PANDAS - INTRO

Vedremo come usare PYTHON con PANDAS per gestire un DATASET.

Pandas è una libreria python per l’analisi dei dati e può essere installata tramite PIP o insieme ad anaconda (io l’ho installata tramite pip).

# IMPORTAZIONE E CARICAMENTO DATASET

import pandas as pd  
  
# la libreria pandas da l'opportunità di leggere un file (in vari formati)  
# per estrarre i dati molte volte espressi in tabelle o raggruppati in vari formati  
iris = pd.read\_csv*(*"/Users/massimilianoguida/Repo\_Corso\_DeepLearning/2 - Datasets e data preprocessing/data/iris.csv"*)*

Pandas ti da l’opportunità, appunto di caricare diversi formati di file, nel nostro caso usiamo un CSV. Il file sarà caricato in una speciali struttura dati di PANDAS chiamata DATAFRAME (tabulare e versatile).

## VISUALIZZAZIONE ed ACCESSO AI DATI

Per visualizzare i primi dati del dataset usiamo la funzione head, per visualizzare le ultime useremo la funzione tail.

# head prende le prime n righe della tabella considerata  
print*(*iris.head*(*10*))*# tail prende le ultime n righe della tabella considerata  
print*(*iris.tail*(*10*))*

Specificando un parametro le funzioni considereranno il numero di tuple specificate dal parametro, mentre se vuoto di default prenderà 5 valori.

Il risultato della funzione tail(10) sarà:

sepal\_length sepal\_width petal\_length petal\_width species

140 6.7 3.1 5.6 2.4 virginica

141 6.9 3.1 5.1 2.3 virginica

142 5.8 2.7 5.1 1.9 virginica

143 6.8 3.2 5.9 2.3 virginica

144 6.7 3.3 5.7 2.5 virginica

145 6.7 3.0 5.2 2.3 virginica

146 6.3 2.5 5.0 1.9 virginica

147 6.5 3.0 5.2 2.0 virginica

148 6.2 3.4 5.4 2.3 virginica

149 5.9 3.0 5.1 1.8 virginica

Dove la prima riga è relativa all’header della tabella.

Se non avessimo la riga degli header allora potremmo gestirla noi autonomamente inserendola

iris\_noheader = \  
 pd.read\_csv*(*"/Users/massimilianoguida/Repo\_Corso\_DeepLearning/2 - Datasets e data preprocessing/data/iris\_noheader.csv"  
 , header=None, names=*[*"sepal\_length", "sepal\_width", "petal\_length", "petal\_width", "species"*])*

Inoltre possiamo avere metainformazioni come il nome delle colonne appartenenti al dataframe:

iris.columns

oppure ad ulteriori informazioni generiche

ris.info*()*

si può accedere ai dati di una singola colonna

Y = iris*[*'species'*]*

In questo caso Y è una series che è un vettore che appartiene ad un DATAFRAME pandas.

Per dar luogo a sottoinsiemi di DATAFRAME rispetto a quello d’origine ci sono 2 modi.

1. Specificare le series necessarie

X = iris*[[*"sepal\_length", "sepal\_width", "petal\_length"*]]*

1. Filtrare le Series che non vogliamo visualizzare tramite la drop

Z = iris.drop*(*"species", axis=1*)*

AXIS è un parametron che appare in tutte quelle funzioni che operano su righe e/o colonne.

* AXIS = 1 🡪 quando l’operazione avviene sulle righe (come ad esempio nella drop, dove si va ad eliminare la colonna species da tutte le righe del dataframe
* AXIS = 0 🡪 quando l’operazione deve essere svolta per colonne

### SLICING

Lo slicing del dataset permette di selezionare solo alcune righe od alcune colonne. Per far ciò abbiamo due soluzioni diverse:

* Lock: lavora lui lable
* iloc: lavora sugli indici

Per capire meglio, si va a fare una copia del dataset e si assegna ad una variabile **iris\_sampled**

iris\_sampled = iris.copy*()*# mescoliamola utilizzando il metodo sample e l'attributo frac = 1 per tornare il dataset per intero (altrimenti tornerebbe un solo elemento  
iris\_sampled = iris\_sampled.sample*(*frac=1*)*print*(*iris\_sampled.head*())*

il risultato è il seguente (e cambierà per ogni esecuzione):

sepal\_length sepal\_width petal\_length petal\_width species

129 7.2 3.0 5.8 1.6 virginica

63 6.1 2.9 4.7 1.4 versicolor

13 4.3 3.0 1.1 0.1 setosa

28 5.2 3.4 1.4 0.2 setosa

66 5.6 3.0 4.5 1.5 versicolor

Data la definizione data in precedenza, quindi, utilizzando il comando iLoc[3] si va a richiedere l’elemento con indice 3 (selezionato in verde)

iris\_sampled.iloc*[*3*]*

per ottenere lo stesso risultato con loc avremmo dovuto specificare il numero dell’indice nella prima colonna d’interesse (quindi il label relativo a quella riga, **28**)

Se si vuol selezionare sia per riga sia per colonna sarà necessario specificare un secondo parametro, la colonna d’interesse.

print*(*"\n", iris\_sampled.loc*[*28, "species"*])*

si può selezionare, con iloc, le prime 10 righe e la colonna selezionata mediante l'indice di colonna

print*(*"\n", iris\_sampled.iloc*[*:10, 4*])*

9 setosa

88 versicolor

117 virginica

12 setosa

113 virginica

51 versicolor

92 versicolor

101 virginica

81 versicolor

13 setosa

Name: species, dtype: object

## OTTENERE ALCUNE INFO STATISTICHE

Con l’attributo **shape** otteniamo le info sulle dimensioni, si tratta di una tupla che ha come primo elemento la dimensione delle righe e come secondo elemento la dimensione delle colonne

iris.shape

(150, 5)

Utilizzando il metodo **describe** avremmo una serie di informazioni statistiche sul dataset:

* per ogni colonna il numero di valori presenti
* la media
* la deviazione standard
* la distribuzione
* il min e il max

iris.describe*()*

sepal\_length sepal\_width petal\_length petal\_width

count 150.000000 150.000000 150.000000 150.000000

mean 5.843333 3.054000 3.758667 1.198667

std 0.828066 0.433594 1.764420 0.763161

min 4.300000 2.000000 1.000000 0.100000

25% 5.100000 2.800000 1.600000 0.300000

50% 5.800000 3.000000 4.350000 1.300000

75% 6.400000 3.300000 5.100000 1.800000

max 7.900000 4.400000 6.900000 2.500000

queste info possiamo averle anche singolarmente, sia sull’intero dataset sia sulle singole colonne

iris*[*"sepal\_length"*]*.describe*()*

count 150.000000

mean 5.843333

std 0.828066

min 4.300000

25% 5.100000

50% 5.800000

75% 6.400000

max 7.900000

Name: sepal\_length, dtype: float64

Si può anche essere interessati ad un solo valore dell’insieme dei **describe** come ad esempio **max** o **mean** etc.

print*(*"\n", iris.max*())*

print*(*"\n", iris*[*"sepal\_length"*]*.max*())*

sepal\_length 7.9

sepal\_width 4.4

petal\_length 6.9

petal\_width 2.5

species virginica

dtype: object

7.9

Inoltre se vogliamo isolare una feature e farci ritornare la classe dei valori che possono appartenere a tale feature sarà neessario utilizzare il metodo **unique**. Questo metodo ritorna la lista di valori che, ad esempio, la feature “species” può assumere all’interno del dataset.

iris*[*'species'*]*.unique*()*

['setosa' 'versicolor' 'virginica']

### UTILIZZO DELLE MASCHERE

Un’altra funzionalità utilissima è la possibilità di applicare dei filtri usando delle maschere.

Per prima cosa capiamo come creare una maschera. Si tratta di un operazione molto semplice, ad esempio, se volessimo creare una maschera per selezionare le osservazioni che hanno la lunghezza del petalo maggiore della lunghezza media:

long\_petal\_mask = iris*[*"petal\_length"*]* > iris*[*"petal\_length"*]*.mean*()*print*(*"\n", long\_petal\_mask*)*

0 False

1 False

2 False

3 False

4 False

...

145 True

146 True

147 True

148 True

149 True

Come si può notare una maschera non è altro che un array contenente delle variabili booleane. Le osservazioni che contengono True in tale array verranno selezionate, mentre le altre verranno scartate.

Per applicare la maschera basta usarla come fosse un indice:

iris\_long\_petal = iris*[*long\_petal\_mask*]*print*(*iris\_long\_petal.head*(*150*))*

sepal\_length sepal\_width petal\_length petal\_width species

50 7.0 3.2 4.7 1.4 versicolor

51 6.4 3.2 4.5 1.5 versicolor

52 6.9 3.1 4.9 1.5 versicolor

53 5.5 2.3 4.0 1.3 versicolor

54 6.5 2.8 4.6 1.5 versicolor

.. ... ... ... ... ...

145 6.7 3.0 5.2 2.3 virginica

146 6.3 2.5 5.0 1.9 virginica

147 6.5 3.0 5.2 2.0 virginica

148 6.2 3.4 5.4 2.3 virginica

149 5.9 3.0 5.1 1.8 virginica

[93 rows x 5 columns]

Possiamo utilizzare una maschera insieme a loc per modificare le nostre osservazioni in base a delle condizioni. Ad esempio si crei una maschera che ci permetta di cambiare il valore della specie da “setosa” a “undefined”:

iris\_copy = iris.copy*()*# si noti che viene applicata la maschera allo stesso dataframe  
iris\_copy*[*iris\_copy*[*"species"*]* == "setosa"*]* = "undefined"  
print*(*iris\_copy*[*"species"*]*.unique*())*

['undefined' 'versicolor' 'virginica']

### OPERAZIONI ARITMETICHE NEL DATAFRAME

È molto semplice eseguire delle operazioni aritmetiche all’interno del nostro dataframe.

Proviamo ad esempio ad eseguire una normalizzazione delle features numeriche:

Per prima cosa escludiamo la colonna delle specie perché non numerica:

X = iris.drop*(*"species", axis=1*)*

La normalizzazione verrà eseguita su tutte le colonne senza bisogno di iterare il calcolo per ogni elemento:

Xnorm = *(*X - X.min*())* / *(*X.max*()* - X.min*())*print*(*Xnorm.head*())*

sepal\_length sepal\_width petal\_length petal\_width

0 0.222222 0.625000 0.067797 0.041667

1 0.166667 0.416667 0.067797 0.041667

2 0.111111 0.500000 0.050847 0.041667

3 0.083333 0.458333 0.084746 0.041667

4 0.194444 0.666667 0.067797 0.041667

Sono possibili anche operazioni di ordinamento tramite il metodo **sort\_values** a bui si dovrà passare il parametro relativo alla colonna da ordinare (di default in ordine crescente, usare il parametro **ascending = False** per un ordinamento decrescente):

print*(*"\n", iris.sort\_values*(*"petal\_length", ascending=False*)*.head*(*10*))*

print*(*"\n", iris.sort\_values*(*"petal\_length"*)*.head*(*10*))*

Si può operare un raggruppamento mediante la **groupby**:

# Raggruppamento mediante la groupby e stampa della media degli altri valori per ogni specie  
grouped\_species = iris.groupby*(*"species"*)*print*(*"\n", grouped\_species.mean*())*

sepal\_length sepal\_width petal\_length petal\_width

species

setosa 5.006 3.418 1.464 0.244

versicolor 5.936 2.770 4.260 1.326

virginica 6.588 2.974 5.552 2.026

### APPLICARE FUNZIONI SU RIGHE O COLONNE

Utilizziamo il metodo **apply** passando come primo argomento il nome della funzione da applicare, che può essere una funzione python, una funzione definita da noi oppure una funzione di una libreria esterna da applicare.

Prendiamo ad esempio la libreria **numpy** ed applichiamo il metodo **count\_nonzero** per contare i valori diversi da 0 per ogni riga

# ad esempio applichiamo la funzione numpy per contare valori differenti da zero per riga  
# il risultato è una series quindi possiamo ricavarne i primi valori grazie alla funzione head  
print*(*"\n", iris.apply*(*np.count\_nonzero, axis=1*)*.head*(*10*))*

0 5

1 5

2 5

3 5

4 5

5 5

6 5

7 5

8 5

9 5

dtype: int64

qualora vorremmo contare i valori per colonna dovremmo cambiare **axis = 0**

print*(*"\n", iris.apply*(*np.count\_nonzero, axis=0*)*.head*(*10*))*

sepal\_length 150

sepal\_width 150

petal\_length 150

petal\_width 150

species 150

dtype: int64

Possiamo applicare anche una funzione valore per valore utilizzando il metodo **applymap** ad esempio proviamo ad arrotondare i valori del dataframe all’intero più vicino:

# proviamo ad arrotondare i valori del dataframe all'intero più vicino  
# per prima cosa leviamo species in quanto non è numerico  
X = iris.drop*(*"species", axis=1*)*# poi utilizziamo applymap usando una funzione lamda (funzione usa e getta per essere utilizzata una volta  
# solo quando il cursore ci passa sopra (rivedere le funzioni lamda)  
X = X.applymap*(*lambda val: int*(*round*(*val, 0*)))*print*(*X.head*())*

sepal\_length sepal\_width petal\_length petal\_width

0 5 4 1 0

1 5 3 1 0

2 5 3 1 0

3 5 3 2 0

4 5 4 1 0

### VALORI INVALIDI NEL DATASET

Si può risolvere usando il metodo **fillna**. Per testare questo esempio andiamo a creare valori non validi:

# per testare fillna si vanno ad inserire valori invalidi  
# per prima cosa si usa una copia del dataframe  
iris\_nan = iris.copy*()*# creiamo un vettore di 10 valori casuali interi  
sample = np.random.randint*(*iris.shape*[*0*]*, size=10*)*# usiamo loc per modificare in None la la feature in corrispondenza di queste 10 osservazioni  
iris\_nan.loc*[*sample, "petal\_length"*]* = None  
# per verificare se la modifica è andata a buon fine andiamo a contare il numero di  
# valori null all'interno di iris\_nan in corrispondenza della feature petal\_length  
print*(*"\n", iris\_nan*[*"petal\_length"*]*.isnull*()*.sum*())*# ora possiamo applicare FILLNA  
# sostituiamo questi valori invalidi con il valore medio della lunghezza del petalo  
# calcolo valore medio  
mean\_petal\_length = iris\_nan*[*"petal\_length"*]*.mean*()*print*(*"\n", mean\_petal\_length*)*# possiamo applicare fillna. Questa istruzione fa tornare una nuova series  
# e non verrà modificato il valore della colonna petal\_length  
# per farlo o eseguiamo un assegnazione  
# iris\_nan["petal\_length"] = iris\_nan["petal\_length"].fillna(mean\_petal\_length)  
# oppure ancora potremmo usare il parametro inplace  
iris\_nan*[*"petal\_length"*]*.fillna*(*mean\_petal\_length, inplace=True*)*# eseguendo il tutto andiamo a controllare i valori nulli dopo il fillna (dovrebbero essere 0)  
print*(*"\n", iris\_nan*[*"petal\_length"*]*.isnull*()*.sum*())*

### INTERFACCIAMENTO CON MATPLOT LIB

Per prima cosa installiamo il package matplotlib (pycharm identifica la dipendenza e si propone di installarlo). Poi cerchiamo di fare un grafico per visualizzare la relazione del sepalo e la larghezza.

import matplotlib.pyplot as plt  
  
iris.plot*(*x="sepal\_length", y="sepal\_width", kind="scatter"*)*# show su plt mostra il plot creato  
plt.show*()*