Argentina programa 4.0

Ministerio de Desarrollo Productivo Argentina

Secretaría de Economía del Conocimiento

PROGRAMADOR JUNIOR EN MACHINE LEARNING I

UNIVERSIDAD NACIONAL DE MISIONES

FACULTAD DE CIENCIAS EXACTAS QUÍMICAS Y NATURALES

CURSO 1: Fundamentos de Machine Learning (ML) Python - Nivel 1 -

FUNDAMENTACIÓN

Python es un lenguaje de programación ampliamente utilizado en las aplicaciones web, el desarrollo de software, la ciencia de datos y el machine learning (ML). Los desarrolladores utilizan Python porque es eficiente y fácil de aprender, además de que se puede ejecutar en muchas plataformas diferentes.

Python es un lenguaje sencillo de leer y escribir debido a su alta similitud con el lenguaje humano. Además, se trata de un lenguaje multiplataforma de código abierto y, por lo tanto, gratuito, lo que permite desarrollar software sin límites. Python puede utilizarse en proyectos de inteligencia artificial, para crear sitios web escalables, realizar cálculos estructurales complejos con elementos finitos, y diseñar videojuegos, entre otras muchas aplicaciones

Objetivos

- Familiarizar al estudiante con el entorno y las herramientas de programación de Python.
- Desarrollar las capacidades de plasmar una propuesta de resolución a un problema sencillo planteado utilizando las herramientas planteadas.
- Identificar las ventajas del trabajo colaborativo y las herramientas disponibles.
- Promover la indagación sobre los temas planteados.

Análisis exploratorio de datos

El análisis exploratorio de datos (EDA) es un proceso que consiste en examinar y resumir los datos de una manera sistemática y visual. El objetivo del EDA es obtener una comprensión general de los datos, identificar posibles patrones, anomalías, relaciones y tendencias. Una de las herramientas más populares para realizar el EDA es Pandas, una biblioteca de Python que ofrece una gran variedad de funciones para manipular y analizar datos tabulares.

Pandas permite realizar operaciones básicas de estadística descriptiva sobre los datos, como calcular la media, la mediana, la desviación estándar, los cuartiles y los percentiles de las variables numéricas, o contar la frecuencia, la proporción y la moda de las variables categóricas. Estas medidas ayudan a describir las características principales de los datos y a detectar posibles valores atípicos o errores.

Además, Pandas facilita la realización de agrupaciones y filtros sobre los datos, lo que permite segmentarlos según diferentes criterios y extraer subconjuntos de interés. Por ejemplo, se puede agrupar los datos por una variable categórica y calcular la media de otra variable numérica para cada grupo, o se puede filtrar los datos según una condición lógica y obtener solo las filas que la cumplen. Estas operaciones permiten explorar las diferencias y las similitudes entre los grupos y las variables.

Finalmente, Pandas también ofrece varias opciones para visualizar los datos mediante gráficos como histogramas, diagramas de caja, gráficos de barras, gráficos de dispersión o mapas de calor. Estos gráficos ayudan a complementar el análisis numérico y a revelar patrones visuales que podrían pasar desapercibidos en una tabla. La visualización también facilita la comunicación y la presentación de los resultados del EDA.

Introducción a la librería Pandas

Pandas es una biblioteca de Python que ofrece estructuras de datos y herramientas de análisis de alto nivel para facilitar el manejo y la manipulación de datos. Pandas permite trabajar con datos tabulares, como hojas de cálculo o bases de datos, y ofrece diversas operaciones para filtrar, agrupar, transformar y visualizar los datos. Además, Pandas se integra con otras bibliotecas de Python, como NumPy, SciPy o Matplotlib, para ampliar sus funcionalidades y ofrecer soluciones para problemas complejos de ciencia de datos.

Instalación

La librería Pandas es una herramienta muy útil para el análisis y manipulación de datos en Python. Para poder utilizarla, es necesario instalarla e importarla en el entorno virtual de Python. A continuación se explican los pasos para hacerlo.

Para instalar la librería Pandas, se puede utilizar el gestor de paquetes pip, que permite instalar paquetes de Python desde la línea de comandos. El comando para instalar Pandas es:

```
pip install pandas
```

Este comando descarga e instala la última versión de Pandas y sus dependencias. Se recomienda ejecutar este comando dentro del entorno virtual de Python que se vaya a utilizar, para evitar conflictos con otras versiones o librerías.

Una vez instalada la librería Pandas, se puede importar en el código de Python con el siguiente código:

```
import pandas as pd
```

Este comando importa la librería Pandas con el alias pd, que es el más común y recomendado. De esta forma, se puede acceder a las funciones y clases de Pandas con el prefijo pd. Por ejemplo:

```
df = pd.DataFrame(data)
```

Este código crea un objeto DataFrame, que es una estructura de datos tabular de Pandas, a partir de un diccionario llamado data.

Estructuras de datos en Pandas

Introducción

Pandas es una biblioteca de Python que ofrece herramientas para el análisis y la manipulación de datos. Una de las características más importantes de Pandas es que proporciona dos tipos de estructuras de datos: las series y los dataframes. Estas estructuras de datos permiten almacenar, acceder y modificar los datos de forma eficiente y sencilla.

Las series son arreglos unidimensionales etiquetados que pueden contener cualquier tipo de dato (numérico, categórico, texto, etc.). Los dataframes son arreglos bidimensionales etiquetados que pueden contener diferentes tipos de datos en cada columna. Tanto las series como los dataframes tienen un índice que identifica cada fila o elemento.

Para crear una serie o un dataframe, se puede usar el constructor de Pandas o pasarle una lista, un diccionario, un arreglo de NumPy o un archivo CSV. También se puede acceder a los elementos de una serie o un dataframe mediante el índice, el nombre de la columna o la posición. Además, se pueden aplicar operaciones y funciones sobre las series y los dataframes, como filtrar, ordenar, agrupar, agregar, fusionar o transformar los datos.

Pandas es una biblioteca de Python que ofrece estructuras de datos y herramientas de análisis de datos de alto nivel y fáciles de usar. Las dos estructuras de datos principales que ofrece Pandas son las Series y los DataFrames.

Series y DataFrames

Una Serie es un objeto similar a un array unidimensional que puede contener cualquier tipo de datos (enteros, cadenas, flotantes, objetos de Python, etc.). Además, una Serie tiene un índice, que es una etiqueta que identifica cada elemento de la Serie. El índice puede ser numérico, alfanumérico o basado en fechas.

Un DataFrame es un objeto similar a una tabla bidimensional que puede contener varios tipos de datos en diferentes columnas. Cada columna de un DataFrame es una Serie, y comparten el mismo índice. Un DataFrame puede crearse a partir de diccionarios, listas, arrays, Series u otros DataFrames.

Pandas ofrece muchas funciones y métodos para manipular, filtrar, agrupar, combinar y visualizar los datos almacenados en las Series y los DataFrames. Algunos ejemplos de código son:

```
# Crear una Serie a partir de una lista
s = pd.Series([1, 2, 3, 4, 5])
print(s)


# Crear un DataFrame a partir de un diccionario
df = pd.DataFrame({'nombre': ['Ana', 'Luis', 'Pedro'], 'edad': [25, 32, 28], 'sexo': ['F', 'M', 'M']})
print(df)
# Seleccionar una columna de un DataFrame
print(df['nombre'])
# Filtrar los datos de un DataFrame según una condición
print(df[df['edad'] > 30])
# Agrupar los datos de un DataFrame por una columna y aplicar una función de agregación
print(df.groupby('sexo').mean())
# Combinar dos DataFrames por una columna común
df2 = pd.DataFrame({'nombre': ['Ana', 'Luis', 'Pedro'], 'ciudad': ['Madrid', 'Barcelona', 'Valencia']})
print(pd.merge(df, df2, on='nombre'))
# Visualizar los datos de un DataFrame con un gráfico de barras
df.plot.bar(x='nombre', y='edad')
plt.show()
```

Operaciones básicas con Pandas

Para crear un DataFrame, podemos usar el constructor pandas.DataFrame(), que recibe como argumento una lista de listas, un diccionario, un array de NumPy o cualquier otro objeto iterable. Por ejemplo, podemos crear un DataFrame con los datos de cuatro estudiantes y sus notas en tres asignaturas:

```
import pandas as pd
datos = [["Ana", 8.5, 9.0, 7.5],
         ["Luis", 6.0, 7.0, 8.0],
         ["Pedro", 9.5, 8.5, 9.0],
         ["Sofía", 7.0, 6.5, 7.5]]
```

```
df = pd.DataFrame(datos, columns=["Nombre", "Matemáticas", "Lengua", "Historia"])

print(df)
```

El resultado es el siguiente:

```
   Nombre  Matemáticas  Lengua  Historia
0     Ana          8.5     9.0       7.5
1    Luis          6.0     7.0       8.0
2   Pedro          9.5     8.5       9.0
3   Sofía          7.0     6.5       7.5
```

Podemos observar que el DataFrame tiene un índice numérico por defecto, que va desde el 0 hasta el número de filas menos uno. También podemos especificar el índice al crear el DataFrame, pasando un argumento index con una lista de valores únicos:

```
df = pd.DataFrame(datos, columns=["Nombre", "Matemáticas", "Lengua", "Historia"], index=["A", "B", "C", "D"])

print(df)
```

El resultado es el siguiente:

```
   Nombre  Matemáticas  Lengua  Historia
A     Ana          8.5     9.0       7.5
B    Luis          6.0     7.0       8.0
C   Pedro          9.5     8.5       9.0
D   Sofía          7.0     6.5       7.5
```

Para leer datos desde un archivo externo, como un CSV o un Excel, podemos usar las funciones pandas.read_csv() o pandas.read_excel(), respectivamente. Estas funciones devuelven un DataFrame con los datos del archivo, y tienen varios parámetros para ajustar la lectura según el formato del archivo y nuestras preferencias.

Por ejemplo, si tenemos un archivo llamado notas.csv con los mismos datos que el DataFrame anterior, pero separados por punto y coma (;) y sin nombres de columnas, podemos leerlo de la siguiente forma:

```
df = pd.read_csv("notas.csv", sep=";", header=None)

df.columns = ["Nombre", "Matemáticas", "Lengua", "Historia"]

print(df)
```

El resultado es el mismo que el del primer DataFrame.

Para modificar los datos de un DataFrame, podemos usar diferentes métodos y operadores según lo que queramos hacer.

Por ejemplo, si queremos añadir una nueva columna con la media de las notas de cada estudiante, podemos hacer lo siguiente:

```
df["Media"] = (df["Matemáticas"] + df["Lengua"] + df["Historia"]) / 3

print(df)
```

El resultado es el siguiente:

```
   Nombre  Matemáticas  Lengua  Historia     Media
A     Ana          8.5     9.0       7.5  8.333333
B    Luis          6.0     7.0       8.0  7.000000
C   Pedro          9.5     8.5       9.0  9.000000
D   Sofía          7.0     6.5       7.5  7.000000
```

Si queremos modificar el valor de una celda específica, podemos usar el método .loc[] o .iloc[], que permiten acceder a los datos por etiqueta o por posición, respectivamente.

Por ejemplo, si queremos cambiar la nota de Matemáticas de Luis por un 6.5, podemos hacer lo siguiente:

```
df.loc["B", "Matemáticas"] = 6.5

print(df)
```

El resultado es el siguiente:

```
   Nombre  Matemáticas  Lengua  Historia     Media
A     Ana          8.5     9.0       7.5  8.333333
B    Luis          6.5     7.0       8.0  7.166667
C   Pedro          9.5     8.5       9.0  9.000000
D   Sofía          7.0     6.5       7.5  7.000000
```

Para eliminar datos de un DataFrame, podemos usar el método .drop(), que recibe como argumento el índice o el nombre de las filas o columnas que queremos eliminar, y el parámetro axis que indica si se trata de filas (axis=0) o columnas (axis=1).

Por ejemplo, si queremos eliminar la columna Media, podemos hacer lo siguiente:

```
df = df.drop("Media", axis=1)

print(df)
```

El resultado es el siguiente:

```
   Nombre  Matemáticas  Lengua  Historia
A     Ana          8.5     9.0       7.5
B    Luis          6.5     7.0       8.0
C   Pedro          9.5     8.5       9.0
D   Sofía          7.0     6.5       7.5
```

Funciones de análisis exploratorio

En este texto se presentan algunos ejemplos de código Python con funciones estadísticas descriptivas, agrupaciones, filtros y visualización en Pandas. Pandas es una librería de Python que ofrece estructuras de datos y herramientas de análisis de datos de alto nivel y fácil de usar. Una de las estructuras principales de Pandas es el DataFrame, que representa una tabla bidimensional con filas y columnas etiquetadas.

Para ilustrar el uso de Pandas, se utilizará un conjunto de datos sobre el consumo de alcohol en diferentes países. El conjunto de datos se puede descargar desde este enlace: https://raw.githubusercontent.com/plotly/datasets/master/2010_alcohol_consumption_by_country.csv

Primero, se importan las librerías necesarias y se lee el archivo CSV con la función read_csv:

```
import pandas as pd

import matplotlib.pyplot as plt

df = pd.read_csv('2010_alcohol_consumption_by_country.csv')
```

Luego, se puede explorar el DataFrame con algunas funciones básicas:

```
# Mostrar las primeras 5 filas

df.head()

# Mostrar las últimas 5 filas

df.tail()

# Mostrar el número de filas y columnas

df.shape

# Mostrar los nombres de las columnas

df.columns

# Mostrar el tipo de dato de cada columna

df.dtypes

# Mostrar información general del DataFrame

df.info()

# Mostrar estadísticas descriptivas de las columnas numéricas

df.describe()
```

Manipulación de datos con Pandas

En este artículo, vamos a ver algunos ejemplos de código Python sobre cómo realizar diferentes operaciones de manipulación de datos con Pandas, tales como:

- Limpieza de datos: eliminar valores nulos o duplicados, reemplazar o imputar valores faltantes, cambiar el tipo de dato de las columnas, etc.

- Transformación de datos: aplicar funciones o cálculos a las columnas o filas, crear nuevas columnas derivadas, filtrar o seleccionar subconjuntos de datos, etc.

- Combinación de datos: unir o fusionar diferentes fuentes de datos en un solo DataFrame, concatenar o apilar DataFrames, etc.

- Pivoteo de datos: cambiar la forma o la estructura del DataFrame, crear tablas dinámicas o resúmenes estadísticos, agrupar o segmentar los datos por categorías, etc.

```
import pandas as pd

# Crear un DataFrame con datos de ejemplo

data = {

    'Ciudad': ['Ciudad A', 'Ciudad B', 'Ciudad C', 'Ciudad A', 'Ciudad B'],

        'Fecha': ['2022-01-01', '2022-01-02', '2022-01-03', '2022-01-01', '2022-01-02'],

    'Ventas': [1000, 2000, 1500, 1800, 2500],

    'Gastos': [800, 1200, 1000, 900, 1500]

}

df = pd.DataFrame(data)

# Mostrar el DataFrame original

print("DataFrame original:")

print(df)

print()

# Limpieza de datos

df = df.drop_duplicates()  # Eliminar filas duplicadas

df = df.dropna()  # Eliminar filas con valores faltantes

df = df.reset_index(drop=True)  # Resetear el índice del DataFrame

# Transformación de datos

df['Utilidad'] = df['Ventas'] - df['Gastos']  # Agregar una nueva columna de utilidad

df['Fecha'] = pd.to_datetime(df['Fecha'])  # Convertir la columna de fechas a tipo datetime

# Combinación de datos

data2 = {

    'Ciudad': ['Ciudad C', 'Ciudad D'],
```

```
    'Fecha': ['2022-01-04', '2022-01-03'],
    'Ventas': [1200, 1800],
    'Gastos': [1000, 900]
}
df2 = pd.DataFrame(data2)
df = pd.concat([df, df2])  # Concatenar el DataFrame original con df2
# Pivoteo de datos
pivot_table = df.pivot_table(index='Ciudad', columns='Fecha', values='Utilidad', aggfunc='sum', fill_value=0)
# Mostrar el DataFrame modificado y la tabla pivote
print("DataFrame modificado:")
print(df)
print()
print("Tabla pivote:")
print(pivot_table)
```

Aplicaciones de Pandas

Pandas es una biblioteca de Python que ofrece herramientas para el análisis y la manipulación de datos. Pandas permite trabajar con estructuras de datos como Series y DataFrames, que facilitan el acceso, la exploración y la transformación de los datos. Pandas también se integra con otras bibliotecas de Python como NumPy, SciPy y Matplotlib, lo que amplía sus posibilidades y funcionalidades.

Pandas tiene muchas aplicaciones en diferentes contextos y problemas. Algunos ejemplos prácticos de uso de Pandas son:

- Análisis exploratorio de datos: Pandas permite realizar operaciones básicas de estadística descriptiva, como calcular medidas de tendencia central, dispersión y correlación, así como visualizar los datos mediante gráficos de barras, histogramas, diagramas de caja o mapas de calor. Estas técnicas ayudan a comprender las características y distribuciones de los datos, así como a identificar posibles anomalías o patrones.

- Limpieza y preparación de datos: Pandas ofrece métodos para tratar con datos faltantes, duplicados o erróneos, como eliminarlos, reemplazarlos o imputarlos. También permite realizar operaciones de filtrado, selección, agrupación y agregación de los datos, así como combinar o fusionar diferentes fuentes de datos. Estas tareas son esenciales para obtener un conjunto de datos limpio y homogéneo que pueda ser utilizado para el análisis posterior.

- Análisis de series temporales: Pandas facilita el manejo de datos que tienen una dimensión temporal, como fechas o horas. Pandas permite crear índices temporales, convertir entre diferentes formatos o zonas horarias, generar rangos o frecuencias de tiempo, y realizar operaciones como resampleo, desplazamiento o interpolación. Estas funciones permiten

analizar el comportamiento de los datos a lo largo del tiempo, así como detectar tendencias, estacionalidades o ciclos.

- Análisis de datos tabulares: Pandas es una herramienta muy útil para trabajar con datos que tienen una estructura tabular, como hojas de cálculo o bases de datos relacionales. Pandas permite leer y escribir archivos en diferentes formatos, como CSV, Excel o SQL. También permite realizar consultas, operaciones lógicas y aritméticas, y funciones de agregación sobre los datos. Estas capacidades facilitan el acceso y la manipulación de los datos desde Python.

Estos son solo algunos ejemplos de las aplicaciones de Pandas en diferentes contextos y problemas. Pandas es una biblioteca muy versátil y potente que ofrece muchas más funcionalidades y ventajas para el análisis y la manipulación de datos en Python.

Cuestionario guia de lectura del material

1. ¿Qué es numpy y para qué se utiliza?
2. ¿Qué ventajas tiene usar numpy sobre las listas de Python para el cálculo científico?
3. ¿Qué es un array de numpy y cómo se crea?
4. ¿Qué operaciones se pueden realizar con los arrays de numpy?
5. ¿Cuáles son las funciones universales de numpy y cómo se aplican a los arrays?

Bibliografía:

1. Bagnato, J. i., (2020). Aprende Machine Learning en Español: Teoría + Práctica Python. Editorial Leanpub.
2. Britos, P. V., & García Martínez, R. (2009). Propuesta de Procesos de Explotación de Información. In XV Congreso Argentino de Ciencias de la Computación.
3. Chazallet, S. (2016). Python 3: los fundamentos del lenguaje. Ediciones ENI.
4. Geron, A., (2020). Aprende Machine Learning con Scikit-Learn, Keras y Tensor Flow: Conceptos, herramientas y técnicas para construir sistemas inteligentes. Editorial O'Reilly y Anaya
5. Hilera, J. R. y Martinez, V. J. (2000) Redes Neuronales Artificiales. Fundamentos. modelos y aplicaciones. Alfaomega Ed
6. JIMÉNEZ, R. O., (2021). Python a fondo. Editorial Marcombo
7. Kuna, H. D., Caballero, S., Rambo, A., Meinl, E., Steinhilber, A., Pautsch, J., ... & Villatoro, F. (2010). Avance en procedimientos de la explotación de información para la identificación de datos faltantes, con ruido e inconsistentes. In XII Workshop de Investigadores en Ciencias de la Computación.
8. Kuna, H., Pautsch, G., Rey, M., Cuba, C., Rambo, A., ... & Villatoro, F. (2012). Obtenido de COMPARACIÓN DE LA EFECTIVIDAD DE PROCEDIMIENTOS DE LA EXPLOTACIÓN DE INFORMACIÓN PARA LA IDENTIFICACIÓN DE OUTLIERS EN BASES DE DATOS:
9. Matthes, E. (2021) Curso intensivo de Python, 2ª edición: Introducción práctica a la programación basada en proyectos. Editorial Anaya Multimedia
10. Ochoa, M. A. (2004). Herramientas inteligentes para explotación de información. Trabajo Final Especialidad en Ingeniería de Sistemas Expertos url: https://ri.itba.edu.ar/server/api/core/bitstreams/a848d640-0277-459d-9104-b37017309d31/content