Marziani - Parte 3 (Pandas e Matplotlib + filtri dati)

```
# Pandas = Manipolazione / analisi dei dati da file
# Matplotlib (Analisi dati -> Calcolo numerico / Analisi numerica)
# Es. calcolo errori in varie situazioni e visualizzazione plot

import pandas as pd
import matplotlib.pyplot as plt

## 1. Lettura del dato / file (pulito)

marziani = pd.read_csv("marziani.csv")
# CSV = Comma Separated Values
# valore1, valore2, .....
# Formato veloce soprattutto esatto per Python/AI

# Sintassi: header, delimiter, rows...
## |specie|colore|n_arti|peso|altezza|larghezza|

# dropna = Ripulisco tutti i valori nulli (mancanti)
# DataFrame = Formato di pandas per leggere le tuple
marziani_clean = marziani.dropna()
marziani_clean.info() # Vis. contenuto

# 2. Salvataggio su file

marziani_clean.to_excel("marziani.xlsx", sheet_name="Marziani")
# Altri formati di salvataggio disponibili
# marziani_ripulito.to_csv("marziani_pulito.csv")
# marziani_ripulito.to_json("marziani.json")
# marziani_ripulito.to_pickle("marziani.pkl")

# 3. Filtro dei singoli valori [subscripting o funzioni]

# Filtro semplice: marziani con più di 27 arti
marziani_con_molti_arti = marziani_clean[marziani_ripulito["n_arti"] >
27]
print(marziani_con_molti_arti)

# Conteggio: quanti marziani hanno più di 28 arti?
num_marziani_28_arti = len(marziani_ripulito[marziani_ripulito["n_arti"] >
28])
print(f"Numero di marziani con più di 28 arti: {num_marziani_28_arti}")

# Metodi classici aggregati -> min() / max / mean()
```

```python
# Calcoliamo prima le medie
media_peso = marziani_ripulito["peso"].mean()
media_altezza = marziani_ripulito["altezza"].mean()

# Applichiamo il filtro combinato
marziani_filtrati = marziani_clean[
(marziani_ripulito["specie"] == "Robby") &
(marziani_ripulito["colore"] == "blu") &
((marziani_ripulito["peso"] > media_peso) |
(marziani_ripulito["altezza"] > media_altezza))
]
print(f"Numero di Robby blu con peso o altezza superiore alla media:
{len(marziani_filtrati)}")

## Statistica: min / max / mean / deviazione standard (gaussiana) ->
normalmente se va bene o male / le singole parti (quartili)

# Massimo e minimo del numero di arti
max_arti = marziani_clean["n_arti"].max()
min_arti = marziani_clean["n_arti"].min()
print(f"Numero massimo di arti: {max_arti}")
print(f"Numero minimo di arti: {min_arti}")

# Altre statistiche utili
# Media
media_peso = marziani_clean["peso"].mean()
# Mediana
mediana_peso = marziani_clean["peso"].median()
# Deviazione standard
std_peso = marziani_clean["peso"].std()
# Quartili (1/4, 2/4, 3/4)
quartili_peso = marziani_ripulito["peso"].quantile([0.25, 0.5, 0.75])

print(f"Statistiche per il peso:")
print(f"Media: {media_peso:.2f}")
print(f"Mediana: {mediana_peso:.2f}")
print(f"Deviazione standard: {std_peso:.2f}")
print(f"Quartili: {quartili_peso.to_dict()}"

# Creare un nuovo file csv e prendere soltanto le colonne soltanto con gli
arti che siano < 21 E > 25

marziani = pd.read_csv("marziani.csv")
marziani_clean = marziani.dropna()
marziani_clean.info()

# Crea il nuovo file
marziani_filtered = marziani_clean.to_csv("marziani_filtered.csv",
sheet_name="Marziani")
```

```
# Applichiamo il filtro combinato
marziani_filtrati = marziani_ripulito[
(marziani_ripulito["specie"] == "Robby") &
(marziani_ripulito["colore"] == "blu") &
((marziani_ripulito["peso"] > media_peso) |
(marziani_ripulito["altezza"] > media_altezza))
]

# Prendiamo il file applicando un filtro come condizione (nota: non uso le
quadre perché non lo sto applicando direttamente alle tuple (campi) del
file)
marziani_filtered =
len(marziani_ripulito[marziani_ripulito["n_arti"] <
21]) & len(marziani_ripulito[marziani_ripulito["n_arti"] >
25])

# Dare una struttura - secondo specie / colore, etc.
aggregazione = marziani_filtered.groupby(["specie", "colore"]).agg({
"n_arti": ["mean", "max", "min"],
"peso": ["mean", "median", "std"],
"altezza": ["mean", "max"]
})
print(aggregazione)

# Visualizzazione grafico con sottografici (subplot) per specie
fig, ax = plt.subplots(figsize=(10, 8)) #figure and axis

# Barre = Scatter

# Plot per i Robby (in rosso)
marziani_clean[marziani_filtered["specie"] == "Robby"].plot.scatter(
x="peso",
y="larghezza",
c="r", # c = color
label="Robby",
ax=ax # asse passato in parametro
)

# Plot per i Robby (in rosso)
marziani_clean[marziani_filtered["specie"] == "Robby"].plot.scatter(
x="peso",
y="larghezza",
c="g",
label="Simmy",
ax=ax
)

# Personalizzazione del grafico
ax.set_xlabel("Peso", fontsize=16)
ax.set_ylabel("Larghezza", fontsize=16)
```

```
ax.tick_params(axis='x', labelsize=14) # Parametri assi
ax.tick_params(axis='y', labelsize=14)
ax.set_title("Relazione tra Peso e Larghezza dei Marziani", fontsize=20)
ax.grid(True)
ax.legend(fontsize=16)

# Mostra il grafico
plt.tight_layout() # Do un layout al container
plt.show() # Mostrare il plot
```

Marziani - Parte 4 (Scikit-learn e dataset)

```
# ` = backtick = ALT + 96

# Scikit = Libreria classica per lettura e il calcolo di probabilità
# Classificazione -> Prendo un dataset e cerco di capire come impostare
l'addestramento
# Regressione = Se sto sbagliando devo capire in modo efficiente come
calcolare i passi precedenti (polinomio)
# Clustering = Raggruppamento di dati in forma ordinata
# SVM = Support Vector Machina = Forma matematica lineare (meno complessità)
per trovare in media (valori attesi = bayes) i valori migliori (ogni k
valori= k-mean)

# Dataset in una forma ordinata
# Dizionario {[key1 | value1] - [key2 | value2]... n}

from sklearn import datasets
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Scikit = Selezioniamo alcune componenti
# 1. Modello di selezione per training
# 2. Classificatore (usando dei metodi di regressioni e secondo certi
ragionamenti)
# 3. Metriche (accuratezza training)

from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from joblib import dump, load

# Dump = Prendi più dati possibile
# Load = Carica tutti i dati

# 1. Lettura dati usando pandas
```

```python
marziani = 'marziani.csv'
data = pd.read_csv(nomefile)
print(data.head())
print(">>colonne: ", data.columns)
print(">>tipi\n", data.dtypes)

# 2. Scraping (Filtro dati capendo come organizzare le righe o colonne)

# Verificare quante specie ci sono e quanti campioni
print(">>Specie")
print(data['specie'].unique())
print(">>Describe")
print(data['specie'].describe())
# Filtri iniziali per campi univoci per "migliorare" la probabilità di
successo

# Qui sopra mostra che ci sono due specie (Simmy e Robby) presenti in
proporzioni
# uguali nel dataset

# Statistiche per ogni caratteristica, divise per specie
for specie in data.specie.unique():
    dati = data[data['specie'] == specie]
    print('>>', specie)
    for x in data.columns[1:]:
        print(dati[x].describe())

# Cominci a fare dei filtri -> I Simmy (1) sono più pesanti in media dei
Robby, il colore diffuso è il blu etc.
# Natural Language Processing (NLP) = usa il linguaggio naturale per capire
come discriminare i dati a seconda del contesto

# 3. Normalizzazione (pulizia dei valori mancanti)
# e selezione delle feature e del target
# Features = Insieme dei dati di input
# Target = Insieme di output

# Conversione dei colori da categorici a numerici (in forma mappa /
dizionario per avere tutto categorizzato)
colori = np.sort(data['colore'].dropna().unique())
print(colori)
d = data.copy()
mapping = {'blu': 0, 'rosso': 1, 'viola': 2}
d['colore'] = d['colore'].map(mapping)

# Visualizzazione della correlazione tra caratteristiche
sns.set_theme(font_scale=2)
sns.pairplot(d, diag_kind="hist", hue='specie', dropna=True)
sns.set()
```

```
# Identificare le colonne con dati mancanti
cols_with_missing = [col for col in d.columns if d[col].isnull().sum()]
print(cols_with_missing)
# Selezione delle caratteristiche più rilevanti
cols_selected = ['peso', 'altezza', 'larghezza']
# Rimuovere le righe con dati mancanti solo nelle colonne selezionate
d = data.dropna(subset=cols_selected)
print(d.shape)
# Controllo delle colonne rimanenti con dati mancanti
print([col for col in d.columns if d[col].isnull().sum()])


### Features e addestramento ###

X = d[cols_selected] # Dati di input (features)
y = d['specie'] # Output desiderato (target)
print(X.head())
print(y.head())

# Divisione del dataset in training (70%) e test (30%)
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7,
random_state=0)
print(X_train.head())
print("Numero di campioni in X train: ", X_train.shape[0])
print(y_train.value_counts())
print(y_train.head())

## Visualizziamo dati comprendendo
# - 1. l'allontanamento dalla media (punta della curva gaussiana)
# - 2. l'allontanamento dalle code (deviazione standard)
# - 3. convergenza dell'algoritmo di addestramento

# Normalizzazione = Media -> 0, Deviazione standard = 1, Distribuzione dei
dati -> Preservazione dati in modo efficiente

# Calcolo di media e deviazione standard dal training set
mean = X_train.mean()
std = X_train.std()
# Standardizzazione (z-score normalization)
X_train_std = ((X_train-mean)/std)
X_test_std = ((X_test-mean)/std)

print(f">>X train Normalizzato \n {X_train_std.describe()}")
# Score = Allontanamento dai parametri statistici

## Modello MLP (Multi-Layer Perceptron)
# Rete neurale = Emula il neurone umano "Matematicamente" -> percettrone
# Percettrone = Input / Output (funzione)
# Sormonto vari strati di iterazioni per l'addestramento "al miglior valore
```

```
possibile"

# Feed-forward = Input espliciti e andando avanti rifiniamo l'analisi con
degli input nascosti (numeri casuali per cercare di continuare a riprodurre
un'analisi buona)

# Creazione del modello MLP (Multi-Layer Perceptron) usando un numero di
strati con un certo numero di neuroni (per ogni funzione/neurone,
utilizziamo un certo numero di iterazioni e un valore fisso per garantire
riproducibilità)
model = MLPClassifier(hidden_layer_sizes=(100,100), random_state=1,
max_iter=300)
# Addestramento del modello
model.fit(X_train_std, y_train)

# 4. Valutaizone modello
# Aka: cerca di capire come si comporta il modello "prevedendo"
(allontanandosi dai parametri statistici) quanto si comporti "bene" il
modello
# Test del modello
print("Train")
print(y_train.values[:5])
print(model.predict(X_train_std[:5]))

print("Test")
print(y_test.values[:5])
print(model.predict(X_test_std[:5]))

# Predizione su un nuovo caso
caso = [4.8, 31.4, 70.8]
caso_std = ((caso - mean) / std)
print("Caso")
print(model.predict([caso_std]))

# Calcolo dell'accuratezza (allontanamento dei campioni di test rispetto
allo score (generale))
train_score = model.score(X_train_std, y_train)
print(f"Accuratezza dati di TRAIN: {round(train_score, 3)}")
test_score = model.score(X_test_std, y_test)
print(f"Accuratezza dati di TEST: {round(test_score, 3)}")

# Overfitting = Modello usa troppi parametri rispetto al numero di
osservazioni
# Graficamente capisci che buttando vari dati, in media hai la stessa curva
# Ogni volta il campionamento cerca di approssimare campioni ad un andamento
regolare che non sia "sempre campionabile" nella stessa maniera

# Epoche = Numero totale di iterazioni
# Movimento della "pendenza" per capire il tempo = gradiente
```

```
# Monitorare l'accuratezza in funzione del numero di epoche
start = 100
stop = 300
passo = 10

vEpochs = np.arange(start, stop, passo)
vAccTrain = []
vAccTest = []

for e in vEpochs:
        mlp = MLPClassifier(hidden_layer_sizes=(100,100), random_state=1,
max_iter=e)
    mlp.fit(X_train_std, y_train)
    vAccTrain.append(mlp.score(X_train_std, y_train))
    vAccTest.append(mlp.score(X_test_std, y_test))

# Grafico dell'accuratezza
plt.plot(vEpochs, vAccTrain, c='b', label='Train')
plt.plot(vEpochs, vAccTest, c='g', label='Test')
plt.legend()

# Overfitting
# Compromesso tra bias (pregiudizio: dati siano "puri" da un punto di vista
di valutazione e varianza (allontanamento al quadrato rispetto al valore
atteso) )
```

MLP - MNIST

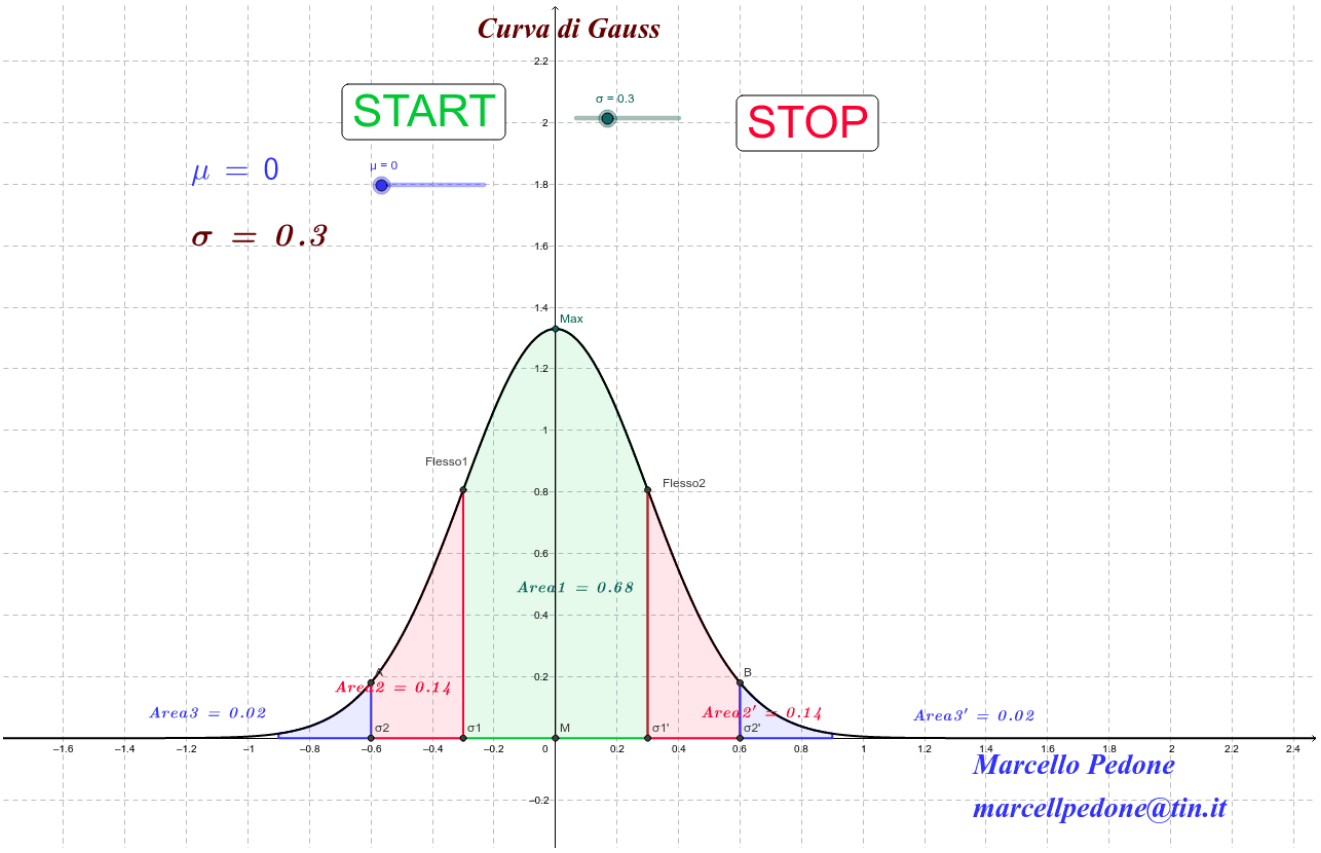
Un Multi-layer Perceptron è un tipo di rete neurale feed-forward composta da almeno tre strati:

- Ogni neurone in un MLP utilizza una funzione di attivazione non lineare (come ReLU, Sigmoid o Tanh) per trasformare i dati in ingresso e passarli allo strato successivo

MNIST: Dataset per immagini di cifre scritte a mano 0 e 9) utilizzato per addestramento pixel o di singoli dati dentro i DB.

Si rifà al codice precedente con qualche accorgimento (vedi appunti...)

Immagine gaussiana (curva normale standard)


Curva di Gauss
START
STOP
σ = 0.3
μ = 0
μ = 0
σ = 0.3
Max
Flesso1
Flesso2
Area1 = 0.68
Area2 = 0.14
Area2' = 0.14
Area3 = 0.02
Area3' = 0.02
A
B
M
σ2
σ1
σ1'
σ2'
Marcello Pedone
marcellpedone@tin.it