Relatório: card 4- Principais Bibliotecas e Ferramentas Python para Aprendizado de Máquina (I)

Jefferson korte junior

Aqui estão meu conhecimentos adquiridos sobre a biblioteca numpy:

NumPy é uma biblioteca fundamental para computação científica em Python. Ela oferece suporte para arrays e matrizes multidimensionais, além de um monte de funções matemáticas de alto nível para operar nesses arrays. Numpy é muito bom quando vai se mexer com uma grande quantidade de dados ou operações matemáticas que vão acontecer.É amplamente utilizada em áreas como ciência de dados, aprendizado de máquina e análise numérica devido à sua eficiência e facilidade de uso.

Uma das primeiras coisas que aprendi nesse curso foi a usar o **np.array** que permite criar um objeto de array N-dimensional. Também aprendi sobre **np.arrays** que são mais eficientes em termos de memória e velocidade de processamento comparado com listas.

Sobre o **Shape** ele retorna uma tupla que indica o número de elementos em cada dimensão do array, em outras palavras ele diz como que os dados estão organizados dentro do array.

O **np.zeros** cria uma matriz apenas com números zeros do tamanho que quiser , o **np.ones** faz quase a mesma coisa, porém preenche a matriz com o número um em vez de zero.

Temos também a **matriz identidade** que é apenas com a diagonal principal com o número 1 o e resto dela sendo com números zeros. Para se usar também é bem fácil, apenas chamando o **np.identity()** e passando o tamanho dentro dos ().

np.empty cria uma matriz com o tamanho que queremos só que com valores aleatórios nela.

Temos também o **np.arange** que é usado para criar arrays com valores espaçados com intervalos específicos. **Exemplo:** np.arange([start,]stop, [step,] = array = np.arange(0, 10, 2), nesse exemplo mostrará os Valores de 0 a 10 com um intervalo de 2.

Parecido com o np.arange temos também o **np.linspace,** enquanto o np.arange usa o passo step para definir os intervalos,o **np.linspace** usa o **num** que define quantos números queremos nesse intervalo de tempo**.** Coisas que podemos fazer: np.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0), oque cada um desses parâmetros faz:

start: O valor inicial da sequência.
stop: O valor final da sequência.
num: (opcional) O número de valores a serem gerados (o padrão é 50).
endpoint: (opcional) Se True, o valor final stop é incluído; se False, ele é excluído (o padrão é True).

retstep: (opcional) Se True, retorna (samples, step) onde step é o espaçamento entre valores.
dtype: (opcional) O tipo de dados do array resultante. Se não especificado, será inferido a partir dos outros argumentos.
axis: (opcional) O eixo ao longo do qual os valores são gerados (o padrão é 0).

Sobre ver os tamanhos das array temos, **array.shape** = que retorna o número de linhas e colunas, temos o **np.size** = que retorna o número total de elementos no array, e também temos o **np.ndim** que retorna o número de dimensões do array.

Aprendi também como mudar o tamanho das arrays, nome_variavel = **array.reshape** ((2, 3)), (apenas um exemplo). Podemos trocar de forma a array só que a nova array terá que ser de um tamanho compatível com o número de elementos no array original.

Rankeando array usando o **np.sort** podemos ordenar a array.

np.newaxis é usado para aumentar a dimensionalidade da array, podendo fazer o **newaxis:** ou **:newaxis** que tem uma grande diferença onde se coloca o **:,** Quando você coloca **:** antes do **np.newaxis**, você está adicionando uma nova dimensão,transformando cada linha em uma coluna. quando se coloca **:** após o **np.newaxis**, você está adicionando uma nova dimensão transformando a linha em uma matriz de uma única linha com várias colunas.

Juntando arrays para se juntar array pode ser usado o **np.concatenate** e passando como parâmetro as arrays usando o (), que ficaria assim. c = np.concatenate ((a, b)). **Apenas um exemplo**.

Operadores com array:
np.sum() Calcula a soma de todos os elementos no array.
np.max () Encontra o valor máximo na array ou da onde foi especificado.
np.min () Encontra o menor valor ao longo da array ou da onde foi especificado.
np.mean() Calcula a média da array ou da onde foi especificado.
array.argmax vai mostrar o índice que está com o maior valor.
caso quiser somar array1 com array2, basta fazer array1 +,-,* ou / com array2 que irá pegar o primeiro índice da array1 com array2 e consecutivamente com todos os outros índices das arrays e irá fazer o cálculo pedido.
exponenciação, array ** 2 aqui estamos elevando cada elemento da array a potência 2.

Operações escalares, array +,-,*, % ou \, 100 assim ira multiplicar ou dividir ou fazer uma operação pedida por 100 e vai mudar todos os índices da array.

np.sqrt faz o cálculo da raiz quadrada.
np.std calcula o desvio o padrão dos elementos do array.
np.sin calcula o seno de cada elemento do array.

Gerando números aleatórios: Para se gerar números aleatórios precisa apenas usar o **np.random.rand** para números aleatórios fracionados e np.random.randint(10, size=(4, 8)) para números aleatórios inteiros.

O primeiro número que ali no exemplo é 10, quer dizer que só vai ter números aleatórios até 10, e o size é a dimensão.

Seleciona números aleatórios de dentro do array usando o **np.random.choice()** e passando a array dentro dos (). Podendo fazer a amostragem com repetição e sem repetição, quando fazemos com repetição significa que ele pode ser "escolhido novamente" e aparecer de novo na repetição. Para se usar :

array = [10, 20, 30, 40, 50]
aleatórios = **np.random.choice(array, size = 4, replace = true ou false** depende se quer com ou sem repetição.

np.random.seed é usado para definir a semente do gerador de números aleatórios. Definir uma semente garante que os números aleatórios gerados sejam os mesmos em cada execução do código.

Puxar elementos de um array, para puxar os elementos que quiser basta digitar: **array[1: 4, 0]**
O número 1 quer dizer que quero pegar da linha com índice um em diante
O número 3 quer dizer que vai ir até a linha com índice três, porém não vai mostrar essa linha.
E o zero quer dizer que vai pegar todas as colunas do índice zero em diante.

Puxando apenas números da array que quero, para isso pode se usar o:
bol = array > 50
resultado = array[bol]
print (resultado)
aqui neste caso só irá pegar os numeros de dentro da array os que são maiores que 50.

Alterando dados de um array, se colocar **array[2:] = 100,** após o indice dois todos os números serão 100.

Adicionando uma matriz até certo ponto a outra, array2 = array1[:3].copy() , a array2 vai receber todos os números só até a linha com índice 3. O bom de usar o copy() é caso quisermos mexer no valor de array2 não vai acabar mudando a array1.

Para medir o tempo de processamento, podemos usar o process.time(). vou colocar um exemplo:
import time
t1 = time.process_time()
aqui pode ter uma operação ou um loop de repetição.

t2 = time.process_time
tempo_execução = t2 - t1
isso era mostrar quanto tempo levou a operação.

Aqui estão meus conhecimentos adquiridos sobre a biblioteca pandas:

Sobre **Series** aprendi o que é uma e como se criar uma. Uma series é uma estrutura de dados só que unidimensional, é como se fosse uma coluna de um DataFrame, logo explicarei sobre DataFrame. Para se criar uma Series basta digitar o comando **pd.Series(data = ~~, index= ~~)**

Parâmetros:
data: Se refere aos dados da series ou do proprio DataFrame.
index: O index é um índice para identificar cada linha.

Para puxar elementos dessa Series: Series['A'] vai puxar o valor associado a A.

Criação de DataFrame: pd.DataFrame(Data = ~~, index= ~~, columns = ~~)
df = pd.dataFrame(np.random.rand(5,4)...) Apenas um exemplo. Aqui criei um DataFrame com números aleatórios

Padrao ao series o **data** se refere aos dados da tabela, o **index** se refere ao índice da tabela e o **columns** são os rótulos de cada coluna.

Puxando colunas de um DataFrame: Parecido até mesmo com o series o DataFrame se digitar o comando df['W'] vai puxar a coluna W do DataFrame.

Criando colunas novas no DataFrame: df['NEW'] = df['W'] + df['X']
Assim, irá criar uma coluna nova chamada NEW com os valores somados de W e X.

Deletando uma coluna: df.drop['NEW', axis = 1) Vai apagar a coluna NEW. Tem que colocar o axis como 1 para se referir às colunas, por padrão vem 0 e daí se refere às linhas. **Para excluir permanentemente tem que colocar ao final um inplace = True.**

Localizando elementos por índice: df.loc['A'] vai puxar a linha inteira.
Também podemos localizar elementos passando os indices. EX: df.iloc[1:4, 2]

Podemos verificar valores na tabela em booleano também:
bol = df > 0
df[bol] Vai aparecer a tabela com os números maiores ou que atendam a condição dada, e os que não atenderem vão aparecer como NAN.

Testes em linhas e colunas específicas: df[df['W'] > 0] vai voltar uma tabela com so os índices que atendam a condição dada. **Podemos fazer isso com mais de uma coluna também:**
df[(df['W'] > 0) & (df['Y'] > 5) assim só vai puxar os índices que atendem às duas condições
& se refere ao and.
|| se refere ao or.

Resetando o index: df.reset_index() vai resetar os índices e vão voltar para o padrão 0, 1, 2, 3…

Adicionando uma nova coluna: vamos supor que criei:
col = 'RS SP RJ AM SC'.split()
df['estados'] = col
Assim eu adicionei no meu DataFrame uma coluna chamada estados com os dados que estavam dentro de col.

Sobre o método Zip() aprendi que ele pode juntar duas listas ou duas tuplas, e podemos criar muita coisa a partir disso: Vamos a um exemplo:

inder_index = list(zip(lista1, lista2)) aqui criei uma lista com os dados de duas listas.

hier_index = pd.MultiIndex.from_tuplas(hier_index) Esse método do Pandas cria um índice hierárquico (MultiIndex) a partir de uma lista de tuplas. Podendo se criar um DataFrame com indices hierárquicos

MultiIndex: É um tipo especial de índice que permite que você trabalhe com múltiplos níveis de indexação. Isso é útil para operações de agrupamento, agregação e análise de dados complexos.

Acessando esses índices multiníveis:
df.loc['G1'] Aparecerá todo o conteúdo do índice G1, claro se o nome do meu índice multinivel for G1.
df.loc['g1'].loc[1] acessa dados em um DataFrame com índice hierárquico. seleciona todas as linhas onde o índice principal é 'g1'. Depois, .loc[1] acessa a linha específica onde o segundo nível do índice é 1 dentro desse subconjunto.

df.dropna() é usado para remover dados ausentes do dataFrame, ele limpa dados NAN de colunas ou linhas, dependendo do parâmetro colocado.

df.fillna() esse método é usado para preencher valores ausentes do DataFrame.
Principais Parâmetros:
value: Especifica o valor para substituir os NaNs.

method: Usa métodos de preenchimento como 'ffill' (preenchimento para frente) ou 'bfill' (preenchimento para trás).
axis: Define se o preenchimento é realizado ao longo das linhas (0) ou colunas (1).
inplace: Se True, modifica o DataFrame original.

df.groupby () é usado para agrupar dados em de um DataFrame, a partir disso podemos fazer operaçoes.
group.sum() vou ter um DataFrame das soma total do que pedi dentro dos ( ).
group.mean() vai voltar um DataFrame da media do que pedi dentro dos ( )
***group.describe()** esse comando volta um resumo doa dados com: count( ), mean, sum, max, min, STD, 25%, 50%, 75%.

Juntar, Mesclar, Concatenar:

pd.concat ([df1, df2, df3]) vai juntar os tres DataFrames em um só, Porém é bom os tres DataFrames ter o mesmo número de colunas.

pd.merge(esquerda, direita, how = 'left', 'right', 'outer', ou 'inner') é usada para combinar dois DataFrames.

esquerda.join(direita) aqui ele combina dos DataFrames pelos índices de cada um.

Operações que podemos fazer com pandas:
df[‘col2’]. unique Retorna uma array com os valores únicos da coluna, ignorando duplicatas.

df[‘col2’].nunique Retorna quantos elementos tem na coluna dois sem repetição.

df[‘col2’].value_counts() Conta e retorna o número de vezes que cada valor único aparece na coluna.

df[‘col2’].value_counts().head() Conta e retorna o número de vezes que cada valor único aparece na coluna, e com o head podemos delimitar quantos números queremos.

df[df[‘col1’] > 2] Retorna as linhas do DataFrame onde os valores da coluna 1 são maiores que 2.

df[‘col2’].apply(vezes2) Aqui eu apliquei uma função que já tinha criado a coluna2, vai aplicar a função a cada elemento da coluna2.

del df[‘col2’] Vai apagar a coluna dois.

df.coluns Vai mostrar o nome das colunas do DataFrame.

df.index Caso quiser saber os índices do DataFrame

Ordenando valores:
df.sort_values(by: ‘col2’) Vai ordenar os valores da coluna2. Se quiser que mude permanentemente tem que colocar o **inplace=True**.

df.isnull( ) retorna um booleano os valores nulos da tabela, com True e False.

df.pivot_table(values='d', index=['a', 'b'], columns=['c']) Cria uma tabela dinâmica a partir de um DataFrame. Ele permite agregar e reorganizar os dados.

Entrada e saída de dados:

pd.read_csv(dai passa o caminho do arquivo aqui dentro) É uma função usada para ler arquivos CSV e transformá-los em DataFrame.

df.to_CSV(“MeuArquivo.csv”) É usado para exportar um DataFrame para um arquivo CSV.

pd.read_excel (Passa o caminho do arquivo aqui.xlsx) Lê um arquivo excel e carrega em um DataFrame.

df.to_excel(Meuarquivo.xlsx) exportei o DataFrame para excel.

df.read_html(Caminho do arquivo) Le tabelas diretamente da pagina web e transforma elas em um dataFrame.

Conclusão: Durante o estudo deste card, aprofundei meus conhecimentos em Python e nas suas bibliotecas principais, especialmente numpy e Pandas. Aprendi a importar bibliotecas e a utilizar suas funcionalidades e parâmetros. Aprendi técnicas de manipulação de dados, criação de tabelas dinâmicas, fusão de DataFrames utilizando o método merge, e operações de leitura e escrita de arquivos CSV.

Referências :
(115) Como Sair do Zero com a Biblioteca Numpy no Python - YouTube

Python para Análise de dados - Pandas
Python para Análise de dados - Numpy