



Ligdi González

2.^a edición

Ligdi González

2020

Machine Learning con Python

Aprendizaje Supervisado

INDICE

CAPÍTULO 1.....	6
INTRODUCCIÓN	6
MACHINE LEARNING CON PYTHON.....	6
CAPÍTULO 2	8
PREPARAR EL ENTORNO DE PYTHON.....	8
PYTHON	8
LIBRERÍAS PARA LA CIENCIA DE DATOS	9
LIBRERÍAS PARA LA VISUALIZACIÓN DE DATOS	10
LIBRERÍAS PARA MACHINE LEARNING.....	10
INSTALAR EL ENTORNO DE PYTHON.....	11
TRABAJANDO CON EL IDE SPYDER	13
TRABAJANDO CON EL IDE JUPYTER NOTEBOOK.....	14
CAPÍTULO 3	16
CURSOS INTENSIVOS DE PYTHON Y SCIPY.....	16
CURSO INTENSIVO DE PYTHON	16
CURSO INTENSIVO DE NUMPY	17
CURSO INTENSIVO DE PANDAS.....	21
CURSO INTENSIVO DE MATPLOTLIB.....	23
CAPÍTULO 4	25
CARGAR LOS DATOS	25
CONSIDERACIONES DE LOS ARCHIVOS CSV	25
CONJUNTO DE DATOS: DIABETES DE LOS INDIOS PIMA	26
CARGAR ARCHIVOS CSV UTILIZANDO PYTHON.....	26
CARGAR ARCHIVOS UTILIZANDO PYTHON.....	27
CAPÍTULO 5	28
ENTENDIENDO LOS DATOS	28
VERIFICAR LOS DATOS	28
DIMENSIONANDO LOS DATOS.....	29
TIPOS DE DATOS.....	30
DESCRIPCIÓN ESTADÍSTICA	30
DISTRIBUCIÓN DE CLASES (PROBLEMAS DE CLASIFICACIÓN).....	31
CORRELACIÓN ENTRE CARACTERÍSTICAS.....	32
CONSIDERACIONES ADICIONALES.....	32
CAPÍTULO 6	33
VISUALIZANDO LOS DATOS	33
GRÁFICAR CON UNA SOLA VARIABLES	33
GRÁFICAR CON VARIAS VARIABLES	35
CAPÍTULO 7	37
PROCESAMIENTO DE LOS DATOS	37
SEPARACIÓN DE LOS DATOS.....	37
ESTANDARIZACIÓN DE LOS DATOS.....	38
NORMALIZACIÓN DE LOS DATOS.....	40
ELIMINACIÓN DE COLUMNAS	41
CAPÍTULO 8	43
SELECCIONANDO CARACTERÍSTICAS.....	43
IMPORTANCIA DE LAS CARACTERÍSTICAS.....	43

MÉTODOS DE FILTRO	44
MÉTODOS DE ENVOLTURA	46
MÉTODOS INTEGRADOS	48
CAPÍTULO 9	49
ALGORITMOS DE CLASIFICACIÓN	49
REGRESIÓN LOGÍSTICA	50
K - VECINOS MÁS CERCANOS	51
MÁQUINAS DE VECTORES DE SOPORTE	52
BAYESIANO INGENUO (NAIVE BAYES)	53
ÁRBOLES DE DECISIÓN CLASIFICACIÓN	54
BOSQUES ALEATORIOS CLASIFICACIÓN	55
CAPÍTULO 10	57
MÉTRICAS DE RENDIMIENTO ALGORITMOS DE CLASIFICACIÓN	57
MATRIZ DE CONFUSIÓN	57
REPORTE DE CLASIFICACIÓN	58
ÁREA BAJO LA CURVA	59
CAPÍTULO 11	61
ALGORITMOS DE REGRESIÓN	61
REGRESIÓN LINEAL	61
REGRESIÓN POLINOMEAL	62
VECTORES DE SOPORTE REGRESIÓN	63
ÁRBOLES DE DECISIÓN REGRESIÓN	65
BOSQUES ALEATORIOS REGRESIÓN	66
CAPÍTULO 12	68
MÉTRICAS DE RENDIMIENTO ALGORITMOS DE REGRESIÓN	68
ERROR CUADRÁTICO MEDIO (RMSE)	68
ERROR ABSOLUTO MEDIO (MAE)	69
CAPÍTULO 13	71
PROYECTO DE MACHINE LEARNING - CLASIFICACIÓN	71
DEFINICIÓN DEL PROBLEMA	71
IMPORTAR LAS LIBRERÍAS	72
CARGAR EL CONJUNTO DE DATOS	72
ENTENDIENDO LOS DATOS	73
VISUALIZANDO LOS DATOS	75
SEPARACIÓN DE LOS DATOS	78
SELECCIONANDO CARACTERÍSTICAS	79
PROCESAMIENTO DE LOS DATOS	80
SEPARACIÓN DE LOS DATOS	80
APLICACIÓN DE LOS ALGORITMOS DE CLASIFICACIÓN	81
CAPÍTULO 14	89
PROYECTO DE MACHINE LEARNING - REGRESIÓN	89
DEFINICIÓN DEL PROBLEMA	89
IMPORTAR LAS LIBRERÍAS	90
CARGAR EL CONJUNTO DE DATOS	90
ENTENDIENDO LOS DATOS	91
VISUALIZANDO LOS DATOS	92
SEPARACIÓN DE LOS DATOS	94
SEPARACIÓN DE LOS DATOS	95
APLICACIÓN DE LOS ALGORITMOS DE REGRESIÓN	96
CAPÍTULO 15	102

CONTINUAR APRENDIENDO MACHINE LEARNING	102
<i>CONSTRUIR PLANTILLAS PARA PROYECTOS</i>	102
<i>CONJUNTOS DE DATOS PARA PRÁCTICA</i>	103
CAPÍTULO 16.....	105
OBTENER MÁS INFORMACIÓN.....	105
<i>CONSEJO GENERAL</i>	105
<i>AYUDA CON PYTHON</i>	106
<i>AYUDA CON SCIPY Y NUMPY</i>	106
<i>AYUDA CON MATPLOTLIB</i>	106
<i>AYUDA CON PANDAS</i>	107
<i>AYUDA CON SCIKIT-LEARN</i>	107

Capítulo 1

INTRODUCCIÓN

Este libro es una guía para aprender Machine Learning aplicado con Python. Se descubrirá, paso a paso, el proceso que se puede usar para comenzar en Machine Learning con el ecosistema de Python.

MACHINE LEARNING CON PYTHON

Este libro se enfoca en proyectos de Machine Learning con el lenguaje de programación de Python. Está escrito para explicar cada uno de los pasos para desarrollar un proyecto de Machine Learning con algoritmos de clasificación y regresión. Se encuentra dividido de la siguiente forma:

- Aprender cómo las subtareas de un proyecto de Machine Learning se asignan a Python y la mejor forma de trabajar a través de cada tarea.
- Finalmente, se une todo el conocimiento explicado para desarrollar un problema de clasificación y otro de regresión.

En los primeros capítulos se aprenderá a cómo completar las subtareas específicas de un proyecto de Machine Learning utilizando Python. Una vez que se aprenda a cómo completar una tarea utilizando la plataforma y obtener un resultado de manera confiable se podrá utilizar una y otra vez en proyecto tras proyecto.

Un proyecto de Machine Learning, se puede dividir en 4 tareas:

- **Definir el problema:** investigar y caracterizar el problema para comprender mejor los objetivos del proyecto.
- **Analizar los datos:** usar estadísticas descriptivas y visualizarlas para comprender mejor los datos que se tiene disponible.
- **Preparar los datos:** utilizar las transformaciones de datos para exponer mejor la estructura del problema de predicción a los algoritmos de Machine Learning.
- **Evaluuar los algoritmos:** diseñar pruebas para evaluar una serie de algoritmos estándar en los datos y seleccionar los mejores para investigar más a fondo.

En la última parte del libro, unirá todo lo aprendido anteriormente, para el desarrollo de un proyecto, uno con algoritmos de clasificación y otro con algoritmos de regresión.

Cada una de las lecciones están diseñadas para leerse de principio a fin en orden, y mostrar exactamente cómo completar cada tarea en un proyecto de Machine Learning. Por supuesto, se puede dedicar a capítulos específicos, posteriormente, para refrescar los conocimientos. Los capítulos están estructurados para demostrar las librerías y funciones y mostrar técnicas específicas para una tarea de Machine Learning.

Cada capítulo esta diseñado para completarse en menos de 30 minutos (dependiendo de su nivel de habilidad y entusiasmo). Es posible trabajar con todo el libro en un fin de semana. También funciona, si desea, sumergirse en capítulos específicos y utilizar el libro como referencia.

Capítulo 2

PREPARAR EL ENTORNO DE PYTHON

Python está creciendo y puede convertirse en la plataforma dominante para Machine Learning, la razón principal para adoptar este lenguaje de programación es que es de propósito general que puede ser usado tanto para investigación y desarrollo, como en producción.

En este capítulo aprenderás todo lo relacionado al entorno de Python para Machine Learning:

1. Python y su uso para Machine Learning.
2. Las librerías básicas para Machine Learning como lo son SciPy, NumPy y Pandas.
3. La librería para la visualización de datos como lo es matplotlib.
4. La librería para implementar algoritmos de Machine Learning, scikit-learn.
5. Configurar el entorno de Python utilizando Anaconda.
6. Configurar el IDE Spyder.
7. Configurar el IDE Jupyter Notebook.

PYTHON

Python lidera en lenguajes de desarrollo de Machine Learning debido a su simplicidad y facilidad de aprendizaje. Python es utilizado por más y más científicos de datos y

desarrolladores para la construcción y análisis de modelos. Además, es un éxito entre los principiantes que son nuevos en Machine Learning.

A diferencia de otros lenguajes de programación para Machine Learning, como R o MATLAB, el procesamiento de datos y las expresiones matemáticas científicas no están integradas en el lenguaje en sí, pero las librerías como SciPy, NumPy y Pandas, ofrecen una funcionalidad equivalente en una sintaxis posiblemente más accesible.

Las librerías especializadas de Machine Learning como scikit-learn, Theano, TensorFlow, le brindan la capacidad de capacitar a una variedad de modelos de Machine Learning, que potencialmente utilizan infraestructura de computación distribuida.

LIBRERÍAS PARA LA CIENCIA DE DATOS

Estas son las librerías básicas que transforman Python de un lenguaje de programación de propósito general en una poderosa y robusta herramienta para el análisis y la visualización de datos, estas son las bases sobre las que se basan las herramientas más especializadas.

SciPy

SciPy es una librería de software para ingeniería y ciencia, SciPy incluye funciones para álgebra lineal, optimización, integración y estadísticas. Proporciona rutinas numéricas eficientes como integración numérica, optimización y muchas otras a través de sus submódulos específicos. La funcionalidad principal de esta librería se basa en NumPy y sus matrices y el agregar una colección de algoritmos y comandos de alto nivel para manipular y visualizar datos.

NumPy

NumPy se refiere a Numerical Python y es una librería fundamental para la informática científica en Python ya que proporciona la vectorización de operaciones matemáticas en el tipo de matrices, mejorando el rendimiento y, en consecuencia, acelera la ejecución.

Esta orientada en administrar y tratar los datos como matrices, su propósito es proporcionar la capacidad de hacer operaciones complejas de matriz que son requeridas por redes neuronales y estadísticas complejas de manera fácil.

En resumen, NumPy presenta objetos para matrices y matrices multidimensionales, así como rutinas que permiten a los desarrolladores realizar funciones matemáticas y estadísticas avanzadas en esas matrices con el menor número de código posible.

NumPy es una librería de administración de datos que normalmente está emparejado con TensorFlow, SciPy, matplotlib y muchas otras librerías de Python orientadas hacia Machine Learning y la ciencia de datos.

Pandas

Pandas es una librería de Python diseñada para trabajar con datos “etiquetados” y “relacionales” de manera simple e intuitivos, está diseñado para una manipulación, agregación y visualización de datos rápida y fácil. Pandas agrega estructura de datos y herramientas que están diseñadas para el análisis de datos prácticos en finanzas, estadísticas e ingeniería. Pandas funciona bien con datos incompletos, desordenados y no etiquetados, es decir, el tipo de datos que es probable que se encuentre en el mundo real, y proporciona herramientas para configurar, fusionar, remodelar y dividir conjuntos de datos. Con esta librería se puede agregar y eliminar columnas fácilmente desde el DataFrame, convertir estructuras de datos en objetos y manejar datos faltantes (NaN).

LIBRERÍAS PARA LA VISUALIZACIÓN DE DATOS

El mejor y más sofisticado análisis no tiene sentido si no puedes comunicarlo a otras personas. Estas librerías permiten crear fácilmente gráficos, tablas y mapas visualmente más atractivos y sofisticados, sin importar qué tipo de análisis estés tratando de hacer.

Matplotlib

Es una librería estándar de Python para crear diagramas y gráficos en 2D, es de muy bajo nivel, lo que significa que requiere más comandos para generar gráficos y figuras agradables que con algunas librerías más avanzadas, sin embargo, la otra cara de eso es la flexibilidad, con suficientes comandos, puedes hacer casi cualquier tipo de gráficos que deseas con matplotlib.

LIBRERÍAS PARA MACHINE LEARNING

Machine Learning se encuentra en la intersección de la Inteligencia Artificial y el análisis estadístico. Al entrenar computadoras con conjuntos de datos del mundo real, se puede crear algoritmos que hacen predicciones más precisas y sofisticadas, ya sea que se este hablando de obtener mejores direcciones para conducir o construir computadores que puedan identificar hitos simplemente observando imágenes.

Scikit-learn

La librería de scikit-learn es definitivamente una de las más populares de Machine Learning, tiene una gran cantidad de características para la minería de datos y el

análisis de datos, por lo que es una excelente opción tanto para investigadores como para desarrolladores. Está construido sobre las populares librerías NumPy, SciPy y matplotlib, por lo que tendrás una sensación familiar al utilizarla.

Scikit-learn expone una interfaz concisa y consistente para los algoritmos comunes de Machine Learning, por lo que es sencillo llevar a los sistemas de producción. La librería combina código de calidad y buena documentación, facilidad de uso y alto rendimiento, y es un estándar de la industria de facto para Machine Learning con Python.



Para mayor información se puede revisar la siguiente información:

<http://bit.ly/2Qmt2Zu>

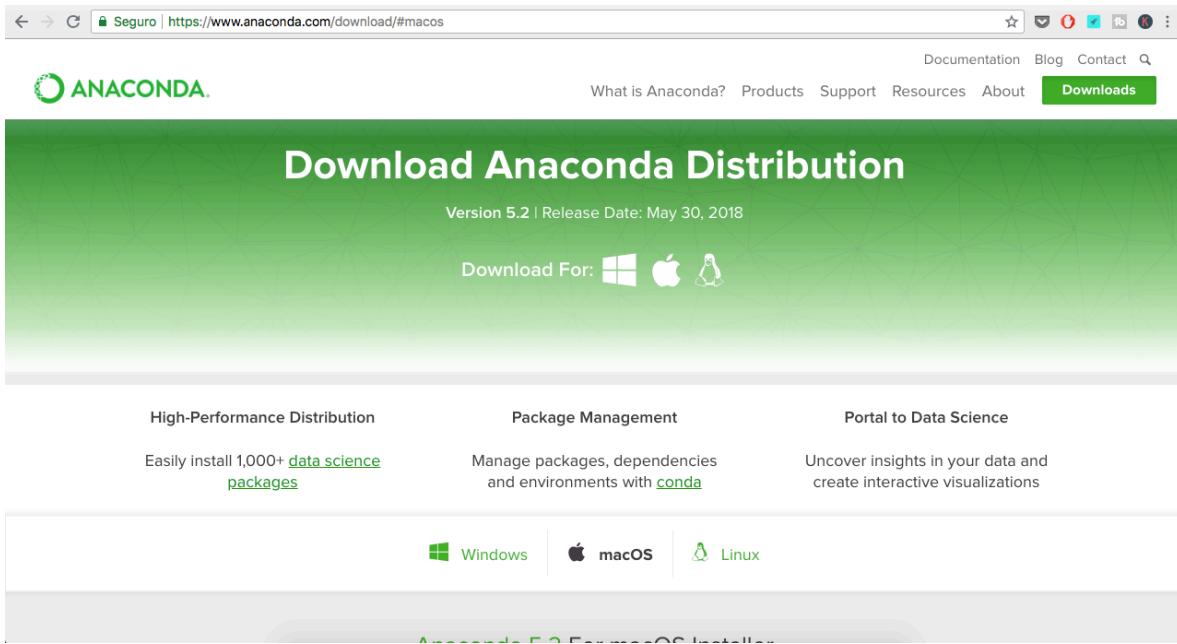
INSTALAR EL ENTORNO DE PYTHON

Existen múltiples formas de instalar el entorno de Python, IDEs y librerías para ser utilizados en Machine Learning, pero la manera más fácil, sobretodo si apenas estas aprendiendo y no conoces cómo trabajar muy bien en la terminal de comandos, es utilizando el entorno de Anaconda.

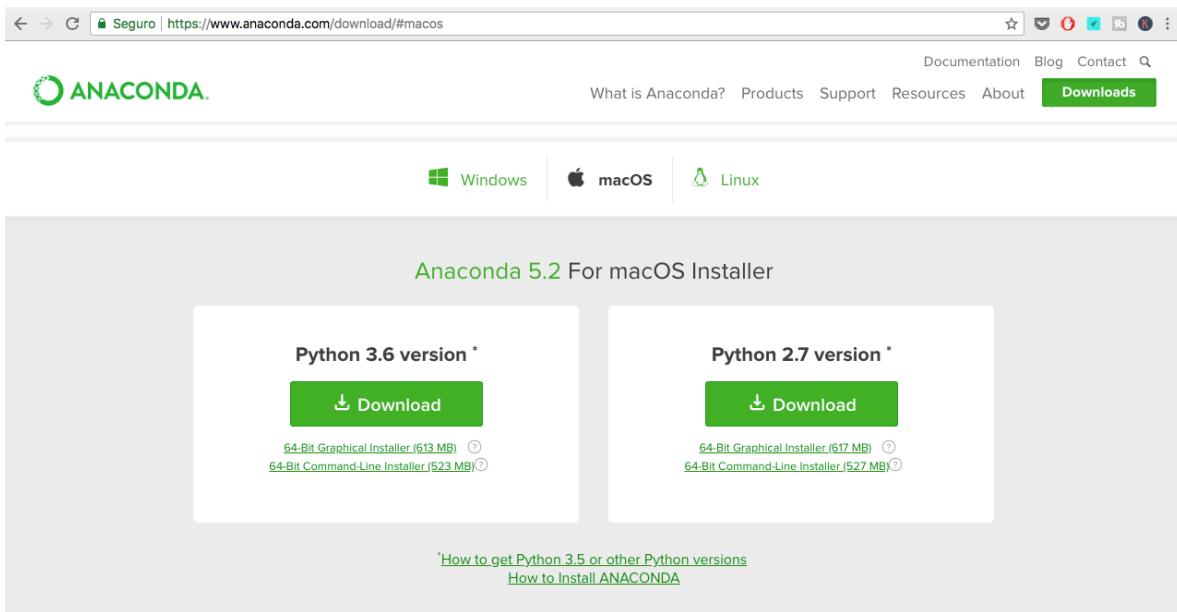
Solamente tienes que dirigir a la página oficial, [anaconda.com](https://www.anaconda.com) y presionar el botón de “Downloads”.

A screenshot of the Anaconda Data Science Certification landing page. The top navigation bar includes links for Documentation, Blog, Contact, and Downloads. The main heading is "Anaconda Data Science Certification" with a subtext "Objectively Demonstrate Your Data Science Experience". A "Learn More" button is visible. Below the heading is a large image of a diverse group of people in a classroom setting, smiling and looking at their work. The bottom section features the text "The Most Popular Python Data Science Platform" and three small icons representing different data science tools: Jupyter Notebook, Python, and Pandas.

Allí seleccionar el sistema operativo que utiliza el computador.



Luego seleccionar la versión de Python a utilizar, se recomienda seleccionar la versión 3,6 ya que la mayoría de las librerías de Machine Learning están desarrolladas para esta versión, mientras que la otra esta quedando cada día más obsoleta para su uso.

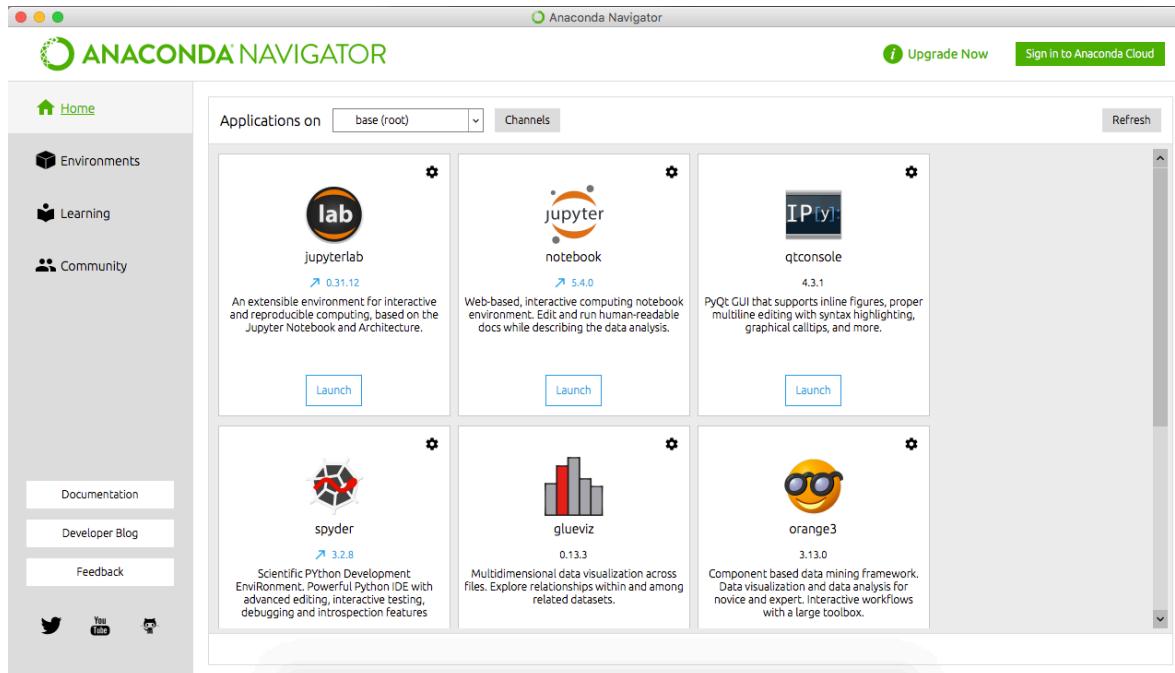


Una vez que se haya descargado, proceder a instalarlo en el computador, se debe tener un poco de paciencia porque en ocasiones todo este proceso lleva un tiempo.

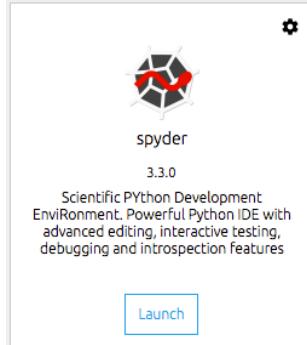
Al finalizar todo esto ya estará instalado en el computador, Python, gran parte de las librerías a implementar en Machine Learning e inclusive varios IDEs que se puede utilizar para los desarrollos de proyectos.

TRABAJANDO CON EL IDE SPYDER

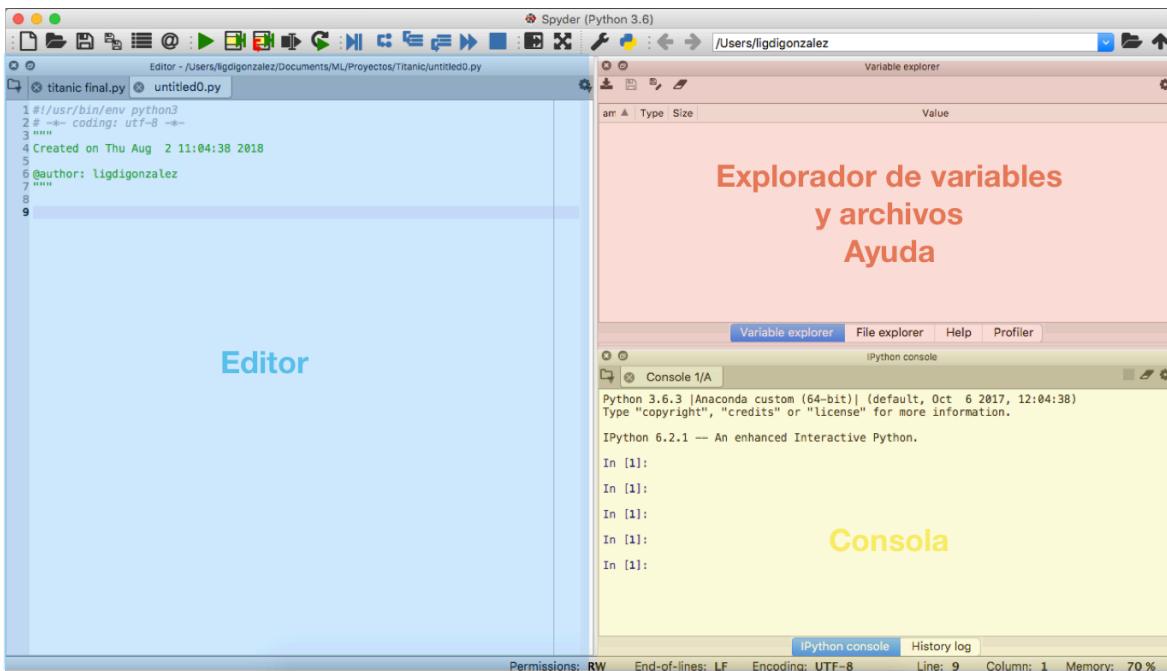
Spyder se puede encontrar en el navegador de Anaconda “Anaconda-Navigator”.



Para abrir Spyder solamente se tiene que presionar “Launch” sobre el recuadro de la aplicación.



Una vez que inicie Spyder deberas ver una ventana de editor abierto en el lado izquierdo y una ventana de consola de Python en el lado inferior derecho. Por su parte, el panel ubicado en la parte superior derecha se usa un explorador de variables, un explorador de archivos y un navegador de ayuda. Al igual que la mayoría de los IDE, se puede cambiar qué paneles son visibles y su diseño dentro de la ventana.



Se puede comenzar a trabajar con Spyder inmediatamente en la ventana de la consola. De manera predeterminada, Spyder proporciona una consola IPython que se puede usar para interactuar directamente con el motor de Python. Funciona, esencialmente, de la misma manera que funciona en la línea de comando, la gran diferencia es que Spyder puede inspeccionar los contenidos del motor de Python y puede hacer otras cosas como mostrar variables y sus contenidos dentro del explorador de variables.

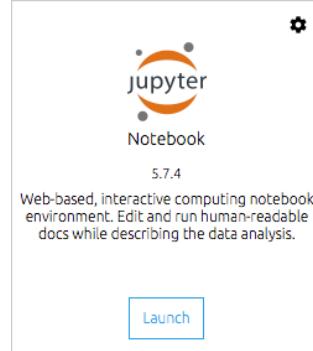


Para mayor información se puede revisar la siguiente información:

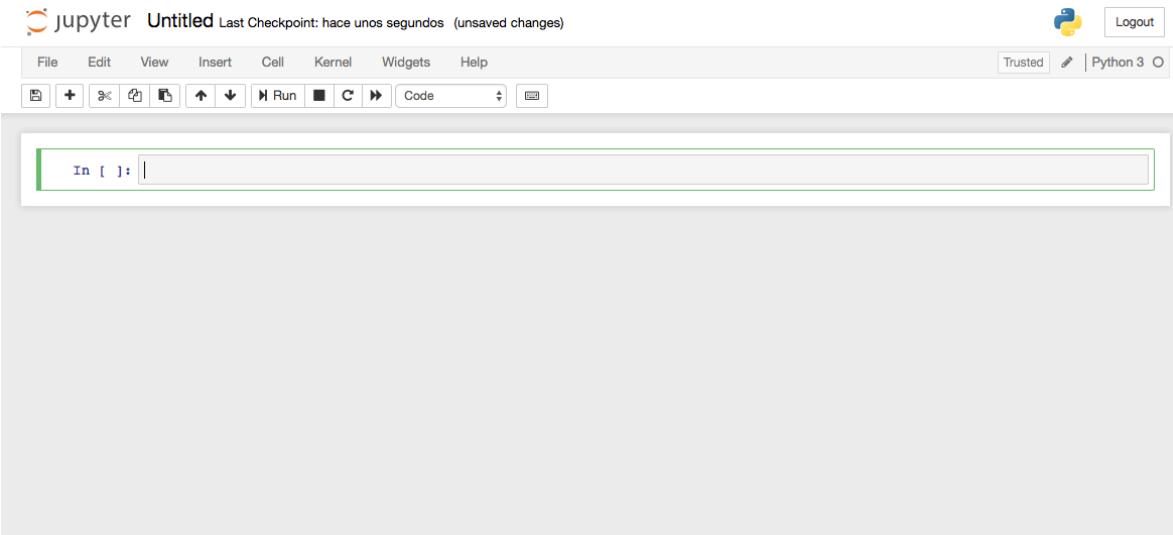
<http://bit.ly/2MxKDvb>

TRABAJANDO CON EL IDE JUPYTER NOTEBOOK

Jupyter Notebook lo puedes encontrar en el navegador de Anaconda “Anaconda-Navigator”, al igual que Spyder.



Una vez seleccionado para abrir Jupyter, se abre una pestaña en el navegador web predeterminado del computador. Desde acá se puede navegar a través de las carpetas del computador para determinar en donde se guardará el documento del proyecto. Se debe crear un nuevo Notebook de Python 3, en este momento se abre una nueva pestaña, acá es donde se debe empezar a desarrollar el proyecto.



Para efectos del libro todas las líneas de códigos y resultados de códigos se hará utilizando como IDE, Jupyter Notebook, esto se debe que la presentación de los datos es ideal para facilitar el entendimiento, pero puedes seleccionar cualquier IDE de tú preferencia.

Capítulo 3

CURSOS INTENSIVOS DE PYTHON Y SCIPY

Para desarrollar algoritmos de Machine Learning en Python no es necesario ser un desarrollador experto en Python, con solo tener las bases de cómo programar en algún lenguaje de programación se puede ser capaz de entender Python muy fácilmente, solamente necesitas conocer algunas propiedades del lenguaje y transferir lo que ya realmente sabe a Python.

En este capítulo aprenderás:

1. Sintaxis básica de Python.
2. Conocimientos básicos en la programación de las librerías NumPy, Pandas y Matplotlib.
3. Las bases para construir las tareas en Python para Machine Learning.

Si ya conoces algo de Python, este capítulo será un refrescamiento en los conocimientos.

CURSO INTENSIVO DE PYTHON

Python es una herramienta computacional poderosa que se puede utilizar para resolver tareas complicadas en el área de finanzas, economía, ciencia de los datos y Machine Learning.

Las ventajas técnicas que ofrece Python en comparación con otros lenguajes de programación son las siguientes:

- Es gratis y actualizado constantemente.
- Puede ser utilizado en múltiples dominios.
- No requiere de mucho tiempo para procesar cálculos y posee una sintaxis intuitiva que permite programar complejas líneas de código.

Con todas estas características hace que Python pueda ser implementando en un sin fin de aplicaciones.

Por lo tanto, Python basa su popularidad en dos pilares, el primero es que es fácil de aprender ya que contiene una sintaxis clara e intuitiva y la segunda razón es que es muy poderoso ya que puede ejecutar una variedad de complejas líneas de código.



Para mayor información se puede revisar la siguiente información:

<http://bit.ly/2JEOBnR>

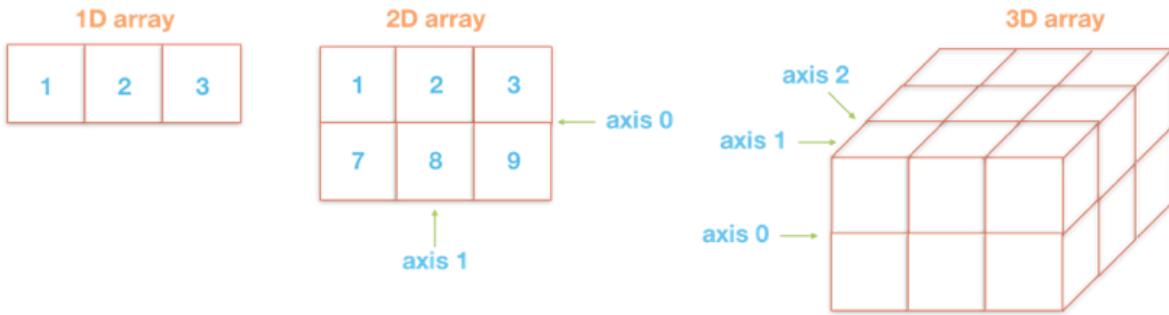
CURSO INTENSIVO DE NUMPY

NumPy es un paquete de Python que significa “Numerical Python”, es la librería principal para la informática científica, proporciona una estructura de datos de matriz que tiene algunos beneficios sobre las listas regulares de Python.



Por su parte, NumPy array o el arreglo de matrices de NumPy es un potente objeto de matriz N-dimensional que tiene forma de filas y columnas, en la que se tiene varios elementos que están almacenados en sus respectivas ubicaciones de memoria. En la figura se puede observar esto más claramente, esta es una matriz

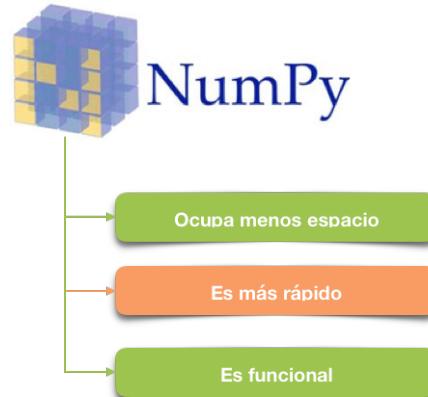
bidimensional porque tiene filas y columnas. Como se puede ver tiene cuatro filas y tres columnas, en el caso de que solo tuviera una hilera entonces habría sido una matriz unidimensional.



Según lo explicado anteriormente, en la primera figura se tiene una matriz unidimensional o 1D. En la segunda figura, se tiene una matriz bidimensional o 2D, en donde las filas se indican como el eje 0, mientras que las columnas son el eje 1.

El número de ejes aumenta de acuerdo con el número de dimensiones, en matrices 3D, se tendrá un eje 2, adicional. Estos ejes solo son válidos para matrices que tienen al menos dos dimensiones, ya que no tiene sentido esto para matrices unidimensional.

Entre las diferencias que tiene NumPy y las listas propias que ofrece Python para manejar los datos se encuentran que, NumPy ocupa menos memoria en comparación a las listas de Python, a su vez es bastante rápido en términos de ejecución y no se requiere mucho esfuerzo en la programación para ejecutar las rutinas. Por estas razones NumPy es mucho más fácil y conveniente su uso en el desarrollo de los algoritmos de Machine Learning.



Sabiendo todo esto, se crea dos matrices una unidimensional y otra bidimensional, esto utilizando por supuesto NumPy.

```
import numpy as np

a = np.array([1,2,3])
print('1D array:')
print(a)

b = np.array([(1,2,3), (4,5,6)])
print('2D array:')
print(b)
```

```
1D array:
[1 2 3]
2D array:
[[1 2 3]
 [4 5 6]]
```

Con la primera instrucción se indica al programa de Python que de ahora en adelante np será la referencia para todo lo referente a NumPy. De esta manera, muy simple, se declara las matrices dentro de NumPy.

NumPy contiene varias instrucciones para obtener más información sobre las matrices que se han creado utilizando esta librería, algunas de ellas son las siguientes:

Se puede encontrar el *tipo de datos* de los elementos que están almacenados en una matriz, para ello se utiliza la función “dtype”, el cual arrojará el tipo de datos junto con el tamaño.

```
#Conocer el tipo de datos
a = np.array([(1,2,3)])
print(a.dtype)
```

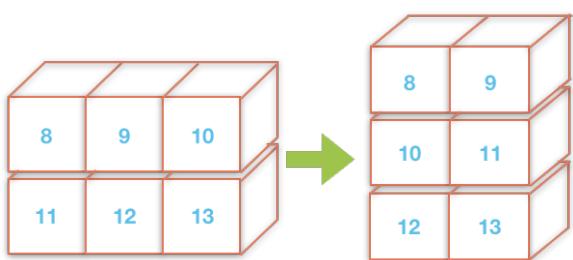
int64

Del mismo modo, se puede encontrar el *tamaño y la forma de la matriz* con la función “size” y “shape”, respectivamente.

```
#Conocer el tamaño de la matriz
a = np.array([(1,2,3,4,5,6)])
print(a.size)
print(a.shape)
```

6
(1, 6)

Otra de las operaciones que se puede realizar con NumPy es el *cambio de tamaño y forma de las matrices*.



El cambio de la forma de las matrices es cuando se cambia el número de filas y columnas que le da una nueva vista a un objeto.

En la imagen se tiene 3 columnas y 2 filas que se han convertido en 2 columnas y 3 filas utilizando la instrucción “reshape”.

```
#Cambio de forma de una matriz
a = np.array([(8,9,10), (11,12,13)])
print(a)
a = a.reshape(3,2)
print(a)
```

```
[[ 8  9 10]
 [11 12 13]]
[[ 8   9]
 [10  11]
 [12  13]]
```

Otra función que se puede utilizar con NumPy es *seleccionar un solo elemento de la matriz*, para ello solamente se tiene que especificar el número de columna y fila en donde está ubicado y devolverá el valor almacenado en dicha ubicación.

```
#Extraer un solo valor de la matriz - el valor ubicado en la fila 0 columna 2
a = np.array([(1,2,3,4), (3,4,5,6)])
print(a[0,2])
```

3

Con la librería de NumPy se puede realizar de manera muy fácil *operaciones aritméticas*. Se puede encontrar el valor mínimo, máximo y la suma de la matriz con NumPy, para ello utilizamos “min”, “max” y “sum”, respectivamente.

```
#Encontrar el mínimo, máximo y la suma
a = np.array([2,4,8])
print(a.min())
print(a.max())
print(a.sum())
```

2
8
14

Se puede realizar más operaciones con las matrices de NumPy, como son la suma, resta, multiplicación y división de las dos matrices.

```
#Calcular la suma, resta, multiplicación y división de dos matrices
x = np.array([(1,2,3), (3,4,5)])
y = np.array([(1,2,3), (3,4,5)])
print('Suma:')
print(x+y)
print('Resta:')
print(x-y)
print('Multiplicación:')
print(x*y)
print('División:')
print(x/y)
```

```
Suma:
[[ 2  4  6]
 [ 6  8 10]]
Resta:
[[ 0  0  0]
 [ 0  0  0]]
Multiplicación:
[[ 1  4  9]
 [ 9 16 25]]
División:
[[1.  1.  1.]
 [1.  1.  1.]]
```

Se ha cubierto gran parte de las funciones que se pueden utilizar al momento de manipular los datos con NumPy, aún faltan muchas más instrucciones, pero a medida

que se vaya programando seguramente se irán aprendiendo, recuerda que la página web propia de esta librería ofrece muy buena información sobre todas las funciones que ofrece por lo que no está demás visitarla.



Para mayor información se puede ver revisar la siguiente información:

<http://bit.ly/2MSJkqu>
<http://bit.ly/2QiAN2v>

CURSO INTENSIVO DE PANDAS

Pandas es una librería de código abierto de Python que proporciona herramientas de análisis y manipulación de datos de alto rendimiento utilizando sus potentes estructuras de datos. El nombre de Pandas se deriva del término “Panel Data” y es la librería de análisis de datos de Python.

Usando esta librería se puede lograr cinco pasos típicos en el procesamiento y análisis de datos, independientemente del origen de los datos: cargar, preparar, manipular, modelar y analizar.

Un DataFrame es la estructura fundamental de Pandas, estos son estructuras de datos etiquetados bidimensionales con columnas de tipos potencialmente diferentes. Los Pandas DataFrame constan de tres componentes principales: los datos, el índice y las columnas.

	1	2	3
A	6	5	4
B	7	8	9

Adicionalmente con la estructura Pandas DataFrame se puede especificar los nombres de índice y columna. El índice indica la diferencia en las filas, mientras que los nombres de las columnas indican la diferencia en las columnas. Estos componentes son muy útiles cuando se requiera manipular los datos.

Las características principales de la librería Pandas son:

- Objeto DataFrame rápido y eficiente con indexación predeterminada y personalizada.
- Herramientas para cargar datos en objetos de datos en memoria desde diferentes formatos de archivo.
- Alineación de datos y manejo integrado de datos faltantes.

- Remodelación y giro de conjuntos de fechas.
- Etiquetado, corte, indexación y subconjunto de grandes conjuntos de datos.
- Las columnas de una estructura de datos se pueden eliminar o insertar.
- Agrupa por datos para agregación y transformaciones.
- Alto rendimiento de fusión y unión de datos.
- Funcionalidad de la serie de tiempo.

Como cualquier otra librería en Python, se debe importarla dentro del programa, en este caso se suele usar el alias pd, mientras que para la librería NumPy se carga como np.

```
import numpy as np
import pandas as pd
```

Crear los DataFrames es el primer paso en cualquier proyecto de Machine Learning con Python, por lo que para crear una trama de datos desde una matriz NumPy se debe pasarla a la función DataFrame() en el argumento de datos.

```
data = np.array([[1, 'Col1', 'Col2'], ['Fila1', 11, 22], ['Fila2', 33, 44]])
print(pd.DataFrame(data[1:, 1:], index = data[1:, 0], columns = data[0, 1:]))
```

```
      Col1  Col2
Fila1    11    22
Fila2    33    44
```

Si se observa el resultado de este código, se fragmenta los elementos seleccionados de la matriz NumPy para construir el DataFrame, primero se selecciona los valores que figuran en las listas que comienza con Fila1 y Fila2, luego selecciona el índice o los números de fila Fila1 y Fila2 y luego los nombres de las columnas Col1 y Col2.

La manera en que se crea este DataFrame será la misma para todas las estructuras.

```
df = pd.DataFrame(np.array([[1,2,3], [4,5,6]]))
print(df)
```

```
   0  1  2
0  1  2  3
1  4  5  6
```

Una vez creado el DataFrame se puede explorarlo con todas las instrucciones con las que Pandas cuenta. Lo primero que se debe hacer es *conocer la forma de los datos* para ello se utiliza la instrucción shape. Con esta instrucción se puede conocer las dimensiones del DataFrame, es decir el ancho y altura.

```
df = pd.DataFrame(np.array([[1,2,3], [4,5,6]]))
print('Forma del DataFrame:')
print(df.shape)
```

Forma del DataFrame:
(2, 3)

Por otra parte, se puede utilizar la función len() en combinación con la instrucción index para conocer la *altura* del DataFrame.

```
df = pd.DataFrame(np.array([[1,2,3], [4,5,6]]))
print('Altura del DataFrame:')
print(len(df.index))
```

Altura del DataFrame:
2



Para mayor información se puede revisar la siguiente información:

<http://bit.ly/2yibciB>
<http://bit.ly/2NFsYCa>

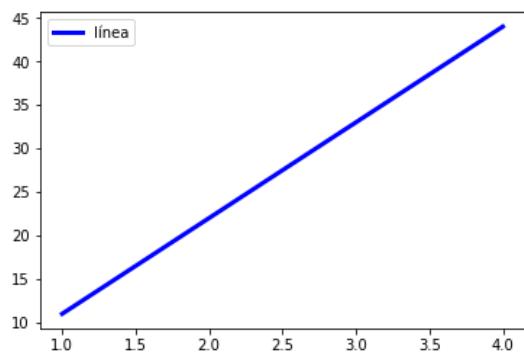
CURSO INTENSIVO DE MATPLOTLIB

Matplotlib es una librería de trazado utilizada para gráficos 2D en lenguaje de programación Python, es muy flexible y tiene muchos valores predeterminados incorporados que ayudan muchísimo en el trabajo. Como tal, no se necesita mucho para comenzar, solamente se tiene que hacer las importaciones necesarias, preparar algunos datos y con esto se puede comenzar a trazar la función con la ayuda de la instrucción plot().

```
import matplotlib.pyplot as plt

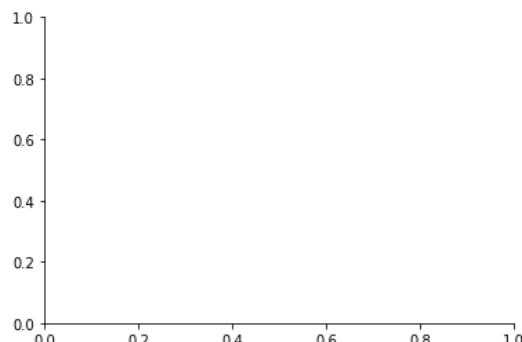
a = [1,2,3,4]
b = [11,22,33,44]

plt.plot(a, b, color='blue', linewidth=3, label='línea')
plt.legend()
plt.show()
```



Al momento de graficar se debe tener en cuenta las siguientes consideraciones:

- La figura es la ventana o página general en la que se dibuja todo, es el componente de nivel superior de todo lo que se considerará en los siguientes puntos. Se puede crear múltiples figuras independientes. Una figura puede tener otras cosas como por ejemplo un subtítulo, que es un título centrado de la figura, una leyenda, una barra de color, entre otros.
- A la figura se le agrega los ejes. Los ejes es el área en la que se grafican los datos con funciones tales como `plot()` y `scatter()` y que pueden tener etiquetas asociadas. Las figuras pueden contener múltiples ejes.
- Cada eje tiene un eje x y otro eje y, y cada uno de ellos contiene una numeración. También existen las etiquetas de los ejes, el título y la leyenda que se deben tener en cuenta cuando se quiere personalizar los ejes, pero también teniendo en cuenta que las escalas de los ejes y las líneas de la cuadrícula pueden ser útiles.
- Las líneas vertebrales son líneas que conectan las marcas de eje y que designan los límites del área de datos, en otras palabras, son el simple cuadrado que puedes ver cuando has inicializado los ejes, como en la siguiente imagen:



Como se puede observar los extremos derecho y superior están configurados como invisibles.



Para mayor información se puede revisar la siguiente información:

<http://bit.ly/2CwmX99>
<http://bit.ly/2AuS57z>

Capítulo 4

CARGAR LOS DATOS

El primer paso al comenzar un proyecto de Machine Learning es el de cargar los datos. Existe muchas formas para hacer esto, pero lo primero que se debe ver es el formato en que se encuentran estos datos, por lo general se encuentran en archivos CSV, por lo que acá se explicará cómo cargar un archivo CSV en Python, aunque también se mostrará como hacerlo en caso de que tengas otro formato.

En este capítulo aprenderás:

1. Cargar archivos CSV utilizando Python.
2. Cargar archivos utilizando Python.

CONSIDERACIONES DE LOS ARCHIVOS CSV

Varias son las consideraciones que se deben tomar en cuenta al momento de cargar los datos desde un archivo CSV, a continuación, se explican algunas de ellas.

Encabezado de archivo

En ocasiones los datos cuentan con un encabezado, esto ayuda en la asignación automática de nombres a cada columna de datos. En caso de que no se cuente con este encabezado, se puede nombrar sus atributos de manera manual. De igual manera, debes especificar explícitamente si el archivo a utilizar cuenta o no con un encabezado.

Delimitador

El delimitador que se usa de manera estándar para separar los valores en los campos es la coma (,). El archivo a utilizar podría utilizar un delimitador diferente como un espacio en blanco o punto y coma (;), en estos casos se debe especificar explícitamente.

CONJUNTO DE DATOS: DIABETES DE LOS INDIOS PIMA

El conjunto de datos o dataset de los indios Pima se utilizará para demostrar la carga de datos en este capítulo. También será utilizado en muchos de los próximos capítulos.

El objetivo de este conjunto de datos es predecir si un paciente tiene diabetes o no, según ciertas mediciones de diagnóstico incluidas en el conjunto de datos. En particular, todos los pacientes aquí son mujeres de al menos 21 años de la herencia india Pima.

Los datos están disponibles gratuitamente en la página de Kaggle bajo el nombre de “Pima Indians Diabetes Database”. Para descargar el archivo se debe estar suscrito a Kaggle.



Para mayor información se puede revisar la siguiente información:

<http://bit.ly/2wqGog6>

CARGAR ARCHIVOS CSV UTILIZANDO PYTHON

Para cargar los datos CSV se utilizará la librería de Pandas y la función “pandas.read_csv()”. Esta función es muy flexible y es la manera más rápida y fácil para cargar los datos en Machine Learning. La función devuelve un DataFrame de Pandas que se puede comenzar a utilizar de manera inmediata.

En el siguiente ejemplo se asume que el archivo diabetes.csv se encuentra guardado en el directorio de trabajo actual.

```
import pandas as pd
### CARGAR LOS DATOS ###
data = pd.read_csv('diabetes.csv')
```

También se puede cargar los datos directamente desde una dirección web, para esto solamente se tiene que incluir la dirección web dentro del paréntesis donde en estos momentos se tiene el nombre del archivo.

CARGAR ARCHIVOS UTILIZANDO PYTHON

En caso de que los archivos de Machine Learning no se encuentren en formato CSV, la librería Pandas ofrece la opción de cargar datos en distintos formatos. A continuación, una lista y las instrucciones que se debe utilizar:

Descripción	Comando
HTML	read_html
MS EXCEL	read_excel
JSON	read_json
SQL	read_sql

Capítulo 5

ENTENDIENDO LOS DATOS

Una vez cargados los datos deben ser comprendidos para obtener los mejores resultados. En este capítulo se explicará varias formas en que se puede usar Python para comprender mejor los datos de Machine Learning.

En este capítulo se aprenderás:

1. Verificar los datos en bruto.
2. Revisar las dimensiones del conjunto de datos.
3. Revisar los tipos de datos de las características.
4. Verificar la distribución de características entre las clases en el conjunto de datos.
5. Verificar los datos utilizando estadísticas descriptivas.
6. Comprender las relaciones en los datos utilizando correlaciones.
7. Revisar el sesgo de las distribuciones de cada característica.

Para implementar cada uno de estos ítems se utilizará el conjunto de datos con el que se trabajo en el anterior capítulo correspondiente a la diabetes de los indios Pima.

VERIFICAR LOS DATOS

Mirar los datos en bruto puede revelar información que no se puede obtener de otra manera. También puede mostrar formas que luego pueden convertirse en ideas sobre cómo procesar mejor y manejar los datos para tareas de Machine Learning.

Para visualizar las primeras 5 filas de los datos se utiliza la función head() de Pandas DataFrame.

```
#### ENTENDIENDO LOS DATOS ####  
#Ver las primeras filas  
data.head()
```

Resultado una vez ejecutado el código anterior:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	
0	6	148	72	35	0	33.6		0.627	50	1
1	1	85	66	29	0	26.6		0.351	31	0
2	8	183	64	0	0	23.3		0.672	32	1
3	1	89	66	23	94	28.1		0.167	21	0
4	0	137	40	35	168	43.1		2.288	33	1

DIMENSIONANDO LOS DATOS

Para trabajar en Machine Learning se debe tener un buen control sobre la cantidad de datos que se tiene, conociendo la cantidad de filas y de columnas.

- Si se tiene demasiadas filas, algunos algoritmos pueden tardar demasiado en entrenar. En cambio, si se tiene muy pocos, de repente no sean suficientes datos para entrenar los algoritmos.
- Si se tiene demasiadas características, algunos algoritmos pueden distraerse o sufrir un rendimiento deficiente debido a la dimensionalidad.

Para conocer la forma y el tamaño del conjunto de datos utilizamos la propiedad *shape* de Pandas.

```
#Dimensión de los datos  
data.shape
```

Resultado una vez ejecutado el código anterior:

(768, 9)

El resultado de este código representa el número de filas y luego las columnas. Por lo tanto, el conjunto de datos cuenta con 768 filas y 9 columnas.

TIPOS DE DATOS

Conocer el tipo de dato de cada característica es importante. Conociendo esta información se puede hacer una idea si se deban convertir los datos originales en otros formatos para que sean más fácil su implementación junto a los algoritmos de Machine Learning.

Para conocer los tipos de datos se puede implementar la función *dtypes* de Pandas, de esta forma se puede conocer el tipo de datos de cada característica que forma parte del conjunto de datos.

```
#Tipos de datos  
data.dtypes
```

Resultado una vez ejecutado el código anterior:

```
Pregnancies          int64  
Glucose             int64  
BloodPressure       int64  
SkinThickness       int64  
Insulin             int64  
BMI                 float64  
DiabetesPedigreeFunction float64  
Age                 int64  
Outcome             int64  
dtype: object
```

El resultado es una lista con cada una de las columnas del conjunto de datos, en donde se especifica el tipo de datos que se maneja.

DESCRIPCIÓN ESTADÍSTICA

La descripción estadística se refiere a la información que se puede obtener de los datos referentes a propiedades estadísticas de cada característica. Para implementarlo solamente se debe utilizar la función *describe()* de Pandas, y las propiedades que devuelve son las siguientes:

- Conteo
- Media
- Desviación estándar
- Valor mínimo
- 25%
- 50%
- 75%
- Valor máximo

```
#Descripción estadística
data.describe()
```

Resultado una vez ejecutado el código anterior:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

Como se puede observar devuelve mucha información del conjunto de datos, pero lo importante de acá es verificar si todas las cantidades de datos coinciden, de esta forma se puede saber si hay o no datos perdidos.

DISTRIBUCIÓN DE CLASES (PROBLEMAS DE CLASIFICACIÓN)

En caso de que se este trabajando con un problema de clasificación, como es el caso actual, se debe conocer qué tan equilibrados están los valores de clases. Cuando los problemas se encuentran altamente desequilibrados (muchas más observaciones para una clase que para otra) se requieren un tratamiento especial al momento de realizar el procesamiento de los datos. Para conocer esta información se puede implementar una función de Pandas, *groupby* junto con la columna *Class* del conjunto de datos con el que se está trabajando.

```
#Distribución de clases
conteo_data = data.groupby('Outcome').size()
print(conteo_data)
```

Resultado una vez ejecutado el código anterior:

```
Outcome
0    500
1    268
dtype: int64
```

La clase 0 corresponde a las personas no diabéticas mientras que la clase 1 corresponden a las mujeres que tienen la enfermedad.

CORRELACIÓN ENTRE CARACTERÍSTICAS

La correlación se refiere a la relación entre dos variables y cómo pueden o no cambiar juntas. Para implementar esto se debe indicar el método a utilizar, el más común es el coeficiente de **correlación de Pearson**, que asume una distribución normal de los atributos involucrados. Si la correlación se encuentra entre -1 o 1 muestra una correlación negativa o positiva completa, respectivamente. Mientras que un valor de 0 no muestra ninguna correlación en absoluto. Para calcular una matriz de correlación se utiliza la función corr().

```
#Correlación entre características
correlacion = data.corr(method='pearson')
print(correlacion)
```

Resultado una vez ejecutado el código anterior:

	preg	plas	pres	skin	test	mass	pedi	age	class
preg	1.000	0.129	0.141	-0.082	-0.074	0.018	-0.034	0.544	0.222
plas	0.129	1.000	0.153	0.057	0.331	0.221	0.137	0.264	0.467
pres	0.141	0.153	1.000	0.207	0.089	0.282	0.041	0.240	0.065
skin	-0.082	0.057	0.207	1.000	0.437	0.393	0.184	-0.114	0.075
test	-0.074	0.331	0.089	0.437	1.000	0.198	0.185	-0.042	0.131
mass	0.018	0.221	0.282	0.393	0.198	1.000	0.141	0.036	0.293
pedi	-0.034	0.137	0.041	0.184	0.185	0.141	1.000	0.034	0.174
age	0.544	0.264	0.240	-0.114	-0.042	0.036	0.034	1.000	0.238
class	0.222	0.467	0.065	0.075	0.131	0.293	0.174	0.238	1.000

Algunos algoritmos de Machine Learning, como regresión lineal y logística, pueden sufrir un rendimiento deficiente si existen atributos altamente correlacionados en el conjunto de datos.

CONSIDERACIONES ADICIONALES

Los puntos acá tratados son solamente algunos consejos que se deben considerar al momento de revisar los datos, adicionalmente a estos, se debe tomar en consideración lo siguiente:

- Repasar los números. Generar la descripción estadística no es suficiente. Se debe tomar un tiempo para leer y entender muy bien los números que se está viendo.
- Al ver los números se debe entender muy bien cómo y por qué se están viendo estos números específicos, cómo se relacionan con el dominio del problema en general, lo importante acá es hacerse pregunta y conocer muy bien de qué se trata la información que nos presenta el conjunto de datos.
- Es aconsejable escribir todas las observaciones que se obtengan o ideas que surjan al momento de analizar los datos. En ocasiones esta información será muy importante cuando se este tratando de idear nuevas cosas para probar.

Capítulo 6

VISUALIZANDO LOS DATOS

Como se explico en el anterior capítulo se debe entender los datos para obtener buenos resultados al momento de implementar los algoritmos de Machine Learning. Otra forma de aprender de los datos es implementando la visualización de los mismos.

En este capítulo aprenderás:

1. Graficar con una sola variable
2. Graficar con múltiples variables

GRÁFICAR CON UNA SOLA VARIABLES

Las gráficas con una sola variable son útiles para comprender cada característica del conjunto de datos de forma independiente. Acá se explicarán las siguientes:

- Histogramas
- Diagrama de cajas o Box Plot

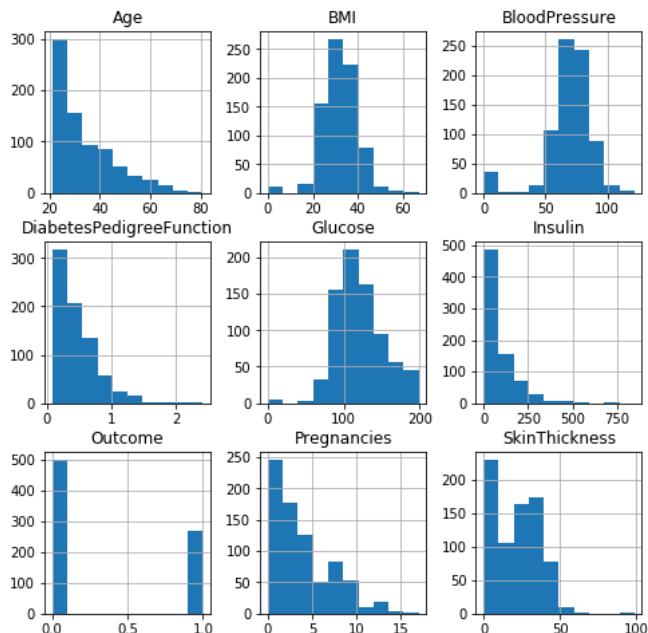
Histograma

Una forma rápida de tener una idea de la distribución de cada atributo es observar los histogramas. Estos son utilizados para presentar la distribución y las relaciones de una sola variable en un conjunto de características. Los histogramas agrupan los datos en

contenedores y de la forma de los mismos, se puede obtener rápidamente una idea de si un atributo es gaussiano, sesgado o inclusive tiene una distribución exponencial. Inclusive puede ayudar a determinar si hay posibles valores atípicos.

```
import matplotlib.pyplot as plt  
  
#Histograma  
data.hist(figsize = (8, 8))  
plt.show()
```

Resultado una vez ejecutado el código anterior:

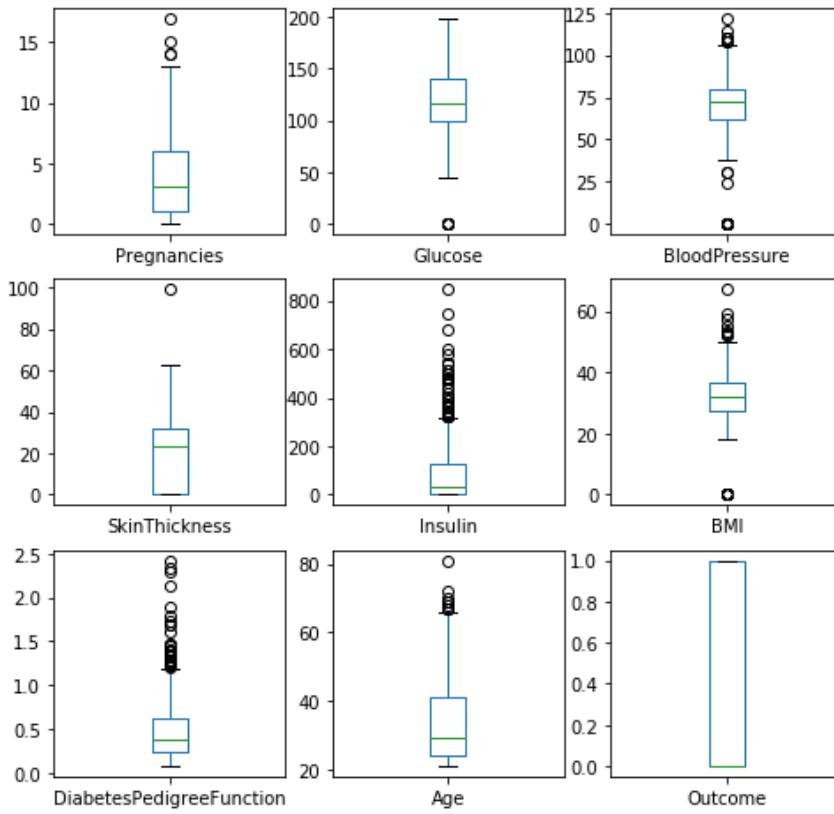


Diagramas de cajas o Box Plot

Otra forma de revisar la distribución de cada atributo es usar los diagramas de cajas. Esto resume la distribución de cada atributo, trazando una línea para la mediana o valor medio y un cuadro alrededor de los percentiles 25 y 75.

```
#Diagrama de cajas  
data.plot(kind='box', subplots=True, layout=(3,3), sharex=False, sharey=False, figsize = (8, 8))  
plt.show()
```

Resultado una vez ejecutado el código anterior:



Para mayor información se puede revisar la siguiente información:

<http://bit.ly/2XgTTOg>

GRÁFICAR CON VARIAS VARIABLES

Esta sección proporciona ejemplo de un gráfico que muestra las interacciones entre múltiples variables en el conjunto de datos:

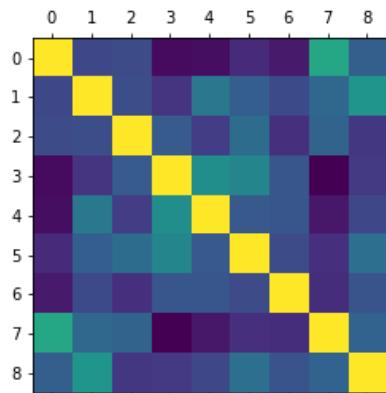
→ Matriz de correlación

Matriz de correlación

La correlación presenta una indicación de qué tan relacionados están los cambios entre dos variables. La correlación positiva se refiere a cuando dos variables cambian en la misma dirección. En cambio, la correlación es negativa cuando cambian en direcciones opuestas, una sube y la otra baja. La matriz de correlación se refiere a la correlación entre cada par de atributos, de esta forma se puede verificar qué variables tienen una alta correlación.

```
#Matriz de Correlación  
plt.matshow(data.corr())  
plt.show
```

Resultado una vez ejecutado el código anterior:



Si se analiza la matriz, ésta es simétrica, es decir, la parte inferior izquierda de la matriz es la misma que la superior derecha. Esto es útil ya que podemos ver dos vistas diferentes en los mismos datos en una gráfica.



Para mayor información se puede revisar la siguiente información:

<http://bit.ly/2XgTTOg>

Capítulo 7

PROCESAMIENTO DE LOS DATOS

El procesamiento de los datos es un paso fundamental en cualquier problema de Machine Learning, debido a que los algoritmos hacen suposiciones sobre los datos, por lo que los mismos deben estar presentados de manera correcta.

En este capítulo aprenderás a cómo preparar los datos para Machine Learning en Python utilizando scikit-learn:

1. Estandarizar los datos
2. Normalizar los datos
3. Eliminación de columnas

SEPARACIÓN DE LOS DATOS

En este capítulo se explicará varias formas para el procesamiento de datos para Machine Learning. El conjunto de datos que se utilizará será el de la diabetes de los indios Pima.

Antes de empezar el procesamiento de los datos es recomendable separar el conjunto de datos con las variables de entrada y salida o, como también se le conoce, las variables independientes y dependientes.

Observando el conjunto de datos con el que se ha venido trabajando, la variable de salida o dependiente vendría siendo la columna “Outcome”, debido a que en esta se refleja si la persona tiene diabetes o no.

Definido esto se procede a realizar la separación de los datos, creando la variable “X” para los datos de entrada o independientes y la variable “y” para la columna correspondiente a “Outcome”.

```
#Datos de entrada o independientes
X = data.drop(['Outcome'], axis=1)
X.head()
```

Resultado una vez ejecutado el código anterior:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.6		0.627 50
1	1	85	66	29	0	26.6		0.351 31
2	8	183	64	0	0	23.3		0.672 32
3	1	89	66	23	94	28.1		0.167 21
4	0	137	40	35	168	43.1		2.288 33

```
#Datos de salida o dependientes
y = data['Outcome']
y.head()
```

Resultado una vez ejecutado el código anterior:

```
0    1
1    0
2    1
3    0
4    1
Name: Outcome, dtype: int64
```

Para el procesamiento de datos se hará solamente para el conjunto correspondiente a la variable “X”, los datos de entrada o independientes. Los datos de la variable “y” no se le hará ningún procedimiento.

ESTANDARIZACIÓN DE LOS DATOS

La estandarización de los datos es un requisito común para muchos estimadores de Machine Learning, esto se debe a que los algoritmos se pueden comportar mal si las características individuales no se parecen más o menos a los datos estándar normalmente distribuidos.

Para cumplir con este procedimiento se utiliza la función *StandardScaler* de la librería scikit-learn y se aplica, en este caso a todos los datos del conjunto de datos “X”.

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

X_estandar = StandardScaler().fit_transform(X)
print(X_estandar[0:5,:])
```

Resultado una vez ejecutado el código anterior:

```
[[ 0.63994726  0.84832379  0.14964075  0.90726993 -0.69289057  0.20401277
   0.46849198  1.4259954 ]
 [-0.84488505 -1.12339636 -0.16054575  0.53090156 -0.69289057 -0.68442195
  -0.36506078 -0.19067191]
 [ 1.23388019  1.94372388 -0.26394125 -1.28821221 -0.69289057 -1.10325546
   0.60439732 -0.10558415]
 [-0.84488505 -0.99820778 -0.16054575  0.15453319  0.12330164 -0.49404308
  -0.92076261 -1.04154944]
 [-1.14185152  0.5040552  -1.50468724  0.90726993  0.76583594  1.4097456
   5.4849091 -0.0204964 ]]
```

Nota: al ejecutar este código probablemente salga un error, el mensaje lo que indica es que todos los datos del tipo entero y flotante han sido convertidos en flotante para poder implementar la instrucción *StandardScaler*.

Como se puede observar al ejecutar el código genera una matriz del tipo NumPy con los datos estandarizados, en caso de que se quieran convertir los datos en DataFrame de Pandas se debe ejecutar las siguientes líneas de código adicionales.

```
X_pandas = pd.DataFrame(data = X_estandar, columns= X.columns)
X_pandas.head()
```

Resultado una vez ejecutado el código anterior:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	0.639947	0.848324	0.149641	0.907270	-0.692891	0.204013	0.468492	1.425995
1	-0.844885	-1.123396	-0.160546	0.530902	-0.692891	-0.684422	-0.365061	-0.190672
2	1.233880	1.943724	-0.263941	-1.288212	-0.692891	-1.103255	0.604397	-0.105584
3	-0.844885	-0.998208	-0.160546	0.154533	0.123302	-0.494043	-0.920763	-1.041549
4	-1.141852	0.504055	-1.504687	0.907270	0.765836	1.409746	5.484909	-0.020496



Para mayor información se puede revisar la siguiente información:

<http://bit.ly/2Hjq2f4>
<http://bit.ly/2H63A9R>

NORMALIZACIÓN DE LOS DATOS

Normalizar los datos se refiere a cambiar la escala de los datos de la característica para que tengan una longitud de 1. Este método de procesamiento puede ser útil para conjuntos de datos dispersos (muchos ceros) con atributos de diferentes escalas.

Para implementar este método se utiliza la clase *Normalizer* de la librería scikit-learn.

```
#Normalización de los datos
from sklearn.preprocessing import Normalizer

X_normalizar = Normalizer().fit_transform(X)
print(X_normalizar[0:5,:])
```

Resultado una vez ejecutado el código anterior:

```
[[ 0.03355237  0.82762513  0.40262844  0.19572216  0.          0.18789327
   0.00350622  0.27960308]
 [ 0.008424    0.71604034  0.55598426  0.24429612  0.          0.22407851
   0.00295683  0.26114412]
 [ 0.04039768  0.92409698  0.32318146  0.          0.          0.11765825
   0.00339341  0.16159073]
 [ 0.00661199  0.58846737  0.43639153  0.15207584  0.62152733  0.185797
   0.0011042   0.13885185]
 [ 0.          0.5963863   0.17412739  0.15236146  0.73133502  0.18762226
   0.00996009  0.14365509]]
```

Igual que en el caso anterior, al ejecutar el código genera una matriz del tipo NumPy con los datos normalizados, en caso de que se quieran convertir los datos en DataFrame de Pandas se debe ejecutar las siguientes líneas de código adicionales.

```
X_pandas = pd.DataFrame(data = X_normalizar, columns= X.columns)
X_pandas.head()
```

Resultado una vez ejecutado el código anterior:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabetes	PedigreeFunction	Age
0	0.033552	0.827625	0.402628	0.195722	0.000000	0.187893		0.003506	0.279603
1	0.008424	0.716040	0.555984	0.244296	0.000000	0.224079		0.002957	0.261144
2	0.040398	0.924097	0.323181	0.000000	0.000000	0.117658		0.003393	0.161591
3	0.006612	0.588467	0.436392	0.152076	0.621527	0.185797		0.001104	0.138852
4	0.000000	0.596386	0.174127	0.152361	0.731335	0.187622		0.009960	0.143655



Para mayor información se puede revisar la siguiente información:

<http://bit.ly/2Hjq2f4>

<http://bit.ly/2H63A9R>

ELIMINACIÓN DE COLUMNAS

Otra forma de procesar los datos es eliminando columnas con datos que no son necesarios para el análisis. La función a utilizar para cumplir con esta forma es *drop*, junto con el nombre de la columna a eliminar.

Para realizar la respectiva prueba se eliminará la columna “BloodPressure” del conjunto de datos con el que se está trabajando.

```
#Eliminación de columnas
X = data.drop(['BloodPressure'],axis=1)
X.head()
```

Resultado una vez ejecutado el código anterior:

	Pregnancies	Glucose	SkinThickness	Insulin	BMI	Diabetes	PedigreeFunction	Age	Outcome	
0	6	148	35	0	33.6			0.627	50	1
1	1	85	29	0	26.6			0.351	31	0
2	8	183	0	0	23.3			0.672	32	1
3	1	89	23	94	28.1			0.167	21	0
4	0	137	35	168	43.1			2.288	33	1



Para mayor información se puede revisar la siguiente información:

<http://bit.ly/2Hjq2f4>

<http://bit.ly/2H63A9R>

Capítulo 8

AQUÍ +2-9±2021

SELECCIONANDO CARACTERÍSTICAS

Las características de datos que se utiliza para entrenar los modelos de Machine Learning tienen una gran influencia en el rendimiento que puede lograr. Las características irrelevantes o parcialmente relevantes pueden afectar negativamente el rendimiento del modelo. En este capítulo, se describirán las técnicas de selección automática de características que pueden ser utilizadas para preparar los datos.

En este capítulo aprenderás:

1. Método de filtro
2. Método de envoltura
3. Métodos integrados

IMPORTANCIA DE LAS CARACTERÍSTICAS

Los conjuntos de datos en ocasiones pueden ser pequeños mientras que otros son extremadamente grandes en tamaño, en especial cuando cuentan con un gran número de características, ocasionando que sean muy difícil de procesar.

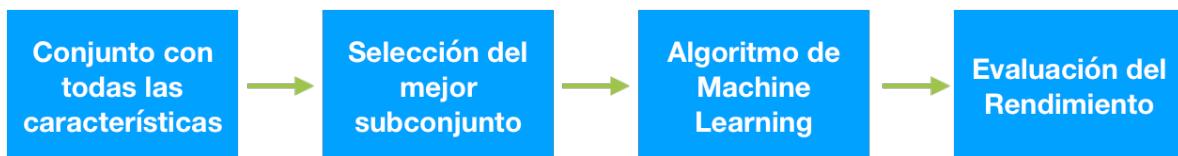
Cuando se tiene este tipo de conjuntos de datos de alta dimensión y se utilizan todas para la creación de modelos de Machine Learning puede ocasionar:

- Las características adicionales actúan como un ruido para el cual el modelo de Machine Learning puede tener un rendimiento extremadamente bajo.
- El modelo tarda más tiempo en entrenarse.
- Asignación de recursos innecesarios para estas características.

Por todo esto, se debe implementar la selección de características en los proyectos de Machine Learning.

MÉTODOS DE FILTRO

La siguiente imagen describe mejor los métodos de selección de características basados en filtros:



Los métodos de filtro se utilizan generalmente como un paso de procesamiento de datos, la selección de características es independiente de cualquier algoritmo de Machine Learning.

Las características se clasifican según los puntajes estadísticos que tienden a determinar la correlación de las características con la variable de resultado, la correlación es un término muy contextual y varía de un trabajo a otro.

En la siguiente tabla se puede utilizar para definir los coeficientes de correlación para diferentes tipos de datos, en este caso, continuo y categórico.

↓Características/Predicción →	Continuo	Categórico
Continuo	Correlación de Pearson	LDA
Categórico	Anova	Chi-cuadrado

Correlación de Pearson: se usa como una medida para cuantificar la dependencia lineal entre dos variables continuas X e Y, su valor varía de -1 a +1.

LDA: el análisis discriminante lineal se usa para encontrar una combinación lineal de características que caracteriza o separa dos o más clases, o niveles, de una variable categórica.

ANOVA: significa análisis de la varianza y es similar a LDA, excepto por el hecho de que opera mediante una o más funciones independientes categóricas y una función dependiente continua. Proporciona una prueba estadística de si las medias de varios grupos son iguales o no.

Chi-cuadrado: es una prueba estadística que se aplica a los grupos de características categóricas para evaluar la probabilidad de correlación o asociación entre ellos utilizando su distribución de frecuencia.

Los métodos de filtro no eliminan la multicolinealidad, por lo tanto, se debe lidiar con ellos también antes de entrenar modelos para tus datos.

De manera práctica lo explicado anteriormente, se implementará una prueba estadística de Chi-cuadrado para características no negativas, para seleccionar 5 de las mejores características del conjunto de datos con el que se ha venido trabajando correspondientes a la diabetes de los indios Pima.

La librería scikit-learn proporciona la clase *SelectKBest* que se puede usar con un conjunto de diferentes pruebas estadísticas para seleccionar un número específico de funciones, en este caso, es Chi-cuadrado.

```
#Métodos de filtro
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
import numpy as np

#Extracción de características: chi-cuadrado
prueba = SelectKBest(score_func=chi2, k=5)
entrenamiento = prueba.fit(X, Y)

#Puntaje de características: chi-cuadrado
np.set_printoptions(precision=3)
print(entrenamiento.scores_)
```

Resultado una vez ejecutado el código anterior:

```
[ 111.52  1411.887   17.605    53.108  2175.565   127.669    5.393  181.304]
```

El resultado obtenido en esta parte del programa es el puntaje que tiene cada una de las características una vez aplicado el método Chi-cuadrado. De acuerdo a esto ya se puede definir las más importantes características que pueden afectar el análisis con Machine Learning.

```
#Características seleccionadas: chi-cuadrado
características = entrenamiento.transform(X)
print(características[0:5,:])
```

Resultado una vez ejecutado el código anterior:

```
[[ 6. 148. 0. 33.6 50. ]
 [ 1. 85. 0. 26.6 31. ]
 [ 8. 183. 0. 23.3 32. ]
 [ 1. 89. 94. 28.1 21. ]
 [ 0. 137. 168. 43.1 33. ]]
```

Resumiendo, los resultados obtenidos, se puede observar las 5 características elegidas, tomadas con los puntajes más altos: “Pregnancies” (embarazos), “Glucose” (glucosa), “Insulin” (insulina), “BMI” (índice de masa corporal), “Age” (años).

Estos puntajes ayudan a determinar las mejores características para entrenar el modelo.



Para mayor información se puede revisar la siguiente información:

<http://bit.ly/2CoBojO>

MÉTODOS DE ENVOLTURA

Al igual que los métodos de filtro, se muestra una gráfica en donde se explica mejor este método:



Como se puede observar, **un método de envoltura necesita un algoritmo de Machine Learning y utiliza su rendimiento como criterio de evaluación**. Este método busca una característica que sea más adecuada para el algoritmo y tiene como objetivo mejorar el rendimiento.

Por lo tanto, se trata de usar un subconjunto de características y se entrena un modelo usándolo, basándose en las inferencias que se extrae del modelo anterior, se decide agregar o eliminar características de su subconjunto. El problema se reduce esencialmente a un problema de búsqueda. Estos métodos suelen ser computacionalmente muy caros.

Algunos ejemplos comunes de Métodos de Envoltura son los siguientes:

Selección hacia delante (Forward Selection): es un método iterativo en el que se comienza sin tener ninguna característica en el modelo. En cada iteración, se sigue agregando la función que mejor mejora nuestro modelo hasta que la adición de una nueva variable no mejore el rendimiento del modelo.

Eliminación hacia atrás (Backward Selection): se comienza con todas las características y se elimina la característica menos significativa en cada iteración, lo que mejora el rendimiento del modelo. Se repite esto hasta que no se observe ninguna mejora en la eliminación de características.

Eliminación de características recursivas (Recursive Feature Elimination): es un algoritmo de optimización que busca encontrar el subconjunto de funciones con mejor rendimiento. Crea repetidamente modelos y deja de lado la mejor o la peor característica de rendimiento en cada iteración. Construye el siguiente modelo con las características de la izquierda hasta que se agotan todas las características, luego clasifica las características según el orden de su eliminación.

Aplicando esta teoría al conjunto de datos que se ha trabajado hasta los momentos, se aplicará el método de la eliminación de características recursivas junto con el algoritmo de regresión logística.

```
#Métodos de envoltura
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

#Extracción de características
modelo = LogisticRegression()
rfe = RFE(modelo, 4)
entrenamiento = rfe.fit(X, Y)
print("Número de características: %s" % (entrenamiento.n_features_))
print("Características seleccionadas: %s" % (entrenamiento.support_))
print("Clasificación de características: %s" % (entrenamiento.ranking_))
```

Resultado una vez ejecutado el código anterior:

```
Num Características: 4
Características seleccionadas: [ True  True False False  True  True False]
Clasificación de Características: [1 1 2 4 5 1 1 3]
```

Para este método se eligió las 4 funciones principales y el resultado fue el siguiente: “Pregnancies” (embarazos), “Glucose” (glucosa), “BMI” (índice de masa corporal) y “DiabetesPedigreeFunction” (función pedigrí de la diabetes).

Como se puede observar estos datos se encuentran marcadas como Verdaderos (True) en la matriz de características seleccionadas y como 1 en la matriz de clasificación de características.



Para mayor información se puede revisar la siguiente información:

<http://bit.ly/2CoBojO>

MÉTODOS INTEGRADOS

Combina las cualidades de los métodos de filtro y envoltura. Se implementa mediante algoritmos que tienen sus propios métodos de selección de características incorporados.

Algunos de los ejemplos más populares de estos métodos son la regresión LASSO y RIDGE, que tienen funciones de penalización incorporadas para reducir el sobreajuste.



Para mayor información se puede revisar la siguiente información:

<http://bit.ly/2CoBojO>

Capítulo 9

ALGORITMOS DE CLASIFICACIÓN

Al desarrollar un proyecto de Machine Learning no se puede saber de antemano qué algoritmos son los más adecuados para el problema, se debe probar varios métodos y centrar la atención en aquellos que demuestran ser los más prometedores. En este capítulo, se explicará cómo implementar los algoritmos de Machine Learning que puede utilizarse para resolver problemas de clasificación en Python con scikit-learn.

En este capítulo aprenderás cómo implementar los siguientes algoritmos:

1. Regresión logística
2. K Vecinos más cercanos
3. Máquinas de vectores de soporte
4. Naive bayes
5. Árboles de decisión clasificación
6. Bosques aleatorios clasificación

Para este análisis se tomará el conjunto de datos de diabetes de los indios Pima. De igual forma, se asume que se conoce la parte teórica de cada algoritmo de Machine Learning y cómo usarlos, no se explicará la base ni la parametrización de cada algoritmo.

REGRESIÓN LOGÍSTICA

La Regresión Logística es un método estadístico para predecir clases binarias. El resultado o variable objetivo es de naturaleza dicotómica. Dicotómica significa que solo hay dos clases posibles. Por ejemplo, se puede utilizar para problemas de detección de cáncer o calcular la probabilidad de que ocurra un evento.

La Regresión Logística es uno de los algoritmos de Machine Learning más simples y más utilizados para la clasificación de dos clases. Es fácil de implementar y se puede usar como línea de base para cualquier problema de clasificación binaria. La Regresión Logística describe y estima la relación entre una variable binaria dependiente y las variables independientes.

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

data = pd.read_csv('diabetes.csv')
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values
sc = StandardScaler()
X = sc.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
modelo = LogisticRegression()
modelo.fit(X_train, y_train)
y_pred = modelo.predict(X_test)

print('Datos de entrenamiento:')
print(y_test)
print('Datos de obtenidos en la predicción:')
print(y_pred)
```

Resultado una vez ejecutado el código anterior:

Para implementar este algoritmo solamente se debe definirlo, realizar su respectivo entrenamiento junto a los datos de entrenamiento y se realiza una predicción.



Para mayor información se puede revisar la siguiente información:

<http://bit.ly/2LotXZX>

K - VECINOS MÁS CERCANOS AQUÍ ✓

K vecinos más cercanos es un algoritmo de aprendizaje no paramétrico, esto significa que no hace suposiciones para la distribución de datos subyacentes. En otras palabras, la estructura del modelo es determinada a partir del conjunto de datos. Esto es muy útil en la práctica donde la mayoría de los conjuntos de datos del mundo real no siguen suposiciones teóricas matemáticas.

El algoritmo necesita todos los datos de entrenamiento y son utilizados en la fase de prueba. Esto hace que la capacitación sea más rápida y la fase de prueba más lenta y costosa. Lo costoso se refiere a que se requiere tiempo y memoria. En el peor de los casos, K vecinos más cercanos necesita más tiempo para escanear todos los puntos de datos y escanear todos los puntos de datos requerirá más memoria para almacenar datos de entrenamiento.

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

data = pd.read_csv('diabetes.csv')
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values
sc = StandardScaler()
X = sc.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
modelo = KNeighborsClassifier()
modelo.fit(X_train, y_train)
y_pred = modelo.predict(X_test)

print('Datos de entrenamiento:')
print(y_test)
print('Datos de obtenidos en la predicción:')
print(y_pred)
```

Resultado una vez ejecutado el código anterior:

```

Datos de entrenamiento:
[1 0 0 1 0 0 1 1 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
 0 0 0 0 0 1 1 0 0 1 1 1 0 0 1 0 0 0 0 1 1 1 1 0 0 1 1 1 1 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 1 0 0 0 1 0
 1 1 1 1 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 0
 0 1 0 1 0 0]

Datos de obtenidos en la predicción:
[1 0 0 1 0 0 1 1 1 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0
 0 0 1 0 0 0 1 1 0 0 0 1 0 0 0 1 1 0 0 0 1 0 0 1 0 0 1 1 1 1 0 1 0 0 0 0 1
 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0
 0 1 1 1 1 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0
 0 0 0 0 0]

```

Como se puede observar acá simplemente se debe definir el algoritmo, entrenar el modelo con los datos de entrenamiento y realizar una predicción.



Para mayor información se puede revisar la siguiente información:

<http://bit.ly/2SswHqL>
<http://bit.ly/2JTXZDR>

MÁQUINAS DE VECTORES DE SOPORTE

Las máquinas de vectores de soporte buscan la línea que mejor separa dos clases. Las características de datos que están más cerca de la línea que mejor separa las clases se denominan vectores de soporte e influyen en la ubicación de la línea. De particular importancia es el uso de diferentes funciones del kernel a través del parámetro del kernel.

```

import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

data = pd.read_csv('diabetes.csv')
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values
sc = StandardScaler()
X = sc.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
modelo = SVC()
modelo.fit(X_train, y_train)
y_pred = modelo.predict(X_test)

print('Datos de entrenamiento:')
print(y_test)
print('Datos de obtenidos en la predicción:')
print(y_pred)

```

Resultado una vez ejecutado el código anterior:

```

Datos de entrenamiento:
[1 0 0 1 0 0 1 1 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
 0 0 0 0 0 1 1 0 0 1 1 1 0 0 1 0 0 0 0 1 1 1 1 0 0 1 1 1 1 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 1 0 0 0 1 0 0
 1 1 1 1 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 0 1 0
 0 1 0 1 0 0]

Datos de obtenidos en la predicción:
[1 0 0 1 0 0 1 1 1 0 1 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0
 0 0 1 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 1
 1 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 1 0 0 0 0 1 0 0
 0 1 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0]

```

Para este algoritmo, de igual forma, se debe definir el modelo, entrenarlo y realizar una predicción.



Para mayor información se puede revisar la siguiente información:

<http://bit.ly/2PoF8LY>
<http://bit.ly/2Ufeaz9>

BAYESIANO INGENUO (NAIVE BAYES)

Naive Bayes es una técnica de clasificación estadística basada en el teorema de Bayes. Es uno de los algoritmos de Machine Learning más simple. Es un algoritmo rápido, preciso y fiable, inclusive tiene una alta precisión y velocidad en grandes conjuntos de datos.

```

import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB

data = pd.read_csv('diabetes.csv')
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values
sc = StandardScaler()
X = sc.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
modelo = GaussianNB()
modelo.fit(X_train, y_train)
y_pred = modelo.predict(X_test)

print('Datos de entrenamiento:')
print(y_test)
print('Datos de obtenidos en la predicción:')
print(y_pred)

```

Resultado una vez ejecutado el código anterior:

```

Datos de entrenamiento:
[1 0 0 1 0 0 1 1 0 0 1 1 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
 0 0 0 0 0 0 1 1 0 0 1 1 1 0 0 1 0 0 0 0 1 1 1 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0
 1 1 1 1 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 1 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0
 0 1 0 1 0 0]

Datos de obtenidos en la predicción:
[1 0 0 1 0 0 1 1 1 0 1 1 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1
 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0
 0 1 1 1 1 0 1 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0]

```

Igual que los casos anteriores, para este algoritmo se define el algoritmo a implementar, seguidamente se entrena utilizando los datos de entrenamiento y finalmente se realiza una predicción, bien sea con los datos de prueba o con nueva data.



Para mayor información se puede revisar la siguiente información:

<http://bit.ly/2kpp6ws>
<http://bit.ly/2lAagUr>

ÁRBOLES DE DECISIÓN CLASIFICACIÓN

Árbol de decisión es un tipo de algoritmo de aprendizaje supervisado que se utiliza principalmente en problemas de clasificación, aunque funciona para variables de entrada y salida categóricas como continuas.

En esta técnica, se divide la data en dos o más conjuntos homogéneos basados en el diferenciador más significativos en las variables de entrada. El árbol de decisión identifica la variable más significativa y su valor que proporciona los mejores conjuntos homogéneos de población. Todas las variables de entrada y todos los puntos de división posibles se evalúan y se elige la que tenga mejor resultado.

Los algoritmos de aprendizaje basados en árbol se consideran uno de los mejores y más utilizados métodos de aprendizaje supervisado. Los métodos basados en árboles potencian modelos predictivos con alta precisión, estabilidad y facilidad de interpretación. A diferencia de los modelos lineales, mapean bastante bien las relaciones no lineales.

```

import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

data = pd.read_csv('diabetes.csv')
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values
sc = StandardScaler()
X = sc.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
modelo = DecisionTreeClassifier()
modelo.fit(X_train, y_train)
y_pred = modelo.predict(X_test)

print('Datos de entrenamiento:')
print(y_test)
print('Datos de obtenidos en la predicción:')
print(y_pred)

```

Resultado una vez ejecutado el código anterior:

```

Datos de entrenamiento:
[1 0 0 1 0 0 1 1 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
 0 0 0 0 0 1 1 0 0 1 1 1 0 0 1 0 0 0 0 1 1 1 1 0 0 1 1 1 1 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 1 0 0 0 1 0
 1 1 1 1 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 1 1 0 0 0 0 0 1 0 0 0
 0 1 0 1 0 0]
Datos de obtenidos en la predicción:
[1 0 0 1 0 0 1 1 0 1 1 1 0 0 0 1 1 0 0 0 1 0 0 1 0 1 0 0 0 0 0 0 1 0 0 1 0
 0 1 1 1 0 0 1 1 0 0 0 0 0 1 1 1 1 0 0 0 1 0 0 1 0 0 1 1 1 0 0 1 0 0 0 0 1
 1 1 1 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 1 0
 1 0 1 1 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 1 1 0 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0]

```

Para implementar este algoritmo solamente se debe definirlo, realizar su respectivo entrenamiento junto a los datos de entrenamiento y realizamos una predicción.



Para mayor información se puede revisar la siguiente información:

<http://bit.ly/35McqTm>
<http://bit.ly/2p3vSuo>

BOSQUES ALEATORIOS CLASIFICACIÓN

Bosques aleatorios clasificación es un método versátil de Machine Learning. Lleva a cabo métodos de reducción dimensional, trata valores perdidos, valores atípicos y otros pasos esenciales de exploración de datos, y hace un trabajo bastante bueno. Es un tipo de método de aprendizaje por conjuntos, donde un grupo de modelos débiles se combinan para formar un modelo poderoso.

En los bosques aleatorios se cultivan varios árboles en lugar de un solo árbol. Para clasificar un nuevo objeto basado en atributos, cada árbol da una clasificación y se dice que el árbol “vota” por esa clase. El bosque elige la clasificación con más votos, sobre todos los árboles en el bosque.

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

data = pd.read_csv('diabetes.csv')
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values
sc = StandardScaler()
X = sc.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
modelo = RandomForestClassifier()
modelo.fit(X_train, y_train)
y_pred = modelo.predict(X_test)

print('Datos de entrenamiento:')
print(y_test)
print('Datos de obtenidos en la predicción:')
print(y_pred)
```

Resultado una vez ejecutado el código anterior:

```
Datos de entrenamiento:
[1 0 0 1 0 0 1 1 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
 0 0 0 0 0 1 1 0 0 1 1 1 0 0 1 0 0 0 0 1 1 1 1 0 0 1 1 1 1 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0
 1 1 1 1 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 1 1 0 0 0 0 0 1 0 0 0 1 0
 0 1 0 1 0 0]

Datos de obtenidos en la predicción:
[1 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1
 0 0 1 0 0 0 1 1 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0 1
 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
 0 1 0 1 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0
 0 0 0 0 0 0]
```

Este algoritmo no es distinto a los anteriores, acá se define el algoritmo, se entrena el modelo y se realiza la predicción respectiva.



Para mayor información se puede revisar la siguiente información:

<http://bit.ly/37bxAuD>
<http://bit.ly/2smgZ7F>

Todos los algoritmos se pueden configurar definiendo ciertos parámetros para mejorar el rendimiento del modelo. Todos estos parámetros se encuentran definidos en la librería scikit-learn.

Capítulo 10

MÉTRICAS DE RENDIMIENTO

ALGORITMOS DE

CLASIFICACIÓN

Para los problemas de clasificación de Machine Learning se cuentan con una gran cantidad de métricas que se pueden usar para evaluar las predicciones de estos problemas. En esta sección se explicará cómo implementar varias de estas métricas.

En este capítulo aprenderás:

1. Matriz de confusión
2. Reporte de clasificación
3. Área bajo la curva

MATRIZ DE CONFUSIÓN

La matriz de confusión es una de las métricas más intuitivas y sencillas que se utiliza para encontrar la precisión y exactitud del modelo. Se utiliza para el problema de clasificación donde la salida puede ser de dos o más tipos de clases.

Identificado el problema la matriz de confusión, es una tabla con dos dimensiones, “actual” y “predicción” y conjuntos de clases en ambas dimensiones. Las filas de la matriz indican la clase observada o real y las columnas indican las clases predicha.

Se debe aclarar que la matriz de confusión en sí misma no es una medida de rendimiento como tal, pero casi todas las métricas de rendimiento se basan en ella y en los números que contiene.

A continuación, se muestra un ejemplo del cálculo de una matriz de confusión para el conjunto de datos correspondiente a la diabetes de los indios Pima utilizando como algoritmo Regresión Logística.

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix

data = pd.read_csv('diabetes.csv')
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values
sc = StandardScaler()
X = sc.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
modelo = LogisticRegression()
modelo.fit(X_train, y_train)
y_pred = modelo.predict(X_test)
matriz = confusion_matrix(y_test, y_pred)
print(matriz)
```

Resultado una vez ejecutado el código anterior:

```
[[ 98  9]
 [18  29]]
```

La matriz de confusión nos arroja como resultado que la mayoría de las predicciones caen en la línea diagonal principal de la matriz, siendo estas predicciones correctas.



Para mayor información se puede revisar la siguiente información:

<http://bit.ly/2LQkSet>

REPORTE DE CLASIFICACIÓN

La librería scikit-learn proporciona un informe muy conveniente cuando se trabaja en problemas de clasificación, este da una idea rápida de la precisión de un modelo utilizando una serie de medidas. La función `classification_report()` muestra la precisión, la sensibilidad, la puntuación F1 y el soporte para cada clase.

En el siguiente ejemplo se muestra la implementación de la función en un problema.

```

import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

data = pd.read_csv('diabetes.csv')
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values
sc = StandardScaler()
X = sc.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
modelo = LogisticRegression()
modelo.fit(X_train, y_train)
y_pred = modelo.predict(X_test)
reporte = classification_report(y_test, y_pred)
print(reporte)

```

Resultado una vez ejecutado el código anterior:

	precision	recall	f1-score	support
0	0.84	0.92	0.88	107
1	0.76	0.62	0.68	47
micro avg	0.82	0.82	0.82	154
macro avg	0.80	0.77	0.78	154
weighted avg	0.82	0.82	0.82	154



Para mayor información se puede revisar la siguiente información:

<http://bit.ly/2MfsS8J>
<http://bit.ly/2ETCL6o>

ÁREA BAJO LA CURVA

Cuando se trata de un problema de clasificación, se puede contar con una curva AUC-ROC, para medir el rendimiento. Esta es una de las métricas de evaluación más importante para verificar el rendimiento de cualquier modelo de clasificación. ROC viene de las características de funcionamiento del receptor y AUC del área bajo la curva.

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score

data = pd.read_csv('diabetes.csv')
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values
sc = StandardScaler()
X = sc.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
modelo = LogisticRegression()
modelo.fit(X_train, y_train)
y_pred = modelo.predict(X_test)
resultado = roc_auc_score(y_test, y_pred)
print(resultado)
```

Resultado una vez ejecutado el código anterior:

0.7664545635315172

El valor obtenido es relativamente cercano a 1 y mayor a 0.5, lo que sugiere que gran parte de las predicciones realizadas han sido correctas.



Para mayor información se puede revisar la siguiente información:

<http://bit.ly/2MfsS8J>
<http://bit.ly/2ETCL6O>

Capítulo 11

ALGORITMOS DE REGRESIÓN

La comprobación de puntos es una forma de descubrir qué algoritmos funcionan bien en los problemas de Machine Learning. Es difícil saber con anterioridad qué algoritmo es el más adecuado a utilizar, por lo que se debe comprobar varios métodos y centrar la atención en aquellos que demuestran ser los más prometedores. En este capítulo se explicará cómo implementar los algoritmos de regresión utilizando Python y scikit-learn a los problemas de regresión.

En este capítulo aprenderás cómo implementar los siguientes algoritmos:

1. Regresión lineal
2. Regresión polinomial
3. Vectores de soporte regresión
4. Árboles de decisión regresión
5. Bosques aleatorios regresión

Para este análisis se tomará el conjunto de datos de publicidad que puedes encontrar en la página de Kaggle como “Advertising Data”. De igual forma, se asume que se conoce la parte teórica de cada algoritmo de Machine Learning y cómo usarlos, no se explicará la base ni la parametrización de cada algoritmo.

REGRESIÓN LINEAL

La regresión lineal es una técnica paramétrica utilizada para predecir variables continuas, dependientes, dado un conjunto de variables independientes. Es de

naturaleza paramétrica porque hace ciertas suposiciones basadas en el conjunto de datos. Si el conjunto de datos sigue esas suposiciones, la regresión arroja resultados increíbles, de lo contrario, tiene dificultades para proporcionar una precisión convincente.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

data = pd.read_csv('Advertising.csv')
X = data.iloc[:, 1:-1].values
y = data.iloc[:, -1].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
modelo = LinearRegression()
modelo.fit(X_train, y_train)
y_pred = modelo.predict(X_test)

print('Datos de entrenamiento:')
print(y_test)
print('Datos obtenidos en la predicción:')
print(y_pred)
```

Resultado una vez ejecutado el código anterior:

```
Datos de entrenamiento:
[11.3  8.4  8.7 25.4 11.7  8.7  7.2 13.2  9.2 16.6 24.2 10.6 10.5 15.6
 11.8 13.2 17.4  1.6 14.7 17.  26.2 10.3 14.9 12.9  8.1 15.2 12.6 22.6
 11.6  8.5 12.5 23.7 16.1 21.8  5.6  6.7  9.7 12.9 13.6  7.2]
Datos obtenidos en la predicción:
[10.05739563  7.4522807   7.0197076   24.08029725 12.01786259  6.53793858
 12.78286918 15.10974587 10.76974013 16.34357951 22.88297477  9.12924467
 10.46455672 15.48743552 11.58555633 12.17296914 18.76551502 10.78318566
 15.90515992 17.30651279 24.06692057  9.59834224 15.13512211 12.38591525
 5.71360885 15.24749314 12.29402334 20.9421167 13.40991558  9.04348832
 12.89239415 21.40272028 18.13802209 21.17320803  6.56974433  6.14114206
 7.89018394 13.01541434 14.68953791  6.18835143]
```

Como se puede observar acá simplemente se debe definir el algoritmo, entrenar el modelo con los datos de entrenamiento y realizar una predicción.



Para mayor información se puede revisar la siguiente información:

<http://bit.ly/2RpwDGK>
<http://bit.ly/2Su2aHM>

REGRESIÓN POLINOMIAL

La regresión polinomial es un caso especial de la regresión lineal, extiende el modelo lineal al agregar predictores adicionales, obtenidos al elevar cada uno de los predictores originales a una potencia. El método estándar para extender la regresión lineal a una

relación no lineal entre las variables dependientes e independientes, ha sido reemplazar el modelo lineal con una función polinomial.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

data = pd.read_csv('Advertising.csv')
X = data.iloc[:, 1:-1].values
y = data.iloc[:, -1].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
poli_reg = PolynomialFeatures(degree = 2)
X_train_poli = poli_reg.fit_transform(X_train)
X_test_poli = poli_reg.fit_transform(X_test)
modelo = LinearRegression()
modelo.fit(X_train_poli, y_train)
y_pred = modelo.predict(X_test_poli)

print('Datos de entrenamiento:')
print(y_test)
print('Datos de obtenidos en la predicción:')
print(y_pred)
```

Resultado una vez ejecutado el código anterior:

```
Datos de entrenamiento:
[11.3  8.4  8.7 25.4 11.7  8.7  7.2 13.2  9.2 16.6 24.2 10.6 10.5 15.6
 11.8 13.2 17.4  1.6 14.7 17.  26.2 10.3 14.9 12.9  8.1 15.2 12.6 22.6
 11.6  8.5 12.5 23.7 16.1 21.8  5.6  6.7  9.7 12.9 13.6  7.2]
Datos de obtenidos en la predicción:
[10.32365121  8.39841405  8.89627053 25.24070266 12.30359757  8.20929435
 8.39471096 13.28283504  8.77249196 16.62804746 24.34984266 10.21435934
 10.97354763 15.7894985 11.47758796 12.81754966 17.46393383  6.76725579
 14.25447966 17.47655436 25.41227954 10.48751878 15.61851526 13.11794443
 7.73830842 15.51366219 12.65136685 22.50665202 11.798091  8.06138846
 12.84208623 23.23644925 15.47279687 22.20547191  6.74904844  6.72500414
 9.02164368 13.25649519 13.08533471  7.05095332]
```

Para este algoritmo se debe realizar un paso adicional, antes de definir el modelo, primeramente, se debe definir el grado del polinomio y transformar los datos correspondientes a X. Realizado todo esto ahora si se puede definir el modelo, entrenarlos y finalmente realizar una predicción.



Para mayor información se puede revisar la siguiente información:

<http://bit.ly/2H889RX>

VECTORES DE SOPORTE REGRESIÓN

Los vectores de soporte regresión utiliza los mismos principios que los de clasificación, con solo algunas diferencias menores. En primer lugar, dado que la salida es un número

real, se vuelve muy difícil predecir la información disponible, que tiene infinitas posibilidades, sin embargo, la idea principal es siempre la misma: minimizar el error, individualizar el hiperplano que maximiza el margen, teniendo en cuenta que se tolera parte del error.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler

data = pd.read_csv('Advertising.csv')
X = data.iloc[:, 1:-1].values
sc = StandardScaler()
X = sc.fit_transform(X)
y = data.iloc[:, -1].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
modelo = SVR()
modelo.fit(X_train, y_train)
y_pred = modelo.predict(X_test)

print('Datos de entrenamiento:')
print(y_test)
print('Datos obtenidos en la predicción:')
print(y_pred)
```

Resultado una vez ejecutado el código anterior:

```
Datos de entrenamiento:
[11.3  8.4  8.7 25.4 11.7  8.7  7.2 13.2  9.2 16.6 24.2 10.6 10.5 15.6
 11.8 13.2 17.4  1.6 14.7 17.  26.2 10.3 14.9 12.9  8.1 15.2 12.6 22.6
 11.6  8.5 12.5 23.7 16.1 21.8  5.6  6.7  9.7 12.9 13.6  7.2]
Datos de obtenidos en la predicción:
[10.30751224  8.60871999  9.15994697 22.16004249 11.98364262  8.75084153
 11.6907236 13.31795341 10.65516933 16.78917175 22.44298838 10.10721087
 10.95944249 15.86875751 11.36392211 12.7702331 17.15949279 9.58380712
 14.43333411 17.67239621 21.14095959 10.5307976 15.47183109 12.90006996
 8.27894123 15.4079747 12.27575281 21.32402889 11.8332806 9.29131224
 13.62179758 21.30402924 15.40692401 21.67164179 9.45465365 7.83538382
 9.98145734 13.20177095 13.23163987 8.17688616]
```

Para este algoritmo, de igual forma, se debe definir el modelo, entrenarlo y realizar una predicción. Para este conjunto de datos se realiza, previamente, un escalamiento en los datos ya que el modelo no era del todo efectivo con los datos originales.



Para mayor información se puede revisar la siguiente información:

<http://bit.ly/2CY3xbH>
<http://bit.ly/2TEDkG2>

ÁRBOLES DE DECISIÓN REGRESIÓN

Los árboles de decisión son una técnica de aprendizaje supervisado que predice valores de respuestas mediante el aprendizaje de reglas de decisión derivadas de características.

Los árboles de decisión funcionan al dividir el espacio de la característica en varias regiones rectangulares simples, divididas por divisiones paralelas de ejes. Para obtener una predicción para una observación particular, se utiliza la media o el modo de las respuestas de las observaciones de entrenamiento, dentro de la partición a la que pertenece la nueva observación.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor

data = pd.read_csv('Advertising.csv')
X = data.iloc[:, 1:-1].values
y = data.iloc[:, -1].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
modelo = DecisionTreeRegressor()
modelo.fit(X_train, y_train)
y_pred = modelo.predict(X_test)

print('Datos de entrenamiento:')
print(y_test)
print('Datos obtenidos en la predicción:')
print(y_pred)
```

Resultado una vez ejecutado el código anterior:

```
Datos de entrenamiento:
[11.3  8.4  8.7 25.4 11.7  8.7  7.2 13.2  9.2 16.6 24.2 10.6 10.5 15.6
 11.8 13.2 17.4  1.6 14.7 17.  26.2 10.3 14.9 12.9  8.1 15.2 12.6 22.6
 11.6  8.5 12.5 23.7 16.1 21.8  5.6  6.7  9.7 12.9 13.6  7.2]
Datos de obtenidos en la predicción:
[10.9  9.7  8.8 25.5 12.9  8.6  6.6 12.5  8.  16.6 25.5  9.9 10.6 15.5
 10.9 11.9 15.9  7.3 12.6 17.4 25.4 10.3 16.6 12.2  8.6 14.2 14.4 20.7
 10.6  6.6 12.6 19.2 15.9 22.3  5.5  7.  9.7 15.  11.6  6.6]
```

Igual que los casos anteriores, para este algoritmo se define el algoritmo a implementar, seguidamente se entrena utilizando los datos de entrenamiento y finalmente se realiza una predicción, bien sea con los datos de prueba o con nueva data.



Para mayor información se puede revisar la siguiente información:

<http://bit.ly/2tHfP3U>
<http://bit.ly/2tStiGi>

BOSQUES ALEATORIOS REGRESIÓN

Los bosques aleatorios es un algoritmo de aprendizaje supervisado que, como ya se puede ver en su nombre, crea un bosque y lo hace de alguna manera aleatorio. Para decirlo en palabras simples: el bosque aleatorio crea múltiples árboles de decisión y los combina para obtener una predicción más precisa y estable. En general, mientras más árboles en el bosque se vea, más robusto es el bosque.

En este algoritmo se agrega aleatoriedad adicional al modelo, mientras crece los árboles, en lugar de buscar la característica más importante al dividir un nodo, busca la mejor característica entre un subconjunto aleatorio de características. Esto da como resultado una amplia diversidad que generalmente resulta en un mejor modelo.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor

data = pd.read_csv('Advertising.csv')
X = data.iloc[:, 1:-1].values
y = data.iloc[:, -1].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
modelo = RandomForestRegressor()
modelo.fit(X_train, y_train)
y_pred = modelo.predict(X_test)

print('Datos de entrenamiento:')
print(y_test)
print('Datos de obtenidos en la predicción:')
print(y_pred)
```

Resultado una vez ejecutado el código anterior:

```
Datos de entrenamiento:
[11.3  8.4  8.7 25.4 11.7  8.7  7.2 13.2  9.2 16.6 24.2 10.6 10.5 15.6
 11.8 13.2 17.4  1.6 14.7 17.  26.2 10.3 14.9 12.9  8.1 15.2 12.6 22.6
 11.6  8.5 12.5 23.7 16.1 21.8  5.6  6.7  9.7 12.9 13.6  7.2]
Datos de predichos:
[10.32365121  8.39841405  8.89627053 25.24070266 12.30359757  8.20929435
  8.39471096 13.28283504  8.77249196 16.62804746 24.34984266 10.21435934
 10.97354763 15.7894985 11.47758796 12.81754966 17.46393383  6.76725579
 14.25447966 17.47655436 25.41227954 10.48751878 15.61851526 13.11794443
  7.73830842 15.51366219 12.65136685 22.50665202 11.798091  8.06138846
 12.84208623 23.23644925 15.47279687 22.20547191  6.74904844  6.72500414
  9.02164368 13.25649519 13.08533471  7.05095332]
```

Este algoritmo no es distinto a los anteriores, acá se define el algoritmo, se entrena el modelo y se realiza la predicción respectiva.



Para mayor información se puede revisar la siguiente información:

<http://bit.ly/2FTuYoA>

<http://bit.ly/2PhGwpT>

Recuerda que este algoritmo, al igual que los anteriores, se puede configurar definiendo ciertos parámetros para mejorar el rendimiento del modelo. Todos estos parámetros se encuentran definidos en la librería scikit-learn.

Capítulo 12

MÉTRICAS DE RENDIMIENTO ALGORITMOS DE REGRESIÓN

En este capítulo se revisará las métricas más comunes para evaluar las predicciones sobre problemas de Machine Learning de regresión.

En este capítulo aprenderás:

1. Error cuadrático medio
2. Error absoluto medio
3. R^2

ERROR CUADRÁTICO MEDIO (RMSE)

La métrica más comúnmente utilizada para las tareas de regresión es el error cuadrático medio y representa a la raíz cuadrada de la distancia cuadrada promedio entre el valor real y el valor pronosticado.

Indica el ajuste absoluto del modelo a los datos, cuán cerca están los puntos de datos observados de los valores predichos del modelo. El error cuadrático medio o RMSE es una medida absoluta de ajuste.

```

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from math import sqrt

data = pd.read_csv('Advertising.csv')
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
modelo = LinearRegression()
modelo.fit(X_train, y_train)
y_pred = modelo.predict(X_test)

error = sqrt(mean_squared_error(y_test, y_pred))
print(error)

```

Resultado una vez ejecutado el código anterior:

2.09812256349568

El mejor valor para este parámetro es de 0.0.



Para mayor información se puede revisar la siguiente información:

<http://bit.ly/2TzzKNY>

ERROR ABSOLUTO MEDIO (MAE)

El error absoluto medio es el promedio de la diferencia absoluta entre los valores predichos y el valor observado. Es un puntaje lineal, lo que significa que todas las diferencias individuales se ponderan por igual en el promedio.

```

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

data = pd.read_csv('Advertising.csv')
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
modelo = LinearRegression()
modelo.fit(X_train, y_train)
y_pred = modelo.predict(X_test)

error = mean_squared_error(y_test, y_pred)
print(error)

```

Resultado una vez ejecutado el código anterior:

4.402118291449684

Para este parámetro un valor de 0.0 indica que no hay error, es decir las predicciones son perfectas.



Para mayor información se puede revisar la siguiente información:

<http://bit.ly/2TzzKNY>

R^2

R^2 indica la bondad o la aptitud del modelo, a menudo se utiliza con fines descriptivos y muestra que también las variables independientes seleccionadas explican la variabilidad en sus variables dependiente.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

data = pd.read_csv('Advertising.csv')
X = data.iloc[:, 1:-1].values
y = data.iloc[:, -1].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
modelo = LinearRegression()
modelo.fit(X_train, y_train)
y_pred = modelo.predict(X_test)

error = r2_score(y_test, y_pred)
print(error)
```

Resultado una vez ejecutado el código anterior:

0.8601145185017869

La mejor puntuación posible es 1.0 y puede ser negativa, porque el modelo puede ser arbitrariamente peor. Un modelo constante que siempre predice el valor esperado de "y", sin tener en cuenta las características de entrada, obtendrá una puntuación de 0.0.



Para mayor información se puede revisar la siguiente información:

<http://bit.ly/2TzzKNY>

Capítulo 13

PROYECTO DE MACHINE LEARNING - CLASIFICACIÓN

En este capítulo se desarrollará un proyecto de clasificación utilizando Python, se incluye cada paso del proceso de Machine Learning aplicado al problema.

En este capítulo aprenderás:

1. Cómo trabajar a través de un problema de clasificación
2. Cómo utilizar transformaciones de datos para mejorar el rendimiento del modelo
3. Cómo implementar algoritmos de Machine Learning y comparar sus resultados

DEFINICIÓN DEL PROBLEMA

Para el proyecto se utilizará el mismo conjunto de datos que se ha venido trabajando durante cada uno de los capítulos, pero esta vez se desarrollará de manera completa. El conjunto de datos es el correspondiente a los indios Pima.

Este conjunto describe los registros médicos de los indios Pima y si cada paciente tendrá un inicio de diabetes dentro de los cinco años. Como tal es un problema de clasificación.

El conjunto de datos se puede encontrar en la página de Kaggle como “Pima Indians Diabetes Database”.

IMPORTAR LAS LIBRERÍAS

El primer paso en cualquier proyecto de Machine Learning es el de importar las librerías a utilizar. Es normal que al principio no se conozca todas las librerías a utilizar y que se vayan agregando poco a poco, pero por lo menos se debe comenzar con las básicas que normalmente se utilizan al implementar cualquier proyecto de Machine Learning.

Importar las librerías se puede realizar a medida que se vaya programando, lo que recomiendo es que se coloquen todas estas líneas de códigos al principio del programa para que sea fácil conocer los módulos que se están utilizando dentro del programa, así como también mantener un orden en la programación.

Por los momentos se importarán las librerías de Pandas, NumPy y matplotlib.

```
#### IMPORTAR LAS LIBRERÍAS ####
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

CARGAR EL CONJUNTO DE DATOS

Una vez que se hayan importado las librerías ya se pueden utilizar dentro del programa, para esto se procede a cargar el conjunto de datos.

El conjunto de datos se debe descargar directamente de la página de Kaggle y guardarlo en el computador en una carpeta específica en donde estará el archivo del conjunto de datos junto con el archivo del programa de Python que se está desarrollando, de esta forma se hará más fácil la programación.

Recuerda que para descargar los archivos de Kaggle deberás estar suscrito en la página, esto es completamente gratis y tendrás a tu disposición un gran número de conjuntos de datos con los que podrás practicar más adelante.

```
#### CARGAR LOS DATOS ####
data = pd.read_csv('diabetes.csv')
```

Se debe colocar el nombre exacto del archivo en donde se encuentra el conjunto de datos. Se recomienda guardar los datos en la misma carpeta en donde se encuentra el archivo del programa de esta forma se mantiene un orden.

ENTENDIENDO LOS DATOS

El siguiente paso en el desarrollo de un proyecto de Machine Learning es el de entender los datos con lo se cuenta. Para este paso se utilizan varias funciones que se tiene disponible en la librería de Pandas.

```
#### ENTENDIENDO LOS DATOS ####  
#Ver las primeras 5 filas  
data.head()
```

Resultado una vez ejecutado el código anterior:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	
0	6	148	72	35	0	33.6		0.627	50	1
1	1	85	66	29	0	26.6		0.351	31	0
2	8	183	64	0	0	23.3		0.672	32	1
3	1	89	66	23	94	28.1		0.167	21	0
4	0	137	40	35	168	43.1		2.288	33	1

```
#Dimensión de los datos  
data.shape
```

Resultado una vez ejecutado el código anterior:

(768, 9)

```
#Tipos de datos  
data.dtypes
```

Resultado una vez ejecutado el código anterior:

```
Pregnancies          int64  
Glucose              int64  
BloodPressure        int64  
SkinThickness        int64  
Insulin              int64  
BMI                  float64  
DiabetesPedigreeFunction float64  
Age                  int64  
Outcome              int64  
dtype: object
```

```
#Descripción estadística  
data.describe()
```

Resultado una vez ejecutado el código anterior:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

```
#Distribución de clases
conteo_data = data.groupby('quality').size()
print(conteo_data)
```

Resultado una vez ejecutado el código anterior:

```
Outcome
0    500
1    268
dtype: int64
```

```
#Correlación entre características
correlacion = data.corr(method='pearson')
print(correlacion)
```

Resultado una vez ejecutado el código anterior:

Pregnancies	1.000000	0.129459	0.141282	-0.081672
Glucose	0.129459	1.000000	0.152590	0.057328
BloodPressure	0.141282	0.152590	1.000000	0.207371
SkinThickness	-0.081672	0.057328	0.207371	1.000000
Insulin	-0.073535	0.331357	0.088933	0.436783
BMI	0.017683	0.221071	0.281805	0.392573
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928
Age	0.544341	0.263514	0.239528	-0.113970
Outcome	0.221898	0.466581	0.065068	0.074752
	Insulin	BMI	DiabetesPedigreeFunction	\
Pregnancies	-0.073535	0.017683	-0.033523	
Glucose	0.331357	0.221071	0.137337	
BloodPressure	0.088933	0.281805	0.041265	
SkinThickness	0.436783	0.392573	0.183928	
Insulin	1.000000	0.197859	0.185071	
BMI	0.197859	1.000000	0.140647	
DiabetesPedigreeFunction	0.185071	0.140647	1.000000	
Age	-0.042163	0.036242	0.033561	
Outcome	0.130548	0.292695	0.173844	
	Age	Outcome		
Pregnancies	0.544341	0.221898		
Glucose	0.263514	0.466581		
BloodPressure	0.239528	0.065068		
SkinThickness	-0.113970	0.074752		
Insulin	-0.042163	0.130548		
BMI	0.036242	0.292695		
DiabetesPedigreeFunction	0.033561	0.173844		
Age	1.000000	0.238356		
Outcome	0.238356	1.000000		

El conjunto de datos cuenta con 9 columnas, de las cuales 8 son las variables independientes y la última columna corresponde a la variable dependiente. Adicionalmente todos los datos son numéricos, entre enteros y flotantes.

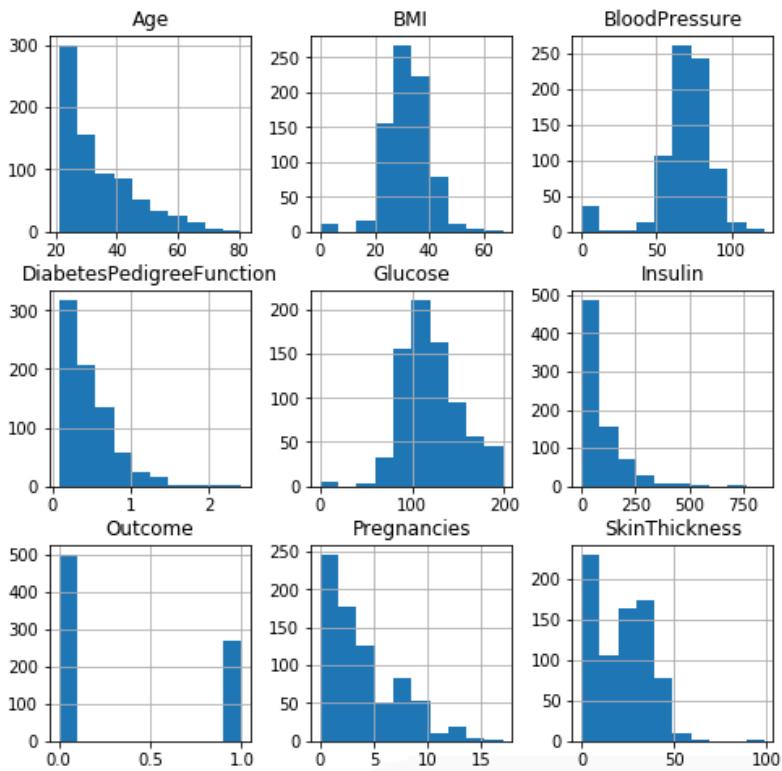
Adicionalmente, los datos se encuentran balanceados por lo que no es necesario realizar ningún paso adicional.

VISUALIZANDO LOS DATOS

Una vez que se haya entendido los datos de manera numérica ahora se analizará de manera visual utilizando la librería matplotlib. Esta librería fue importada al inicio del programa por lo que no es necesario hacerlo nuevamente.

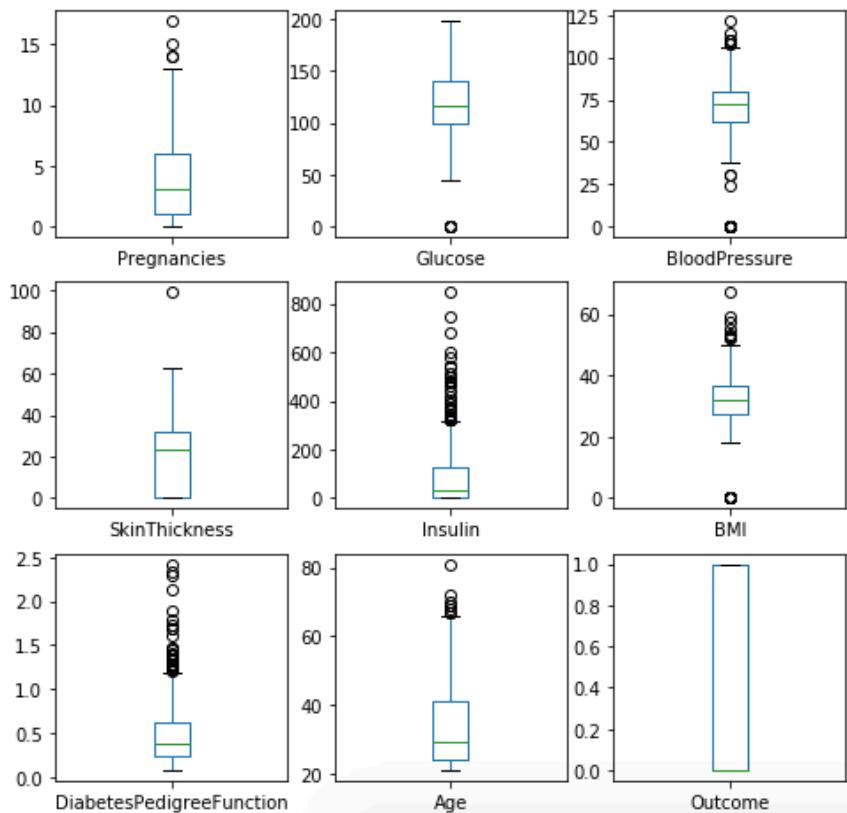
```
#### VISUALIZANDO LOS DATOS ####
#Histograma
data.hist(figsize = (8, 8))
plt.show()
```

Resultado una vez ejecutado el código anterior:



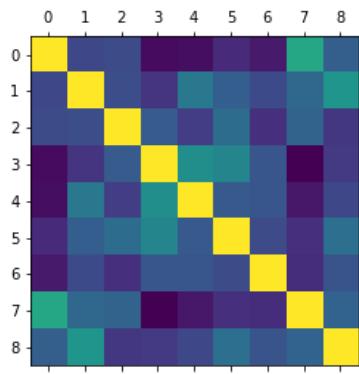
```
#Diagrama de Cajas  
data.plot(kind='box', subplots=True, layout=(4,4), sharex=False, sharey=False, figsize = (8, 8))  
plt.show()
```

Resultado una vez ejecutado el código anterior:



```
#Matriz de Correlación
plt.matshow(data.corr())
plt.show
```

Resultado una vez ejecutado el código anterior:



Como podemos observar en las gráficas obtenidas los resultados son muy similares a los obtenidos en el punto anterior.

SEPARACIÓN DE LOS DATOS

Conociendo el conjunto de datos que se está trabajando dentro del proyecto, se procede a realizar la separación de los datos en dos conjuntos, el primero de variables independientes, X , y un segundo conjunto de variable dependientes, y .

Se comienza separando los datos correspondientes a las variables independientes, que vendrían siendo todas las columnas menos la última.

```
### SEPARACIÓN DE LOS DATOS ###
#Datos de entrada o independientes
X = data.iloc[:, :-1].values
print(X)
```

Resultado una vez ejecutado el código anterior:

```
[[ 6.   148.    72.    ...  33.6    0.627  50.    ]
 [ 1.   85.    66.    ...  26.6    0.351   31.    ]
 [ 8.   183.   64.    ...  23.3    0.672   32.    ]
 ...
 [ 5.   121.   72.    ...  26.2    0.245   30.    ]
 [ 1.   126.   60.    ...  30.1    0.349   47.    ]
 [ 1.   93.    70.    ...  30.4    0.315   23.    ]]
```

Seguidamente se define los variables correspondientes a y , o variable dependiente, que vendría siendo la última columna del conjunto de datos.

```
#Datos de salida o dependientes
y_r = data.iloc[:, -1].values
print(y_r)
```

Resultado una vez ejecutado el código anterior:

```
[1 0 1 0 1 0 1 0 1 1 0 1 0 1 1 1 1 0 1 0 0 1 1 1 1 1 0 0 0 0 1 0 0 0 0 0
1 1 1 0 0 0 1 0 1 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 1 0 1 0 0 0 1 0 1 0
0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 1
1 0 0 1 1 1 0 0 0 0 1 0 0 0 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 1 0 1 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0
1 1 1 1 1 0 0 1 1 0 1 0 1 1 1 0 0 0 0 0 0 1 1 0 1 0 0 0 0 1 1 1 1 0 1 1 1
0 0 0 0 1 0 0 0 1 1 0 0 0 0 1 1 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0
1 0 1 0 0 1 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 1 1 0 0
1 0 1 0 1 1 0 0 0 1 0 1 0 0 1 0 1 0 0 1 0 1 1 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 0 1
0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 1 0 1 0 1 1 0 0 1 0 0 1 0 0 1
1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 0 0 1 0 0 1 0 0 1 0 1 1 0 1 0 1 0 1 0 1
0 1 1 0 0 0 0 1 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 1
1 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1
0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 1 1 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 1 0 1 0 1 0 1 0 1 0
1 0 0 1 0 0 1 0 0 0 0 0 1 1 0 1 0 0 0 0 0 1 1 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 1 1 1 0
1 1 0 0 0 0 0 0 0 1 1 0 1 0 0 1 0 0 0 0 0 0 1 0 1 0 1 0 1 0 1 1 0 0 0 0 0 0 0 0 0 1 1 1
0 0 0 1 0 1 1 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 0 0 1 0 0 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
```

SELECCIONANDO CARACTERÍSTICAS

Como se observo la variable X cuenta con 8 columnas, para este caso se aplicará el procedimiento para seleccionar las 5 características que tengan mayor influencia en la variable dependiente. Para este ejemplo se utilizará el método de filtro.

Para implementar las funciones respectivas se deben importar las librerías correspondientes. Se recomienda colocar estas líneas de código al principio del programa junto al resto de librerías con la que se esta trabajando. Esto hace que el programa sea mucho más limpio y fácil de entender para otras personas.

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
```

Realizado esto ya se puede implementar el código respectivo.

```
#### SELECCIONANDO CARACTERÍSTICAS ####
#Métodos de filtro
#Extracción de características: chi-cuadrado
prueba = SelectKBest(score_func=chi2, k=5)
entrenamiento = prueba.fit(X, y)

#Puntaje de características: chi-cuadrado
np.set_printoptions(precision=3)
print(entrenamiento.scores_)

#Características seleccionadas: chi-cuadrado
características = entrenamiento.transform(X)
print(características[0:5,:])
```

Resultado una vez ejecutado el código anterior:

```
[ 111.52 1411.887   17.605   53.108 2175.565   127.669    5.393  181.304]
[[ 6.  148.    0.   33.6  50. ]
 [ 1.   85.    0.   26.6  31. ]
 [ 8.  183.    0.   23.3  32. ]
 [ 1.   89.   94.   28.1  21. ]
 [ 0.  137.  168.   43.1  33. ]]
```

Las 5 características que tienen mayor impacto para la variable dependiente son las siguientes:

- Columna 0 – pregnancies
- Columna 1 – glucose
- Columna 4 – insulin
- Columna 5 – bmi
- Columna 7 – age

Por lo tanto, se convierte ahora las variables X con solo estas 5 columnas.

```
X = data.iloc[:, [0, 1, 4, 5, 7]].values
print(X)
```

Resultado una vez ejecutado el código anterior:

```
[[ 6.    148.     0.627   33.6    50.    ]
 [ 1.    85.      0.351   26.6    31.    ]
 [ 8.    183.     0.672   23.3    32.    ]
 ...
 [ 5.    121.     0.245   26.2    30.    ]
 [ 1.    126.     0.349   30.1    47.    ]
 [ 1.    93.      0.315   30.4    23.    ]]
```

PROCESAMIENTO DE LOS DATOS

Definido los valores de X que se utilizaran en los algoritmos, ahora se procede a realizar el respectivo procesamiento de los datos.

Para este caso solamente se estandarizarán los datos, ya que los mismos se encuentran en distintas escalas y esto puede ocasionar errores en el análisis.

Antes de aplicar la función se debe importar la respectiva librería. Recuerda colocar esta línea de código al principio del programa.

```
from sklearn.preprocessing import StandardScaler
```

Ahora se realiza la estandarización de los datos, es importante mencionar que este procedimiento solamente se realiza a los datos correspondientes a X solamente.

```
#### PREPROCESAMIENTO LOS DATOS ####
#Estandarización de los datos
sc = StandardScaler()
X = sc.fit_transform(X)
print(X)
```

Resultado una vez ejecutado el código anterior:

```
[[ 0.64   0.848   0.468   0.204   1.426]
 [-0.845 -1.123  -0.365  -0.684  -0.191]
 [ 1.234  1.944   0.604  -1.103  -0.106]
 ...
 [ 0.343  0.003  -0.685  -0.735  -0.276]
 [-0.845  0.16   -0.371  -0.24   1.171]
 [-0.845 -0.873  -0.474  -0.202  -0.871]]
```

SEPARACIÓN DE LOS DATOS

Se llega al último paso antes de implementar los algoritmos de Machine Learning, este es el correspondiente al de separar los datos en entrenamiento y prueba, para esto se utiliza la función `train_test_split` de la librería scikit-learn, recuerda que se debe importar antes de usarla.

```
from sklearn.model_selection import train_test_split
```

Se utilizará solamente un 25% del conjunto de datos como datos de prueba, por lo que se coloca en el tamaño de la prueba 0.25.

```
### SEPARACIÓN DE DATOS DE ENTRENAMIENTO Y PRUEBA ####  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

APLICACIÓN DE LOS ALGORITMOS DE CLASIFICACIÓN

Se implementarán los algoritmos explicados con anterioridad y se evaluará los resultados con respecto al error de cada uno de ellos.

Regresión Logística

El primer algoritmo a evaluar es el más básico de todos y es el de Regresión Logística.

```
### REGRESIÓN LOGÍSTICA ###
from sklearn.linear_model import LogisticRegression

modelo = LogisticRegression()
modelo.fit(X_train, y_train)
y_pred = modelo.predict(X_test)

print('Datos de entrenamiento:')
print(y_test)
print('Datos de obtenidos en la predicción:')
print(y_pred)
```

Resultado una vez ejecutado el código anterior:

Los anteriores datos es la representación de los datos de pruebas, originales, con los datos obtenidos por el modelo. A continuación, se realiza el análisis respectivo con las métricas correspondientes a los algoritmos de clasificación.

La primera métrica a evaluar será la de la matriz de confusión, para ello primero se importará la librería y posteriormente la función.

```
from sklearn.metrics import confusion_matrix
```

```
## Matriz de confusión
matriz = confusion_matrix(y_test, y_pred)
print(matriz)
```

Resultado una vez ejecutado el código anterior:

```
[[117  13]
 [ 29  33]]
```

Los resultados obtenidos acá son bastante satisfactorios, ya que el modelo pudo obtener unos buenos resultados.

Ahora se evalúa los resultados, pero esta vez utilizando el reporte de clasificación disponible en la librería scikit-learn.

```
from sklearn.metrics import classification_report
```

```
## Reporte de clasificación
reporte = classification_report(y_test, y_pred)
print(reporte)
```

Resultado una vez ejecutado el código anterior:

	precision	recall	f1-score	support
0	0.80	0.90	0.85	130
1	0.72	0.53	0.61	62
micro avg	0.78	0.78	0.78	192
macro avg	0.76	0.72	0.73	192
weighted avg	0.77	0.78	0.77	192

En este reporte se puede evidenciar que el porcentaje de precisión, sensibilidad y puntaje F1 del modelo rondan alrededor de 0.8, número que es muy bueno.

K Vecinos más Cercanos

El siguiente algoritmo a evaluar será el de K vecinos más cercanos, acá se realizará el mismo procedimiento que se realizó anteriormente.

```
### VECINOS MÁS CERCANOS ###
from sklearn.neighbors import KNeighborsClassifier

modelo = KNeighborsClassifier()
modelo.fit(X_train, y_train)
y_pred = modelo.predict(X_test)

print('Datos de entrenamiento:')
print(y_test)
print('Datos de obtenidos en la predicción:')
print(y_pred)
```

Resultado una vez ejecutado el código anterior:

```
Datos de entrenamiento:
[1 0 0 1 0 0 1 1 0 0 1 1 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
 0 0 0 0 0 1 1 0 0 1 1 0 0 1 0 0 0 0 1 1 1 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 1 0 0 0 1 0
 1 1 1 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 1 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0
 0 1 0 1 0 0 1 0 0 0 1 1 1 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 1 1
 0 1 1 1 0 0 0]

Datos de obtenidos en la predicción:
[1 0 0 1 0 0 1 1 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0 0 1 0 1 0
 0 0 1 1 0 0 1 0 0 0 1 0 1 0 1 0 0 0 1 0 1 1 0 0 1 1 1 0 0 0 0 0 0 1
 1 1 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 1 1 0 0 0 1 0
 0 0 1 1 1 0 1 0 0 0 0 0 0 0 1 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0
 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 1
 0 1 0 0 0 0]
```

Realizado todo este procedimiento se continúa evaluando el modelo, para ello se aplica las dos métricas que se utilizó con el anterior, primeramente, la matriz de confusión.

```
## Matriz de confusión
matriz = confusion_matrix(y_test, y_pred)
print(matriz)
```

Resultado una vez ejecutado el código anterior:

```
[ [109  21]
 [ 26  36]]
```

Y posteriormente se obtiene el reporte de clasificación.

```
## Reporte de clasificación
reporte = classification_report(y_test, y_pred)
print(reporte)
```

Resultado una vez ejecutado el código anterior:

	precision	recall	f1-score	support
0	0.81	0.84	0.82	130
1	0.63	0.58	0.61	62
micro avg	0.76	0.76	0.76	192
macro avg	0.72	0.71	0.71	192
weighted avg	0.75	0.76	0.75	192

Como se puede observar los resultados obtenidos acá son un poco mejores que los obtenidos con el anterior algoritmo.

Máquinas de Vectores de Soporte

Ahora se evaluará el algoritmo de máquinas de vectores de soporte, el procedimiento es muy similar a los otros algoritmos evaluados anteriormente.

```
### MÁQUINAS DE VECTORES DE SOPORTE ###
from sklearn.svm import SVC

modelo = SVC()
modelo.fit(X_train, y_train)
y_pred = modelo.predict(X_test)

print('Datos de entrenamiento:')
print(y_test)
print('Datos de obtenidos en la predicción:')
print(y_pred)
```

Resultado una vez ejecutado el código anterior:

```
Datos de entrenamiento:
[1 0 0 1 0 0 1 1 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
 0 0 0 0 0 1 1 0 0 1 1 0 0 1 0 0 0 0 1 1 1 1 0 0 1 1 1 1 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 1 0 0 0 1 0
 1 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 1 0 0 0
 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 1 1
 0 1 1 1 0 0 0]

Datos de obtenidos en la predicción:
[1 0 0 1 0 0 1 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0
 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 1 0 0 1 1 0 0 0 0 0 1
 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 1 0
 0 0 1 1 1 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0
 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0
 0 0 0 0 0 0]
```

Igual que en los casos anteriores, se evalúa el modelo obtenido por medio de la matriz de confusión y el reporte de clasificación.

```
## Matriz de confusión
matriz = confusion_matrix(y_test, y_pred)
print(matriz)
```

Resultado una vez ejecutado el código anterior:

```
[[116  14]
 [ 31  31]]
```

```
## Reporte de clasificación
reporte = classification_report(y_test, y_pred)
print(reporte)
```

Resultado una vez ejecutado el código anterior:

	precision	recall	f1-score	support
0	0.79	0.89	0.84	130
1	0.69	0.50	0.58	62
micro avg	0.77	0.77	0.77	192
macro avg	0.74	0.70	0.71	192
weighted avg	0.76	0.77	0.75	192

Naive Bayes

El siguiente algoritmo a evaluar será el de naive bayes o como se le conoce en español, el bayesiano ingenuo.

```
### NAIVE BAYES ###
from sklearn.naive_bayes import GaussianNB

modelo = GaussianNB()
modelo.fit(X_train, y_train)
y_pred = modelo.predict(X_test)

print('Datos de entrenamiento:')
print(y_test)
print('Datos de obtenidos en la predicción:')
print(y_pred)
```

Resultado una vez ejecutado el código anterior:

```
Datos de entrenamiento:
[1 0 0 1 0 0 1 1 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
 0 0 0 0 0 1 1 0 0 1 1 1 0 0 1 0 0 0 0 1 1 1 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0
 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 1 1 0 0 0 0 0 1 0 0 0 1 0 0 0
 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 1 1
 0 1 1 1 0 0 0]
Datos de obtenidos en la predicción:
[1 0 0 1 0 0 1 1 1 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 1
 0 0 1 0 0 0 1 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 1 1 1 0 0 0 0 0 0 1
 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 1 0
 0 1 1 1 1 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0
 0 0 0 1 0 0 1 0 1 0 0 1 1 1 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 1
 0 0 0 0 0 0 0]
```

Definido el modelo, se verifica el error del mismo, para ello se utiliza la función de la matriz de confusión y el reporte de clasificación.

```
## Matriz de confusión
matriz = confusion_matrix(y_test, y_pred)
print(matriz)
```

Resultado una vez ejecutado el código anterior:

```
[[111  19]
 [ 26  36]]
```

```
## Reporte de clasificación
reporte = classification_report(y_test, y_pred)
print(reporte)
```

Resultado una vez ejecutado el código anterior:

	precision	recall	f1-score	support
0	0.81	0.85	0.83	130
1	0.65	0.58	0.62	62
micro avg	0.77	0.77	0.77	192
macro avg	0.73	0.72	0.72	192
weighted avg	0.76	0.77	0.76	192

Árboles de Decisión Clasificación

Para evaluar el problema implementando el algoritmo de árboles de decisión clasificación se realiza el siguiente procedimiento.

```
### ÁRBOLES DE DECISIÓN CLASIFICACIÓN ###
from sklearn.tree import DecisionTreeClassifier

modelo = DecisionTreeClassifier()
modelo.fit(X_train, y_train)
y_pred = modelo.predict(X_test)

print('Datos de entrenamiento:')
print(y_test)
print('Datos de obtenidos en la predicción:')
print(y_pred)
```

Resultado una vez ejecutado el código anterior:

```
Datos de entrenamiento:
[1 0 0 1 0 0 1 1 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
 0 0 0 0 0 1 1 0 0 1 1 1 0 0 1 0 0 0 0 1 1 1 1 0 0 1 1 1 1 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 1 0 0 0 1 0
 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 1 1 0 0 0 0 0 0 1 0 0 0
 0 1 0 1 0 0 1 0 0 0 1 1 1 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 1 0 1 1
 0 1 1 1 0 0 0]

Datos de obtenidos en la predicción:
[1 0 0 0 0 1 0 0 1 1 1 0 0 0 1 1 0 1 1 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0
 0 1 1 0 0 1 1 1 0 0 0 1 0 0 0 0 0 0 1 1 0 0 1 1 1 0 0 1 0 0 0 0 0 0 0
 1 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 1 0 1 0 0 1 0 1 0 1 0 1 0
 1 1 1 0 0 1 0 1 1 1 0 1 0 0 0 1 1 1 0 1 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0
 0 1 0 0 0 0 1 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 1
 0 1 0 0 0 0 0]
```

Se evalúa el error implementando la matriz de confusión y obteniendo el reporte de clasificación.

```
## Matriz de confusión
matriz = confusion_matrix(y_test, y_pred)
print(matriz)
```

Resultado una vez ejecutado el código anterior:

```
[[102  28]
 [ 23  39]]
```

```
## Reporte de clasificación
reporte = classification_report(y_test, y_pred)
print(reporte)
```

Resultado una vez ejecutado el código anterior:

	precision	recall	f1-score	support
0	0.82	0.78	0.80	130
1	0.58	0.63	0.60	62
micro avg	0.73	0.73	0.73	192
macro avg	0.70	0.71	0.70	192
weighted avg	0.74	0.73	0.74	192

Bosques Aleatorios Clasificación

Finalmente, se evalúa el último algoritmo explicado, bosques aleatorios clasificación.

```
### BOSQUES ALEATORIOS CLASIFICACIÓN ###
from sklearn.ensemble import RandomForestClassifier

modelo = RandomForestClassifier()
modelo.fit(X_train, y_train)
y_pred = modelo.predict(X_test)

print('Datos de entrenamiento:')
print(y_test)
print('Datos de obtenidos en la predicción:')
print(y_pred)
```

Resultado una vez ejecutado el código anterior:

```
Datos de entrenamiento:
[1 0 0 1 0 0 1 1 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
 0 0 0 0 0 1 1 0 0 1 1 0 0 1 0 0 0 0 1 1 1 1 0 0 1 1 1 1 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 1 0 0 0 1 0
 1 1 1 1 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 1 1 0 0 0 0 0 1 0 0 0
 0 1 0 1 0 0 1 0 0 0 1 1 1 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 1 1
 0 1 1 1 0 0 0]

Datos de obtenidos en la predicción:
[1 0 0 1 0 0 1 1 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0
 0 0 1 0 0 1 1 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 1 0 0 1 1 1 0 1 0 0 0 0 0
 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0
 1 1 1 1 0 0 1 0 1 0 0 0 0 0 0 0 1 1 0 0 0 1 0 1 0 0 0 0 0 1 0 0 1 0 0 0
 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0
 0 0 0 0 1 0]
```

Se implementa, a continuación, las funciones para evaluar el algoritmo, igual que anteriormente se utiliza la matriz de confusión y el reporte de clasificación.

```
## Matriz de confusión
matriz = confusion_matrix(y_test, y_pred)
print(matriz)
```

Resultado una vez ejecutado el código anterior:

```
[[114  16]
 [ 30  32]]
```

```
## Reporte de clasificación
reporte = classification_report(y_test, y_pred)
print(reporte)
```

Resultado una vez ejecutado el código anterior:

	precision	recall	f1-score	support
0	0.79	0.88	0.83	130
1	0.67	0.52	0.58	62
micro avg	0.76	0.76	0.76	192
macro avg	0.73	0.70	0.71	192
weighted avg	0.75	0.76	0.75	192

Como se puede observar, los algoritmos evaluados obtuvieron unos resultados muy parecidos unos con otros.

En estos casos se selecciona el algoritmo que sea mucho más rápido y fácil de implementar, que sería Regresión Logística.

No todos los casos son así, en ocasiones cuando se está trabajando con un problema de clasificación, los resultados, después de implementar varios algoritmos, son distintos entre ellos, por lo que acá se selecciona el algoritmo con el que se obtengan los mejores resultados.

Capítulo 14

PROYECTO DE MACHINE LEARNING - REGRESIÓN

En este capítulo se desarrollará un proyecto de regresión utilizando Python, se incluye cada paso del proceso de Machine Learning aplicado al problema.

En este capítulo aprenderás:

1. Cómo trabajar a través de un problema de regresión
2. Cómo utilizar transformaciones de datos para mejorar el rendimiento del modelo
3. Cómo implementar algoritmos de Machine Learning y comparar sus resultados

DEFINICIÓN DEL PROBLEMA

Para el proyecto se utilizará el mismo conjunto de datos que se utilizó para explicar los algoritmos de regresión, pero esta vez se desarrollará de manera completa. El conjunto de datos es el correspondiente a los anuncios de publicidad.

El conjunto de datos se puede encontrar en la página de Kaggle como “Advertising Data”.

IMPORTAR LAS LIBRERÍAS

El primer paso en cualquier proyecto de Machine Learning es el de importar las librerías a utilizar. Es normal que al principio no se conozca todas las librerías a utilizar y que se vayan agregando poco a poco, pero por lo menos se debe comenzar con las básicas que normalmente se utilizan al implementar cualquier proyecto de Machine Learning.

Importar las librerías se puede realizar a medida que se vaya programando, lo que recomiendo es que se coloquen todas estas líneas de códigos al principio del programa para que sea fácil conocer los módulos que se están utilizando dentro del programa, así como también mantener un orden en la programación.

Por los momentos se importarán las librerías de Pandas y matplotlib.

```
#### IMPORTAR LAS LIBRERÍAS ####
import pandas as pd
import matplotlib.pyplot as plt
```

CARGAR EL CONJUNTO DE DATOS

Una vez que se hayan importado las librerías ya se pueden utilizar dentro del programa, para esto se procede a cargar el conjunto de datos.

El conjunto de datos se debe descargar directamente de la página de Kaggle y guardarla en el computador en una carpeta específica en donde estará el archivo del conjunto de datos junto con el archivo del programa de Python que se está desarrollando, de esta forma se hará más fácil la programación.

Recuerda que para descargar los archivos de Kaggle deberás estar suscrito en la página, esto es completamente gratis y tendrás a tu disposición un gran número de conjuntos de datos con los que podrás practicar más adelante.

```
#### CARGAR LOS DATOS ####
data = pd.read_csv('Advertising.csv')
```

Se debe colocar el nombre exacto del archivo en donde se encuentra el conjunto de datos. Se recomienda guardar los datos en la misma carpeta en donde se encuentra el archivo del programa de esta forma se mantiene un orden.

ENTENDIENDO LOS DATOS

El siguiente paso en el desarrollo de un proyecto de Machine Learning es el de entender los datos con lo se cuenta. Para este paso se utilizan varias funciones que se tiene disponible en la librería scikit-learn.

```
#### ENTENDIENDO LOS DATOS ####  
#Ver las primeras 5 filas  
data.head()
```

Resultado una vez ejecutado el código anterior:

	Unnamed: 0	TV	radio	newspaper	sales
0	1	230.1	37.8	69.2	22.1
1	2	44.5	39.3	45.1	10.4
2	3	17.2	45.9	69.3	9.3
3	4	151.5	41.3	58.5	18.5
4	5	180.8	10.8	58.4	12.9

```
#Dimensión de los datos  
data.shape
```

Resultado una vez ejecutado el código anterior:

(200, 5)

```
#Tipos de datos  
data.dtypes
```

Resultado una vez ejecutado el código anterior:

```
Unnamed: 0      int64  
TV            float64  
radio         float64  
newspaper     float64  
sales         float64  
dtype: object
```

```
#Descripción estadística  
data.describe()
```

Resultado una vez ejecutado el código anterior:

	Unnamed: 0	TV	radio	newspaper	sales
count	200.000000	200.000000	200.000000	200.000000	200.000000
mean	100.500000	147.042500	23.264000	30.554000	14.022500
std	57.879185	85.854236	14.846809	21.778621	5.217457
min	1.000000	0.700000	0.000000	0.300000	1.600000
25%	50.750000	74.375000	9.975000	12.750000	10.375000
50%	100.500000	149.750000	22.900000	25.750000	12.900000
75%	150.250000	218.825000	36.525000	45.100000	17.400000
max	200.000000	296.400000	49.600000	114.000000	27.000000

```
#Correlación entre características
correlacion = data.corr(method='pearson')
print(correlacion)
```

Resultado una vez ejecutado el código anterior:

	Unnamed: 0	TV	radio	newspaper	sales
Unnamed: 0	1.000000	0.017715	-0.110680	-0.154944	-0.051616
TV	0.017715	1.000000	0.054809	0.056648	0.782224
radio	-0.110680	0.054809	1.000000	0.354104	0.576223
newspaper	-0.154944	0.056648	0.354104	1.000000	0.228299
sales	-0.051616	0.782224	0.576223	0.228299	1.000000

El conjunto de datos cuenta con 5 columnas, de las cuales 4 son las variables independientes y la última columna corresponde a la variable dependiente.

Detallando los datos se puede observar que la primera columna es de numeración por lo que se puede descartar cuando se haga la separación de los datos.

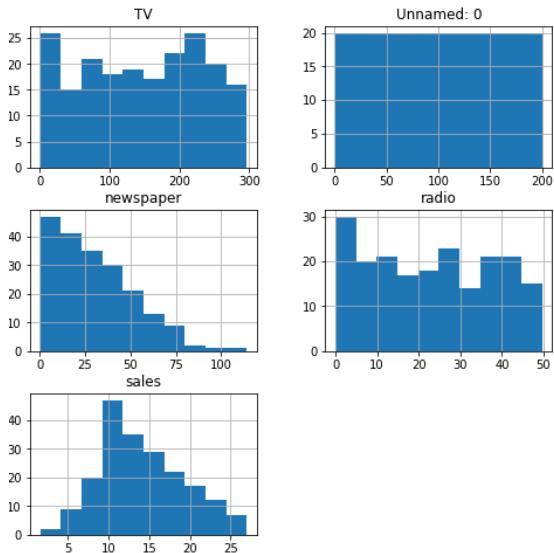
Se puede observar que todo el código de programación utilizado hasta acá es exactamente igual al del problema de clasificación desarrollado en el anterior capítulo.

VISUALIZANDO LOS DATOS

Una vez que se haya entendido los datos de manera numérica ahora se analizará de manera visual utilizando la librería matplotlib. Esta librería fue importada al inicio del programa por lo que no es necesario hacerlo nuevamente.

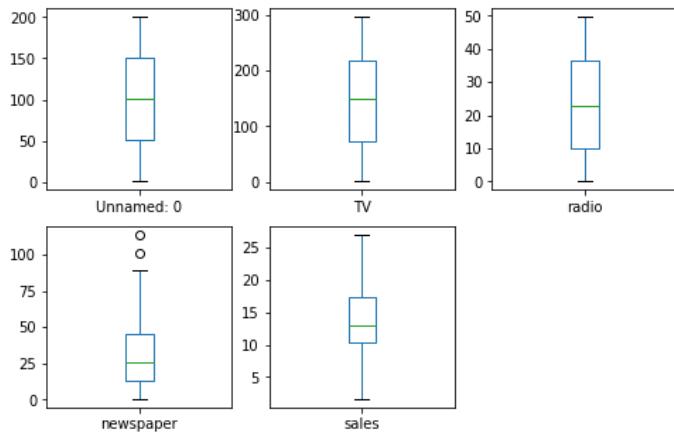
```
#### VISUALIZANDO LOS DATOS ####
#Histograma
data.hist(figsize = (8, 8))
plt.show()
```

Resultado una vez ejecutado el código anterior:



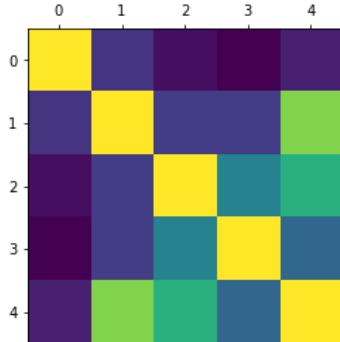
```
#Diagrama de Cajas
data.plot(kind='box', subplots=True, layout=(4,4), sharex=False, sharey=False, figsize = (8, 8))
plt.show
```

Resultado una vez ejecutado el código anterior:



```
#Matriz de Correlación
plt.matshow(data.corr())
plt.show
```

Resultado una vez ejecutado el código anterior:



Como podemos observar en las gráficas obtenidas los resultados son muy similares a los obtenidos en el punto anterior.

Se puede observar que todo el código de programación utilizado hasta acá es exactamente igual al del problema de clasificación desarrollado en el anterior capítulo.

SEPARACIÓN DE LOS DATOS

Conociendo el conjunto de datos que se está trabajando dentro del proyecto, se procede a realizar la separación de los datos en dos conjuntos, el primero de variables independientes, X , y un segundo conjunto de variable dependientes, y .

Se comienza separando los datos correspondientes a las variables independientes, que vendrían siendo todas las columnas menos la primera y la última.

La primera columna se elimina porque contiene la numeración de los filas, información que no influye para el análisis de Machine Learning.

```
### SEPARACIÓN DE LOS DATOS ###
#Datos de entrada o independientes
X = data.iloc[:, 1:-1].values
print(X)
```

Resultado una vez ejecutado el código anterior:

```
[[230.1  37.8  69.2]
 [ 44.5  39.3  45.1]
 [ 17.2  45.9  69.3]
 [151.5  41.3  58.5]
 [180.8  10.8  58.4]
 [  8.7  48.9  75. ]
 [ 57.5  32.8  23.5]
 [120.2  19.6  11.6]
 [  8.6   2.1   1. ]
 [199.8   2.6  21.2]
 [ 66.1   5.8  24.2]
 [214.7  24.    4. ]
 [ 23.8  35.1  65.9]
 [ 97.5   7.6   7.2]
 [204.1  32.9  46. ]
 [195.4  47.7  52.9]
 [ 67.8  36.6 114. ]
 [281.4  39.6  55.8]
 [ 69.2  20.5  18.3]]
```

Seguidamente se define los variables correspondientes a y , o variable dependiente, que vendría siendo la última columna del conjunto de datos.

```
#Datos de salida o dependientes
y = data.iloc[:, -1].values
print(y)
```

Resultado una vez ejecutado el código anterior:

```
[22.1 10.4 9.3 18.5 12.9 7.2 11.8 13.2 4.8 10.6 8.6 17.4 9.2 9.7
19. 22.4 12.5 24.4 11.3 14.6 18. 12.5 5.6 15.5 9.7 12. 15. 15.9
18.9 10.5 21.4 11.9 9.6 17.4 9.5 12.8 25.4 14.7 10.1 21.5 16.6 17.1
20.7 12.9 8.5 14.9 10.6 23.2 14.8 9.7 11.4 10.7 22.6 21.2 20.2 23.7
5.5 13.2 23.8 18.4 8.1 24.2 15.7 14. 18. 9.3 9.5 13.4 18.9 22.3
18.3 12.4 8.8 11. 17. 8.7 6.9 14.2 5.3 11. 11.8 12.3 11.3 13.6
21.7 15.2 12. 16. 12.9 16.7 11.2 7.3 19.4 22.2 11.5 16.9 11.7 15.5
25.4 17.2 11.7 23.8 14.8 14.7 20.7 19.2 7.2 8.7 5.3 19.8 13.4 21.8
14.1 15.9 14.6 12.6 12.2 9.4 15.9 6.6 15.5 7. 11.6 15.2 19.7 10.6
6.6 8.8 24.7 9.7 1.6 12.7 5.7 19.6 10.8 11.6 9.5 20.8 9.6 20.7
10.9 19.2 20.1 10.4 11.4 10.3 13.2 25.4 10.9 10.1 16.1 11.6 16.6 19.
15.6 3.2 15.3 10.1 7.3 12.9 14.4 13.3 14.9 18. 11.9 11.9 8. 12.2
17.1 15. 8.4 14.5 7.6 11.7 11.5 27. 20.2 11.7 11.8 12.6 10.5 12.2
8.7 26.2 17.6 22.6 10.3 17.3 15.9 6.7 10.8 9.9 5.9 19.6 17.3 7.6
9.7 12.8 25.5 13.4]
```

Para este problema no es necesario la selección de características ya que cuenta con muy pocas, por lo que este paso se va a obviar.

De igual forma los datos se encuentran adecuados para ser utilizados dentro de los algoritmos de Machine Learning, por lo que no es necesario realizar un procesamiento a los mismos.

SEPARACIÓN DE LOS DATOS

Se llega al último paso antes de implementar los algoritmos de Machine Learning, este es el correspondiente al de separar los datos en entrenamiento y prueba, para esto se utiliza la función `train_test_split` de la librería scikit-learn, recuerda que se debe importar antes de usarla.

```
from sklearn.model_selection import train_test_split
```

Se utilizará solamente un 25% del conjunto de datos como datos de prueba, por lo que se coloca en el tamaño de la prueba 0.25.

```
#### SEPARACIÓN DE DATOS DE ENTRENAMIENTO Y PRUEBA ####
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

APLICACIÓN DE LOS ALGORITMOS DE REGRESIÓN

Se implementarán los algoritmos explicados con anterioridad y se evaluará los resultados con respecto al error de cada uno de ellos.

Regresión Lineal

El primer algoritmo a evaluar es el más básico de todos y es el de Regresión Lineal.

```
### REGRESIÓN LÍNEAL ###
from sklearn.linear_model import LinearRegression

modelo = LinearRegression()
modelo.fit(X_train, y_train)
y_pred = modelo.predict(X_test)

print('Datos de entrenamiento:')
print(y_test)
print('Datos de obtenidos en la predicción:')
print(y_pred)
```

Resultado una vez ejecutado el código anterior:

```
Datos de entrenamiento:
[11.3  8.4  8.7 25.4 11.7  8.7  7.2 13.2  9.2 16.6 24.2 10.6 10.5 15.6
 11.8 13.2 17.4  1.6 14.7 17.  26.2 10.3 14.9 12.9  8.1 15.2 12.6 22.6
 11.6  8.5 12.5 23.7 16.1 21.8  5.6  6.7  9.7 12.9 13.6  7.2 10.8  9.5
 15.  15.9 17.1 14.  4.8  8.7 15.9 10.4]
Datos de obtenidos en la predicción:
[10.0494569  7.43052335  6.97152143 24.16378667 12.00215643  6.54334645
 13.09526331 14.95879164 11.00528358 16.27234553 22.99324688  9.12188347
 10.33545333 15.39628185 11.60589932 12.11484332 18.60251172 10.81414474
 16.07541355 17.22753644 24.2342995  9.47711838 15.13960412 12.41064749
 5.67814427 15.22889947 12.21635459 20.94370559 13.28068231  9.16578351
 13.30285718 21.5770033 18.098111 21.15572322 6.69734039  6.15355714
 7.96280151 13.09426248 14.81032968 6.22020075 12.2799744  9.1817324
 15.04882696 16.26091437 17.16859664 13.32831849 3.69143664 12.43931798
 15.87909695  8.68626862]
```

Los anteriores datos es la representación de los datos de pruebas, originales, con los datos obtenidos por el modelo. A continuación, se realiza el análisis respectivo con las métricas correspondientes a los algoritmos de regresión.

La primera métrica a evaluar será el error cuadrático medio, para ello primero se importará la librería y posteriormente la función.

```
from sklearn.metrics import mean_squared_error
from math import sqrt
```

```
## Error cuadrático medio
error = sqrt(mean_squared_error(y_test, y_pred))
print(error)
```

Resultado una vez ejecutado el código anterior:

2.0031219440955406

Los resultados obtenidos acá son bastante satisfactorios, recuerdo que si el valor está más cercano a 0 quiere indicar que el modelo es bueno.

Ahora se evalúa los resultados, pero esta vez utilizando la métrica R^2 , importando primero la función en scikit-learn para luego implementarla.

```
from sklearn.metrics import r2_score
```

```
## R2
error = r2_score(y_test, y_pred)
print(error)
```

Resultado una vez ejecutado el código anterior:

```
0.8576396745320893
```

Esta métrica, a diferencia a la anterior, entre más cerca este el resultado a 1 es un muy buen modelo, por lo que el resultado obtenido indica que el modelo desarrollado es bueno.

Regresión Polinomial

El siguiente algoritmo a evaluar será el de regresión polinomial, acá se realizará el mismo procedimiento que se realizó anteriormente.

```
### REGRESIÓN POLINOMIAL ###
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

poli_reg = PolynomialFeatures(degree = 2)
X_train_poli = poli_reg.fit_transform(X_train)
X_test_poli = poli_reg.fit_transform(X_test)
modelo = LinearRegression()
modelo.fit(X_train_poli, y_train)
y_pred = modelo.predict(X_test_poli)

print('Datos de entrenamiento:')
print(y_test)
print('Datos de obtenidos en la predicción:')
print(y_pred)
```

Resultado una vez ejecutado el código anterior:

```

Datos de entrenamiento:
[11.3 8.4 8.7 25.4 11.7 8.7 7.2 13.2 9.2 16.6 24.2 10.6 10.5 15.6
 11.8 13.2 17.4 1.6 14.7 17. 26.2 10.3 14.9 12.9 8.1 15.2 12.6 22.6
 11.6 8.5 12.5 23.7 16.1 21.8 5.6 6.7 9.7 12.9 13.6 7.2 10.8 9.5
 15. 15.9 17.1 14. 4.8 8.7 15.9 10.4]
Datos de obtenidos en la predicción:
[10.29606718 8.39661197 8.93320477 25.26165017 12.29558549 8.22685217
 8.49198281 13.31649422 8.81368555 16.6224964 24.37512292 10.2108201
 11.02194003 15.79199714 11.43816419 12.80603999 17.50687149 6.66253157
 14.24705479 17.47477709 25.46581294 10.53711542 15.59666759 13.09955877
 7.77382034 15.47757867 12.6593029 22.50525107 11.82433019 8.04189778
 13.11841618 23.29272426 15.43681397 22.20509974 6.76196984 6.71905519
 9.02670414 13.24325578 13.04680682 7.05165972 10.13225129 8.2684012
 15.62172361 16.3345946 17.21535271 13.44388485 5.69433371 9.18091154
 16.35045357 10.11289876]

```

Realizado todo este procedimiento se continúa evaluando el modelo, para ello se aplica las dos métricas que se utilizó con el anterior, primeramente, el error cuadrático medio.

```

## Error cuadrático medio
error = sqrt(mean_squared_error(y_test, y_pred))
print(error)

```

Resultado una vez ejecutado el código anterior:

0.888785137887519

Y posteriormente se obtiene la métrica R^2 .

```

## R2
error = r2_score(y_test, y_pred)
print(error)

```

Resultado una vez ejecutado el código anterior:

0.9719735711900083

Como se puede observar los resultados obtenidos acá son muy parecidos al anterior modelo.

Vectores de Soporte Regresión

Ahora se evalúa el algoritmo de vectores de soporte regresión, el procedimiento es muy similar a los otros algoritmos evaluados anteriormente.

```

### VECTORES SOPORTE REGRESIÓN ###
from sklearn.svm import SVR

modelo = SVR()
modelo.fit(X_train, y_train)
y_pred = modelo.predict(X_test)

print('Datos de entrenamiento:')
print(y_test)
print('Datos de obtenidos en la predicción:')
print(y_pred)

```

Resultado una vez ejecutado el código anterior:

```
Datos de entrenamiento:  
[11.3 8.4 8.7 25.4 11.7 8.7 7.2 13.2 9.2 16.6 24.2 10.6 10.5 15.6  
 11.8 13.2 17.4 1.6 14.7 17. 26.2 10.3 14.9 12.9 8.1 15.2 12.6 22.6  
 11.6 8.5 12.5 23.7 16.1 21.8 5.6 6.7 9.7 12.9 13.6 7.2 10.8 9.5  
 15. 15.9 17.1 14. 4.8 8.7 15.9 10.4]  
Datos de obtenidos en la predicción:  
[13.34615384 13.34615385 13.34615385 13.34615385 13.34615385 13.34615385  
 13.34615385 13.34615385 13.34615385 13.34615385 13.34615385 13.34615385  
 13.34615385 13.34615385 13.3461524 13.34615385 13.34615385 13.34615385  
 13.34615385 13.34615409 13.34615395 13.34615355 13.34628137 13.34615385  
 13.34615385 13.34615396 13.34615179 13.34615385 13.30814742 13.34615385  
 13.34615385 13.34615385 13.34615385 13.34615747 13.34615385 13.34258912  
 13.34615385 13.34615385 13.34615385 13.34615385 13.34615385 13.34615385  
 13.34615385 13.34615385 13.34615436 13.34615385 13.34615385 13.34615385  
 13.34615385 13.34615385]
```

Igual que en los casos anteriores, se evalúa el modelo obtenido por medio del error cuadrático medio y la métrica R^2 .

```
## Error cuadrático medio  
error = sqrt(mean_squared_error(y_test, y_pred))  
print(error)
```

Resultado una vez ejecutado el código anterior:

5.3177172515792

```
## R2  
error = r2_score(y_test, y_pred)  
print(error)
```

Resultado una vez ejecutado el código anterior:

-0.003285829756425107

Si se observa los resultados de las métricas de evaluación no son para nada buenos, por lo que se puede deducir que este no es el mejor algoritmo para este conjunto de datos.

Árboles de Decisión Regresión

El siguiente algoritmo a evaluar será el de árboles de decisión regresión.

```
### ÁRBOLES DE DECISIÓN REGRESIÓN ###
from sklearn.tree import DecisionTreeRegressor

modelo = DecisionTreeRegressor()
modelo.fit(X_train, y_train)
y_pred = modelo.predict(X_test)

print('Datos de entrenamiento:')
print(y_test)
print('Datos de obtenidos en la predicción:')
print(y_pred)
```

Resultado una vez ejecutado el código anterior:

```
Datos de entrenamiento:
[11.3  8.4  8.7 25.4 11.7  8.7  7.2 13.2  9.2 16.6 24.2 10.6 10.5 15.6
 11.8 13.2 17.4  1.6 14.7 17.  26.2 10.3 14.9 12.9  8.1 15.2 12.6 22.6
 11.6  8.5 12.5 23.7 16.1 21.8  5.6  6.7  9.7 12.9 13.6  7.2 10.8  9.5
 15.  15.9 17.1 14.  4.8  8.7 15.9 10.4]
Datos de obtenidos en la predicción:
[11.3  7.6  9.6 25.5 12.8  8.6  6.6 12.3  8.8 17.4 25.5 10.6 10.6 15.5
 12.  11.9 15.9  6.6 11.3 17.4 25.5 10.6 16.6 13.2  8.6 14.2 14.4 21.2
 10.6  7.  13.3 19.2 15.7 22.3  5.5  6.6  9.7 14.7 11.3  6.9 13.3  7.6
 16.9 15.5 16.6 14.2  5.7  9.3 16.  10.7]
```

Definido el modelo, se verifica el error del mismo, para ello se utiliza la función del error cuadrático medio y la métrica R^2 .

```
## Error cuadrático medio
error = sqrt(mean_squared_error(y_test, y_pred))
print(error)
```

Resultado una vez ejecutado el código anterior:

1.4522396496446446

```
## R2
error = r2_score(y_test, y_pred)
print(error)
```

Resultado una vez ejecutado el código anterior:

0.92517430236479

Bosques Aleatorios Regresión

Para evaluar el problema implementando el algoritmo de bosques aleatorios regresión se realiza el siguiente procedimiento.

```
### BOSQUES ALEATORIOS REGRESIÓN ###
from sklearn.ensemble import RandomForestRegressor

modelo = RandomForestRegressor()
modelo.fit(X_train, y_train)
y_pred = modelo.predict(X_test)

print('Datos de entrenamiento:')
print(y_test)
print('Datos de obtenidos en la predicción:')
print(y_pred)
```

Resultado una vez ejecutado el código anterior:

```
Datos de entrenamiento:
[11.3  8.4  8.7 25.4 11.7  8.7  7.2 13.2  9.2 16.6 24.2 10.6 10.5 15.6
 11.8 13.2 17.4  1.6 14.7 17.   26.2 10.3 14.9 12.9  8.1 15.2 12.6 22.6
 11.6  8.5 12.5 23.7 16.1 21.8  5.6  6.7  9.7 12.9 13.6  7.2 10.8  9.5
 15.  15.9 17.1 14.   4.8  8.7 15.9 10.4]
Datos de obtenidos en la predicción:
[10.85  9.63  9.44 24.59 12.57  9.02  8.09 12.35  8.62 15.57 24.2 10.39
 10.29 16.02 11.57 12.43 15.69  5.74 13.02 16.06 25.5  10.32 15.67 12.98
 8.99 14.83 14.06 21.25 11.11  7.41 12.08 22.78 15.85 22.59  6.21  6.98
 9.59 14.3 12.8  7.13 10.45  7.54 17.02 16.02 16.03 11.52  5.62  9.1
 15.91 10.58]
```

Se evalúa el error implementando el error cuadrático medio y la métrica R^2 .

```
## Error cuadrático medio
error = sqrt(mean_squared_error(y_test, y_pred))
print(error)
```

Resultado una vez ejecutado el código anterior:

1.0933197153623457

```
## R2
error = r2_score(y_test, y_pred)
print(error)
```

Resultado una vez ejecutado el código anterior:

0.9575899724908237

Como se puede observar, los algoritmos evaluados obtuvieron unos resultados muy parecidos unos con otros, exceptuando el de vectores de soporte regresión. Esto no quiere decir que este algoritmo sea malo, sino que simplemente no es el más adecuado para este conjunto de datos.

Se puede elegir cualquier algoritmo con el que se obtuvieron buenos resultado, en caso de que se requiera un algoritmo que sea rápido y fácil de implementar se puede elegir el de Regresión Lineal o el de Regresión Polinomial.

Capítulo 15

CONTINUAR APRENDIENDO MACHINE LEARNING

En este capítulo encontrarás áreas en las que podrás practicar las nuevas habilidades adquiridas de Machine Learning con Python.

En este capítulo aprenderás:

1. Encontrar conjuntos de datos para trabajar en nuevos proyectos

CONSTRUIR PLANTILLAS PARA PROYECTOS

A lo largo del libro, se ha explicado cómo desarrollar proyectos de Machine Learning utilizando Python. Lo explicado acá es la base que puede ser usada para iniciar nuevos proyectos de Machine Learning. Esto es solamente un inicio, y puedes ir mejorando a medida que desarrolles proyectos más grandes y complicados.

A medida que se aplique lo explicado acá y implementes las habilidades adquiridas en Machine Learning utilizando la plataforma de Python, desarrollarás experiencia y habilidades con nuevas y diferentes técnicas con Python, haciendo más fácil el desarrollo de proyectos dentro de esta área.

CONJUNTOS DE DATOS PARA PRÁCTICA

Lo importante para mejorar las habilidades aprendidas dentro del libro es seguir practicando. En la web puedes encontrar varios conjuntos de datos que puedes utilizar para practicar los conocimientos y, a su vez para crear un portafolio de proyectos para mostrar dentro de la hoja de vida.

El primer lugar donde puedes encontrar conjuntos de datos que puedes utilizar en proyectos de Machine Learning es en el repositorio de Machine Learning de UCI (Universidad de California en Irvine). Los conjuntos de datos de este repositorio son estandarizados, relativamente limpios, bien entendidos y excelentes para que se use como conjuntos de datos de práctica.

Con este repositorio se puede construir y desarrollar aún más las habilidades en el área de Machine Learning. También es útil para empezar a crear un portafolio que puede ser mostrado a futuros empleadores demostrando que puede ser capaz de entregar resultados en proyectos de Machine Learning utilizando Python.

El link para este repositorio es el siguiente: <http://archive.ics.uci.edu/ml/index.php>.

Otra página que cuenta con muy buenos conjuntos de datos para practicar es Kaggle. Acá, puedes descargar y practicar las habilidades aprendidas, pero también puedes participar en competencias. En una competencia, el organizador te proporciona un conjunto de datos de entrenamiento, un conjunto de datos de prueba en el que debes hacer predicciones, una medida de rendimiento y un límite de tiempo. Luego, los competidores trabajan para crear el modelo más preciso posible. Los ganadores a menudo reciben premios en efectivo.

Estas competencias a menudo duran semanas o meses y pueden ser muy divertidas. También ofrecen una gran oportunidad para probar tus habilidades con herramientas de Machine Learning con conjuntos de datos que a menudo requieren mucha limpieza y preparación. Un buen sitio para comenzar serían las competencias para principiantes, ya que a menudo son menos desafiantes y tienen mucha ayuda en forma de tutoriales para comenzar.

El sitio web principal para encontrar los conjuntos de datos es el siguiente: <https://www.kaggle.com/datasets>.

No importa la página que se utilice para obtener los conjuntos de datos, los pasos que se deben seguir para mejorar las habilidades de Machine Learning son los siguientes:

1. Navegar por la lista de conjuntos de datos gratuitos en el repositorio y descargar algunos que parezcan interesantes.

2. Usar las librerías y funciones aprendidas en este libro para trabajar a través del conjunto de datos y desarrollar un modelo preciso.
3. Escribir el flujo de trabajo y las conclusiones obtenidas para que puedan ser consultados más adelante e inclusive si es posible compartir esta información en algún sitio web, un lugar recomendado es en Github.

Trabajar en pequeños proyectos es una buena manera de practicar los fundamentos. En ocasiones algunos problemas se volverán fáciles por lo cual se debe buscar nuevos retos para salir de la zona de confort para ayudar a aumentar las habilidades en Machine Learning.

Capítulo 16

OBTENER MÁS INFORMACIÓN

Este libro es solo el comienzo de tu camino para aprender Machine Learning con Python. A medida que se desarrolle nuevos proyectos, es posible que necesites ayuda. En este capítulo se indica algunas de las mejores fuentes de ayuda de Python y Machine Learning que puedes encontrar.

En este capítulo aprenderás:

1. Sitios web útiles para aclarar dudas.

CONSEJO GENERAL

En general, en la página LigdiGonzalez podrás encontrar mucha más información sobre Machine Learning. Desde información teórica como ejercicios prácticos, de esta forma podrás aumentar tus habilidades dentro de este tema. Toda la información contenida acá es en español.



Para mayor información se puede revisar la siguiente información:

<http://bit.ly/2JbWcaX>

De igual forma, la documentación oficial de Python y SciPy es excelente. Tanto las guías de usuario como la documentación de API son una excelente ayuda para aclarar dudas. Acá podrás tener una comprensión más completa de la configuración más profunda que puedes explorar. La información publicada es en inglés.

Otro recurso muy útil son los sitios de preguntas y respuestas, como StackOverflow. Se puede buscar mensajes de error y problemas que se tenga y encontrar ejemplos de códigos e ideas que pueden ayudar en los proyectos. Esta página se encuentra tanto en español como en inglés, aunque en esta última se encuentra la mayor cantidad de información.

AYUDA CON PYTHON

Python es un lenguaje de programación con todas las funciones. Como tal, cuanto más aprendas sobre él, mejor podrás usarlo. Si es relativamente nuevo en la plataforma Python, aquí hay algunos recursos valiosos para ir un paso más profundo:

Documentación oficial Python 3:

<https://docs.python.org/3/>

AYUDA CON SCIPY Y NUMPY

Es una buena idea familiarizarse con el ecosistema SciPy más amplio, por lo que es recomendable revisar las notas de SciPy y la documentación de NumPy, sobretodo cuando se tenga problemas con estas librerías.

Notas de SciPy:

<http://scipy-lectures.org/>

Documentación oficial NumPy:

<https://docs.scipy.org/doc/numpy/user/>

AYUDA CON MATPLOTLIB

Mostrar gráficamente los datos es algo muy importante en Machine Learning, por lo que se puede revisar la información oficial de matplotlib, en donde se encuentran

muchos ejemplos, con sus respectivos códigos, que pueden ser bastante útil para implementar en los proyectos propios.

Documentación oficial matplotlib:
<https://matplotlib.org/gallery.html>

AYUDA CON PANDAS

Pandas cuenta con una gran cantidad de documentación. Los ejemplos presentados en la documentación oficial son muy útiles ya que darán ideas sobre diferentes maneras de dividir y cortar los datos.

Documentación oficial Pandas:
<http://pandas.pydata.org/pandas-docs/stable/>

AYUDA CON SCIKIT-LEARN

La documentación publicada en la página web de scikit-learn es de gran ayuda al momento de desarrollar proyectos de Machine Learning, revisar la configuración de cada función puede ayudar a mejorar los proyectos y obtener mejores resultados.

Documentación oficial scikit-learn:
<https://scikit-learn.org/stable/documentation.html>

Ligdi González

2020

Machine Learning con Python

Aprendizaje Supervisado