, una técnica que disminuye el tamaño del árbol de aprendizaje sin reducir la precisión se conoce como poda. Existen principalmente dos tipos de tecnología de poda de árboles:

- **Poda de complejidad de costes**
- **Poda de error reducido**

Ventajas del Árbol de Decisiones

- Es fácil de entender, ya que sigue el mismo proceso que sigue un ser humano al tomar cualquier decisión en la vida real
- Puede ser muy útil para resolver problemas relacionados con la toma de decisiones
- Ayuda a pensar en todos los posibles resultados de un problema
- Requiere menos limpieza de datos en comparación con otros algoritmos

Desventajas del Árbol de Decisiones

- El árbol de decisión contiene muchas capas, lo que lo hace complejo
- Puede tener un problema de sobreajuste, que puede resolverse con el **algoritmo Random Forest**
- Para más etiquetas de clase, la complejidad computacional del árbol de decisión puede aumentar

Implementación en Python del Árbol de Decisiones

Ahora implementaremos el árbol de decisión utilizando Python. Para ello, utilizaremos el conjunto de datos "user_data.csv", que hemos utilizado en modelos de clasificación anteriores. Utilizando el mismo conjunto de datos, podemos comparar el clasificador de árbol de decisión con otros modelos de clasificación como KNN SVM, LogisticRegression, etc.

Los pasos también seguirán siendo los mismos, que se indican a continuación:

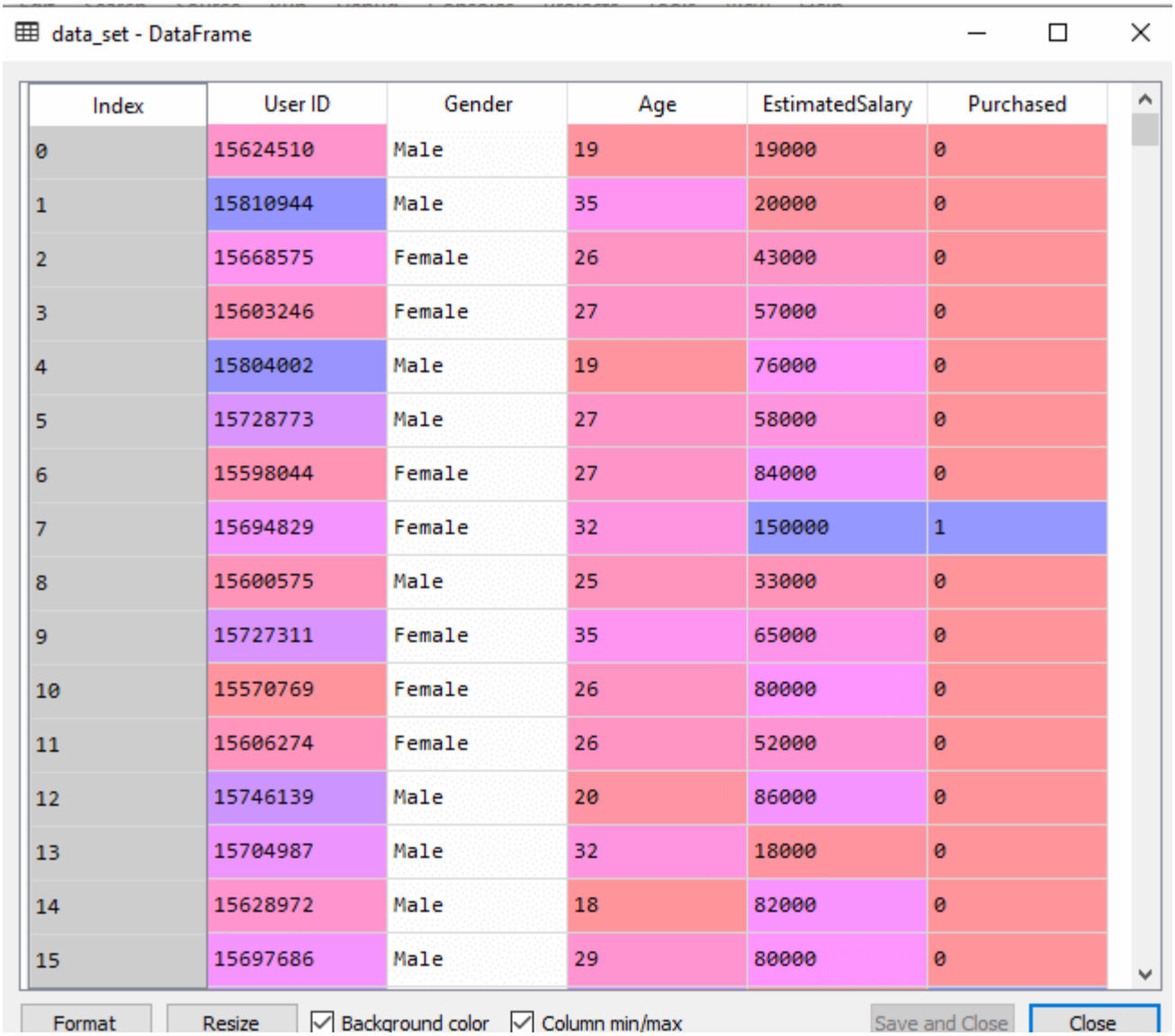
- Paso de pre-procesamiento de datos
- Ajuste de un algoritmo de árbol de decisión al conjunto de entrenamiento
- Predicción del resultado de la prueba
- Precisión de la prueba del resultado (creación de la matriz de confusión)
- Visualización del resultado del conjunto de pruebas

1. Paso de Pre-procesamiento de Datos:

A continuación se muestra el código para el paso de pre-procesamiento.

```
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('user_data.csv')

#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values

# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)

#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

En el código anterior, hemos pre-procesado los datos. Donde hemos cargado el conjunto de datos, que se da como:



Index	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0
11	15606274	Female	26	52000	0
12	15746139	Male	20	86000	0
13	15704987	Male	32	18000	0
14	15628972	Male	18	82000	0
15	15697686	Male	29	80000	0

2. Ajuste de un algoritmo de árbol de decisión al conjunto de entrenamiento
Ahora vamos a ajustar el modelo al conjunto de entrenamiento. Para ello, importaremos la clase **DecisionTreeClassifier** de la biblioteca **sklearn.tree**. A continuación se muestra el código para ello:

```
#Fitting Decision Tree classifier to the training set
From sklearn.tree import DecisionTreeClassifier
classifier= DecisionTreeClassifier(criterion='entropy', random_state=0)
classifier.fit(x_train, y_train)
```

En el código anterior, hemos creado un objeto clasificador, en el que hemos pasado dos parámetros principales:

- **"criterion='entropy'**: El criterio se utiliza para medir la calidad de la división, que se calcula por la ganancia de información dada por la entropía
- **random_state=0"**: Para generar los estados aleatorios

A continuación se muestra la salida de esto:

```
Out[8]:
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
max_features=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, presort=False,
                       random_state=0, splitter='best')
```

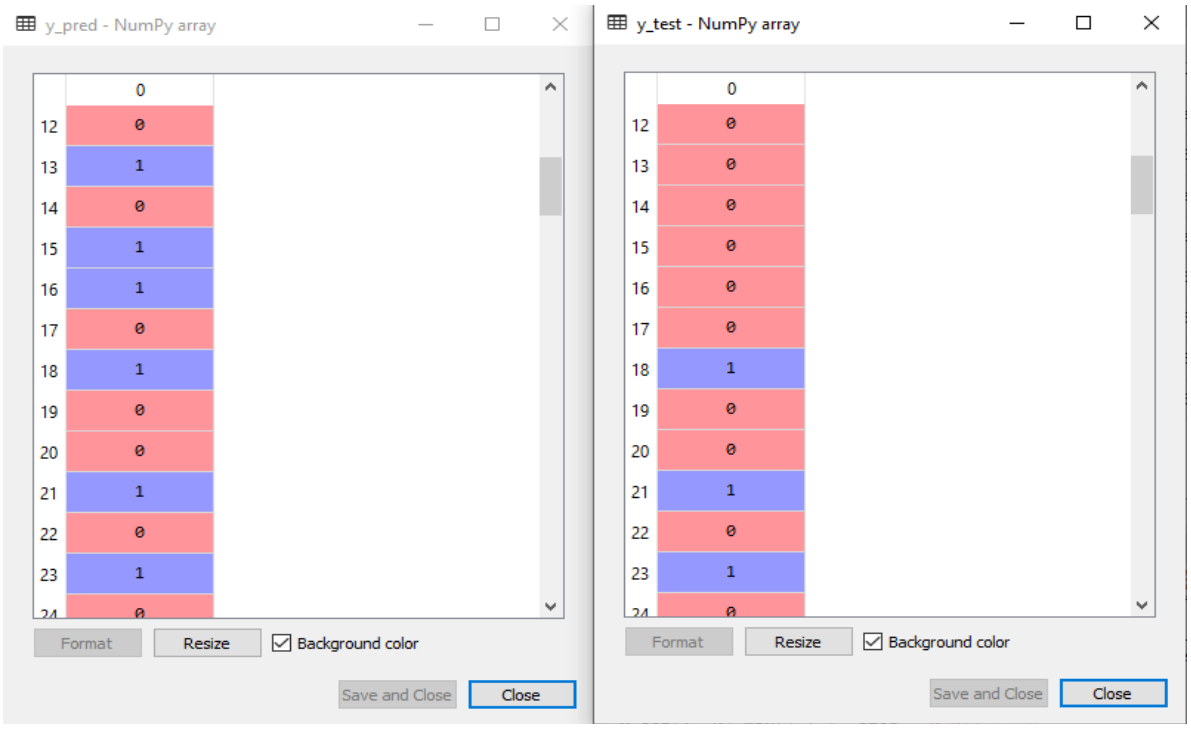
3. Predicción del resultado de la prueba

Ahora vamos a predecir el resultado del conjunto de pruebas. Crearemos un nuevo vector de predicción y_pred. A continuación se muestra el código para ello:

```
#Predicting the test set result
y_pred= classifier.predict(x_test)
```

Salida:

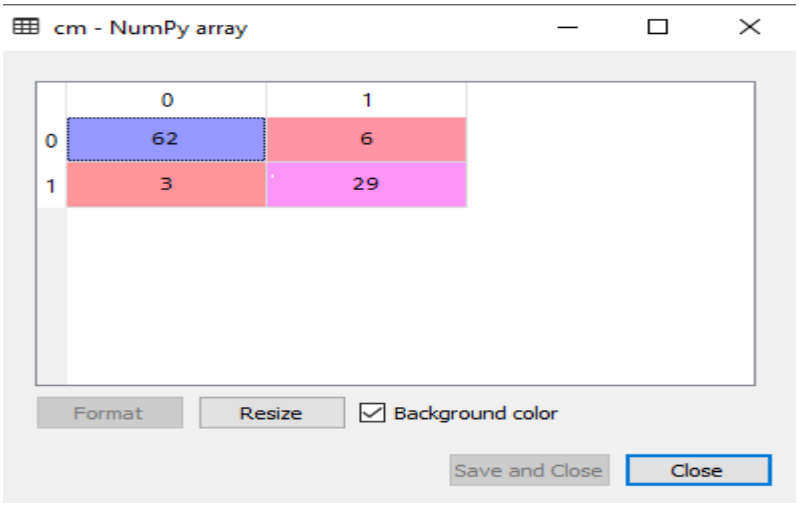
En la siguiente imagen de salida, se da la salida predicha y la salida real de la prueba. Podemos ver claramente que hay algunos valores en el vector de predicción, que son diferentes de los valores del vector real. Se trata de errores de predicción.



4. Comprobar la precisión del resultado (Creación de la matriz de confusión)
En el resultado anterior, hemos visto que había algunas predicciones incorrectas, por lo que si queremos saber el número de predicciones correctas e incorrectas, tenemos que utilizar la matriz de confusión. A continuación se muestra el código para ello:

```
#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
```

Output:



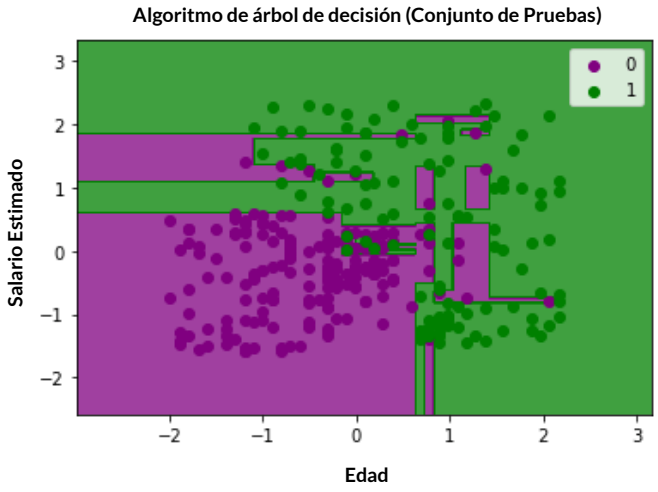
En la imagen de salida anterior, podemos ver la matriz de confusión, que tiene **6+3= 9 predicciones incorrectas y 62+29=91 predicciones correctas. Por lo tanto, podemos decir que, en comparación con otros modelos de clasificación, el clasificador de árbol de decisión hizo una buena predicción.**

5. Visualización del resultado del conjunto de entrenamiento:

Aquí visualizaremos el resultado del conjunto de entrenamiento. Para visualizar el resultado del conjunto de entrenamiento, trazaremos un gráfico para el clasificador del árbol de decisión. El clasificador predecirá Si o No para los usuarios que han comprado o no han comprado el coche SUV como hicimos en la Regresión Logística. A continuación se muestra el código para ello:

```
#Visulaizing the trianing set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1,
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
fori, j in enumerate(nm.unique(y_set)):
mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Decision Tree Algorithm (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

Output:



El resultado anterior es completamente diferente del resto de modelos de clasificación. Tiene líneas verticales y horizontales que están dividiendo el conjunto de datos según la edad y la variable de salario estimado.

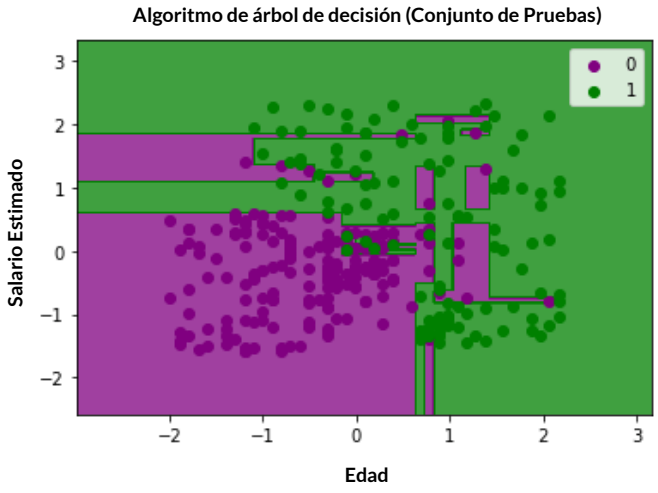
Como podemos ver, el árbol está tratando de capturar cada conjunto de datos, que es el caso de sobreajuste.

6. Visualización del resultado del conjunto de pruebas:

La visualización del resultado del conjunto de prueba será similar a la visualización del conjunto de entrenamiento, excepto que el conjunto de entrenamiento será reemplazado por el conjunto de prueba.

```
#Visulaizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1,
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
fori, j in enumerate(nm.unique(y_set)):
mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Decision Tree Algorithm(Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

Output:



Como podemos ver en la imagen anterior, hay algunos puntos de datos verdes dentro de la región púrpura y viceversa. Por lo tanto, estas son las predicciones incorrectas que hemos discutido en la matriz de confusión.

Proyecto Guiado: Predicción del Alquiler de Bicicletas

Muchas ciudades de Estados Unidos cuentan con estaciones de uso compartido de bicicletas en las que se pueden alquilar por horas o por días. Washington D.C. es una de estas ciudades. El Distrito recoge datos detallados sobre el número de bicicletas que la gente alquila por horas y días.

Hadi Fanaee-T, de la Universidad de Oporto, recopiló estos datos en un archivo CSV, con el que trabajarás en este proyecto. El archivo contiene 17380 filas, en las que cada fila representa el número de bicicletas alquiladas durante una sola hora de un solo día. Puedes descargar los datos del sitio web de la Universidad de California, Irvine. Si necesitas ayuda en algún momento, puedes consultar el cuaderno de soluciones en nuestro repositorio de GitHub.

Este es el aspecto de las cinco primeras filas:

instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
1	2011-01-01	1	0	1	0	0	6	0	1	0.24	0.2879	0.81	0	3	13	16
2	2011-01-01	1	0	1	1	0	6	0	1	0.22	0.2727	0.8	0	8	32	40
3	2011-01-01	1	0	1	2	0	6	0	1	0.22	0.2727	0.8	0	5	27	32
4	2011-01-01	1	0	1	3	0	6	0	1	0.24	0.2879	0.75	0	3	10	13
5	2011-01-01	1	0	1	4	0	6	0	1	0.24	0.2879	0.75	0	0	1	1

Estas son las descripciones de las columnas correspondientes:

- **instant** - Un número de identificación secuencial único para cada fila
- **dteday** - La fecha de los alquileres
- **season** - La temporada en la que se produjeron los alquileres
- **yr** - El año en que se produjeron los alquileres
- **mnth** - El mes en que se produjeron los alquileres
- **hr** - La hora en que se produjo el alquiler
- **holiday**- Si el día era o no festivo
- **weekday**- El día de la semana (como número, de 0 a 7)
- **workingday** - Si el día era o no laborable
- **weathersit** - El tiempo (como variable categórica)
- **temp** - La temperatura, en una escala de 0 a 1
- **atemp** - La temperatura ajustada
- **hum** - La humedad, en una escala de 0 a 1
- **windspeed** - La velocidad del viento, en una escala de 0 a 1
- **casual** - El número de ciclistas ocasionales (personas que no se han inscrito previamente en el programa de bicicletas compartidas)
- **registered** - El número de ciclistas registrados (personas que ya se han inscrito)
- **cnt** - El número total de bicicletas alquiladas (casuales + registradas)

En este proyecto, intentarás predecir el número total de bicicletas alquiladas en una hora determinada. Predecirá la columna cnt utilizando todas las demás columnas, excepto las casuales y las registradas. Para lograrlo, crearás algunos modelos de aprendizaje automático diferentes y evaluarás su rendimiento.

Instrucciones

- Utilice la biblioteca pandas para leer **bike_rental_hour.csv** en el marco de datos **bike_rentals**
- Imprime las primeras filas de **bike_rentals** y echa un vistazo a los datos
- Haga un histograma de la columna cnt de **bike_rentals**, y eche un vistazo a la distribución de los alquileres totales
- Utilice el método corr en el marco de datos **bike_rentals** para explorar cómo se correlaciona cada columna con **cnt**

Referencias y Bibliografía

Trabajos

[1] M. Nasiri, B. Minaei, and Z. Sharifi, "Adjusting data sparsity problem using linear algebra and machine learning algorithm," Appl. Soft Comput., vol. 61, pp. 1153–1159, Dec. 2017, doi: 10.1016/j.asoc.2017.05.042.

[2] G. Marzano and A. Novembre, "Machines that Dream: A New Challenge in Behavioral-Basic Robotics," Procedia Comput. Sci., vol. 104, pp. 146–151, Jan. 2017, doi: 10.1016/j.procs.2017.01.089.

[3] Y. Ao, H. Li, L. Zhu, S. Ali, and Z. Yang, "The linear random forest algorithm and its advantages in machine learning assisted logging regression modeling," J. Pet. Sci. Eng., vol. 174, pp. 776–789, Mar. 2019, doi: 10.1016/j.petrol.2018.11.067.

[4] D. A. Otchere, T. O. Arbi Ganat, R. Gholami, and S. Ridha, "Application of supervised machine learning paradigms in the prediction of petroleum reservoir properties: Comparative analysis of ANN and SVM models," J. Pet. Sci. Eng., vol. 200, p. 108182, May 2021, doi: 10.1016/j.petrol.2020.108182.

[5] Y. Iwamoto et al., "Development and Validation of Machine Learning-Based Prediction for Dependence in the Activities of Daily Living after Stroke Inpatient Rehabilitation: A Decision-Tree Analysis," J. Stroke Cerebrovasc. Dis., vol. 29, no. 12, p. 105332, Dec. 2020, doi: 10.1016/j.jstrokecerebrovasdis.2020.105332.

[6] K. Maheswari, A. Priya, A. Balamurugan, and S. Ramkumar, "Analyzing student performance factors using KNN algorithm," Mater. Today Proc., Feb. 2021, doi: 10.1016/j.matpr.2020.12.1024.

[7] L. Liang et al., "Status evaluation method for arrays in large-scale photovoltaic power stations based on extreme learning machine and k-means," Energy Rep., vol. 7, pp. 2484–2492, Nov. 2021, doi: 10.1016/j.egyr.2021.04.039.

[8] W. A. van Eeden et al., "Predicting the 9-year course of mood and anxiety disorders with automated machine learning: A comparison between auto-sklearn, naïve Bayes classifier, and traditional logistic regression," Psychiatry Res., vol. 299, p. 113823, May 2021, doi: 10.1016/j.psychres.2021.113823.

[9] C. M. Yesilkanat, "Spatio-temporal estimation of the daily cases of COVID-19 in worldwide using random forest machine learning algorithm," Chaos Solitons Fractals, vol. 140, p. 110210, Nov. 2020, doi: 10.1016/j.chaos.2020.110210.

Libros

[10] A. Geron, Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems, $ {number}nd édition. Sebastopol, CA: O'Reilly Media, Inc, USA, 2019.

[11] Fondements de l'apprentissage automatique. Cambridge, MA: The MIT Press, 2012.

Plataformas de Autoaprendizaje

https://www.dataquest.io
https://www.analyticsvidhya.com
https://cloudacademy.com
https://data-flair.training
https://learn.datacamp.com

ARTIFICIAL INTELLIGENCE
PROFESSIONAL CERTIFICATE

¡Síguenos, ponte en contacto!

www.certiprof.com

CertiProf®