Roberto Marmo

Algoritmi per l'intelligenza artificiale

Progettazione,
Machine Learning,
Neural Network,
Deep Learning,
ChatGPT, Python

SECONDA EDIZIONE

HOEPLI

ROBERTO MARMO

Algoritmi per l'intelligenza artificiale

Progettazione, Machine Learning, Neural Network, Deep Learning, ChatGPT, Python

SECONDA EDIZIONE

EDITORE ULRICO HOEPLI MILANO

```
"""CRLF
Codice Python presentato nel libro "Algoritmi per l'Intelligenza Artificiale" CRLF
scritto da Roberto Marmo per editore Hoepli CRLF
sito web https://www.algoritmiia.it/libro/  email ia@robertomarmo.net CRLF
CRLF
aggiornamento 30 agosto 2024 CRLF
CRLF
versione ridotta per motivi di copyright, per avere la versione intera CRLF
scrivere email a ia@robertomarmo.net CRLF
CRLF
CONVIENE cambiare il tipo dell'estensione di questo file da txt a py per avere CRLF
come nome di file Algoritmi_intelligenza_artificiale_Marmo_Hoepli_2_CODICE.py CRLF
"""CRLF
CRLF
CRLF
CRLF
#-------------------------------------------------------------------------------
----CRLF
#Capitolo 5 Evolutionary algorithm CRLF
# scarica da repository GitHub CRLF
!git clone https://github.com/DEAP/deap.git CRLF
# installa libreria CRLF
!pip install deap CRLF
# importare librerie da usare CRLF
import random CRLF
from deap import base, creator, tools CRLF
CRLF
creator.create("FitnessMax", base.Fitness, weights=(1.0,)) CRLF
creator.create("Individual", list, fitness=creator.FitnessMax) CRLF
CRLF
CRLF
toolbox = base.Toolbox() CRLF
toolbox.register("attr_bool", random.randint, 0, 1) CRLF
toolbox.register("individual", tools.initRepeat, creator.Individual, CRLF
    toolbox.attr_bool, 10) CRLF
toolbox.register("population", tools.initRepeat, list, toolbox.individual) CRLF
CRLF
def evalOneMax(individual): CRLF
    return sum(individual), CRLF
toolbox.register("evaluate", evalOneMax) CRLF
toolbox.register("mate", tools.cxTwoPoint) CRLF
toolbox.register("mutate", tools.mutFlipBit, indpb=0.05) CRLF
toolbox.register("select", tools.selTournament, tournsize=3) CRLF
CRLF
def main(): CRLF
    random.seed(64) CRLF
CRLF
    n=50 CRLF
    pop = toolbox.population(n) CRLF
CRLF
    CXPB, MUTPB = 0.5, 0.2 CRLF
    CRLF
    fitnesses = list(map(toolbox.evaluate, pop)) CRLF
    for ind, fit in zip(pop, fitnesses): CRLF
        ind.fitness.values = fit CRLF
    CRLF
    print("  Valutazione %i individui" % len(pop)) CRLF
CRLF
    fits = [ind.fitness.values[0] for ind in pop] CRLF
    g = 0 CRLF
CRLF
    print("Gen.  non-valide Min Mas Media Std \n")    CRLF
    while max(fits) < 10 and g < 100: CRLF
CRLF
        g = g + 1 CRLF
CRLF
        offspring = toolbox.select(pop, len(pop)) CRLF
        offspring = list(map(toolbox.clone, offspring)) CRLF
    CRLF
        for child1, child2 in zip(offspring[::2], offspring[1::2]): CRLF
CRLF
            if random.random() < CXPB: CRLF
```

```
                toolbox.mate(child1, child2)
CRLF
                del child1.fitness.values
                del child2.fitness.values
CRLF
        for mutant in offspring:
CRLF
            if random.random() < MUTPB:
                toolbox.mutate(mutant)
                del mutant.fitness.values
    CRLF
        invalid_ind = [ind for ind in offspring if not ind.fitness.valid]
        fitnesses = map(toolbox.evaluate, invalid_ind)
        for ind, fit in zip(invalid_ind, fitnesses):
            ind.fitness.values = fit
CRLF
        pop[:] = offspring
        CRLF
        fits = [ind.fitness.values[0] for ind in pop]
        CRLF
        length = len(pop)
        mean = sum(fits) / length
        sum2 = sum(x*x for x in fits)
        std = abs(sum2 / length - mean**2)**0.5
CRLF
        print("%4d %3d %2d %2d %0.3f %0.3f" % (g, len(invalid_ind), min(fits), max(
        fits), mean, std))
CRLF
    print("-- Fine dell'evoluzione --")
    CRLF
    best_ind = tools.selBest(pop, 1)[0]
    print("Individuo %s con prestazione migliore %s" % (best_ind, best_ind.fitness.
    values))
CRLF
CRLF
#per fare eseguire il codice
if __name__ == "__main__":
    main()
CRLF
CRLF
#esercizio calcolo di x^2 in [0,31] per il tipo di intervallo chiesto bastano 5 bit
CRLF
CRLF
# scarica da repository GitHub
!git clone https://github.com/DEAP/deap.git
# installa libreria
!pip install deap
# importare librerie da usare
import random
from deap import base, creator, tools
CRLF
creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", list, fitness=creator.FitnessMax)
CRLF
CRLF
toolbox = base.Toolbox()
toolbox.register("attr_bool", random.randint, 0, 1)
toolbox.register("individual", tools.initRepeat, creator.Individual,
    toolbox.attr_bool, 5)  #essendo [0,31] intervallo di ricerca bastano 5 bit per
    ogni individuo
toolbox.register("population", tools.initRepeat, list, toolbox.individual)
CRLF
def evalOneMax(individual):
    def to_int(b):
        return int(b, 2)
    val  = to_int(''.join((str(xi) for xi in individual))) #da bit a numero intero da
    passare in formula
    return val*val,  #qui si calcola la formula
toolbox.register("evaluate", evalOneMax)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)
toolbox.register("select", tools.selTournament, tournsize=3)
```

```
CRLF
CRLF
def main():CRLF
    random.seed(4) #ci sono pochi bit per ogni individuo, abbasso il seme per generare numeri casualiCRLF
CRLF
    n=4 #numero individui da valutare, provare ad aumentare o diminuireCRLF
    pop = toolbox.population(n)CRLF
CRLF
    CXPB, MUTPB = 0.5, 0.2CRLF
    CRLF
    fitnesses = list(map(toolbox.evaluate, pop))CRLF
    for ind, fit in zip(pop, fitnesses):CRLF
        ind.fitness.values = fitCRLF
    CRLF
    print("  Valutazione %i individui" % len(pop))CRLF
CRLF
    fits = [ind.fitness.values[0] for ind in pop]CRLF
    g = 0CRLF
CRLF
    print("Gen.  non-valide Min Mas Media Std \n")    CRLF
    while max(fits) < 960 and g < 100: #1000 perché [0,31] con 31*31=961CRLF
CRLF
        g = g + 1CRLF
CRLF
        offspring = toolbox.select(pop, len(pop))CRLF
        offspring = list(map(toolbox.clone, offspring))CRLF
    CRLF
        for child1, child2 in zip(offspring[::2], offspring[1::2]):CRLF
CRLF
            if random.random() < CXPB:CRLF
                toolbox.mate(child1, child2)CRLF
CRLF
                del child1.fitness.valuesCRLF
                del child2.fitness.valuesCRLF
CRLF
        for mutant in offspring:CRLF
CRLF
            if random.random() < MUTPB:CRLF
                toolbox.mutate(mutant)CRLF
                del mutant.fitness.valuesCRLF
    CRLF
        invalid_ind = [ind for ind in offspring if not ind.fitness.valid]CRLF
        fitnesses = map(toolbox.evaluate, invalid_ind)CRLF
        for ind, fit in zip(invalid_ind, fitnesses):CRLF
            ind.fitness.values = fitCRLF
CRLF
        pop[:] = offspringCRLF
        CRLF
        fits = [ind.fitness.values[0] for ind in pop]CRLF
        CRLF
        length = len(pop)CRLF
        mean = sum(fits) / lengthCRLF
        sum2 = sum(x*x for x in fits)CRLF
        std = abs(sum2 / length - mean**2)**0.5CRLF
CRLF
        print("%4d %3d %2d %2d %0.3f %0.3f" % (g, len(invalid_ind), min(fits), max(fits), mean, std))CRLF
CRLF
    print("-- Fine dell'evoluzione --")CRLF
    CRLF
    best_ind = tools.selBest(pop, 1)[0]CRLF
    print("Individuo %s con prestazione migliore  %s" % (best_ind, best_ind.fitness.values))CRLF
CRLF
CRLF
#per fare eseguire il codiceCRLF
if __name__ == "__main__":CRLF
    main()CRLF
CRLF
CRLF
CRLF
```

```
#-------------------------------------------------------------------------------
----CRLF
# ATTENZIONE il modulo experta funziona fino alla versione 3.9 di Python CRLF
# al momento della pubblicazione del codice non è disponibile la versione per Python
3.10CRLF
# quindi su Google Colab con versione 3.10 non funziona CRLF
CRLF
#CAPITOLO 6 Expert systemCRLF
!pip install expertaCRLF
CRLF
from random import choice #per generare scelte casualiCRLF
from experta import * #per realizzare il sistema espertoCRLF
CRLF
class Luce(Fact):CRLF
    """Informazione sulla luce del semaforo"""CRLF
    passCRLF
CRLF
class Semaforo(KnowledgeEngine):CRLF
#definizione delle regole di calcoloCRLF
    @Rule(Luce(colore='verde'))CRLF
    def luce_verde(self):CRLF
        print ("Passare")CRLF
CRLF
    @Rule(Luce(colore='rosso'))CRLF
    def luce_rossa(self):CRLF
        print ("Non passare") CRLF
CRLF
    @Rule(AS.Luce << Luce(colore=L('giallo') | L('giallo lampeggia')))CRLF
    def attenzione(self, Luce):CRLF
        print ("Attenzione al ", Luce["colore"]) CRLF
CRLF
engine = Semaforo() #scelta del sistema espertoCRLF
engine.reset()CRLF
engine.declare(Luce(colore=choice(['verde', 'giallo', 'giallo lampeggia', 'rosso'])))
#si sceglie un colore a caso, il sistema esperto dice la risposta con le regole
impostateCRLF
engine.run()CRLF
CRLF
CRLF
from experta import *CRLF
CRLF
class Persona(Fact):CRLF
    """Informazioni sulla persona da analizzare"""CRLF
    passCRLF
CRLF
——>#per contare quanti sintomi reali bisogna considerareCRLF
def conteggio(p, *fields):CRLF
    return sum([p.get(x, 0) for x in fields])CRLF
CRLF
class AnalisiDiabete(KnowledgeEngine):CRLF
    @Rule(Persona(eta=MATCH.eta))CRLF
    def persona_maggiorenne(self, eta):CRLF
        if eta >= 18:CRLF
            self.declare(Fact(maggiorenne=True))CRLF
        else:CRLF
            self.declare(Fact(maggiorenne=False))CRLF
            print("ATTENZIONE per i minorenni usare un altro sistema")CRLF
CRLF
    @Rule(Fact(maggiorenne=True), Persona(glicemia=MATCH.glicemia))CRLF
    def iper_glicemia(self, glicemia):CRLF
        if glicemia > 110:CRLF
            self.declare(Fact(rischio_iperglicemia=True))CRLF
            print("Glicemia sopra la soglia")CRLF
        else:CRLF
            self.declare(Fact(rischio_iperglicemia=False))CRLF
CRLF
    @Rule(Fact(maggiorenne=True), Persona(glicemia=MATCH.glicemia))CRLF
    def normo_glicemia(self, glicemia):CRLF
        if (glicemia > 70) and (glicemia < 110):CRLF
            print("Glicemia nella norma")CRLF
            self.declare(Fact(normo_glicemia=True))CRLF
        else:CRLF
```

```
276                 self.declare(Fact(normo_glicemia=False))CRLF
277 CRLF
278     @Rule(Fact(maggiorenne=True), Persona(glicemia=MATCH.glicemia))CRLF
279     def ipo_glicemia(self, glicemia):CRLF
280         if glicemia < 70:CRLF
281             print("Glicemia sotto la soglia")CRLF
282             self.declare(Fact(rischio_ipoglicemia=True))CRLF
283         else:CRLF
284             self.declare(Fact(rischio_ipoglicemia=False))CRLF
285 CRLF
286     #se utente è maggiorenne conto i sintomi dell'avere poco zucchero, esistere
        almeno 2 sintomiCRLF
287     CRLF
288     @Rule(Fact(maggiorenne=True),CRLF
289           AS.p << Persona(),CRLF
290           TEST(lambda p: conteggio(p,CRLF
291                                    'fame_insaziabile',CRLF
292                                    'mal_di_testa',CRLF
293                                    'sudorazione',CRLF
294                                    'tremore_alla_mano',CRLF
295                                    'volto_pallido') > 2))CRLF
296     def poco_zucchero(self, p):CRLF
297         self.declare(Fact(poco_zucchero=True))CRLF
298 CRLF
299     #se utente è maggiorenne e ci sono segnali di poco zucchero nel sangue CRLF
300     @Rule(Fact(maggiorenne=True),CRLF
301           Fact(parenti_diabetici=True),CRLF
302           Fact(poco_zucchero=True))CRLF
303     def rischio_basso(self):CRLF
304         print("Attenzione, il paziente potrebbe essere diabetico")CRLF
305 CRLF
306     #se utente è maggiorenne e ci sono segnali di poco zucchero nel sangue e glicemia
        sotto sogliaCRLF
307     @Rule(Fact(maggiorenne=True),CRLF
308           Fact(rischio_ipoglicemia=True),CRLF
309           Fact(poco_zucchero=True))CRLF
310     def allarme_basso(self):CRLF
311         print("Pericolo! Alto rischio di diabete, visitare da specialista")CRLF
312 CRLF
313     #se utente è maggiorenne e ha parenti con diabete CRLF
314     @Rule(Fact(maggiorenne=True),CRLF
315           Persona(parentela_diabetica=True))CRLF
316     def parenti_diabetici(self):CRLF
317         self.declare(Fact(parenti_diabetici=True))CRLF
318 CRLF
319     #se utente è maggiorenne conto i sintomi dell'avere troppo zucchero, esistere
        almeno 2 sintomiCRLF
320     @Rule(Fact(maggiorenne=True),CRLF
321           AS.p << Persona(),CRLF
322           TEST(lambda p: conteggio(p,CRLF
323                                    'bocca_asciutta',CRLF
324                                    'formicolio_mani_piedi'CRLF
325                                    'lenta_guarigione_ferite',CRLF
326                                    'mal_di_testa',CRLF
327                                    'sete_frequente',CRLF
328                                    'vista_offuscata',CRLF
329                                    'urina_spesso',) > 2)CRLF
330     )CRLF
331     def troppo_zucchero(self, **_):CRLF
332         self.declare(Fact(troppo_zucchero=True))CRLF
333 CRLF
334     @Rule(Fact(maggiorenne=True),CRLF
335           Fact(parenti_diabetici=True),CRLF
336           Fact(troppo_zucchero=True))CRLF
337     def rischio_alto(self):CRLF
338         print("Attenzione, il paziente potrebbe essere diabetico")CRLF
339 CRLF
340     @Rule(Fact(maggiorenne=True),CRLF
341           Fact(rischio_iperglicemia=True),CRLF
342           Fact(troppo_zucchero=True))CRLF
343     def allerta_alto(self):CRLF
344         print("Pericolo! Alto rischio di diabete, visitare da specialista")CRLF
```

```
CRLF
CRLF
engine = AnalisiDiabete() #attiva il sistema esperto CRLF
engine.reset()CRLF
#dichiarazione di eta, analisi glicemia e sintomi in ordine alfabetico per migliorare leggibilità CRLF
engine.declare(Persona(eta=29,CRLF
                       glicemia=80,CRLF
                       bocca_asciutta= False,CRLF
                       fame_insaziabile= True,CRLF
                       formicolio_mani_iedi= True,CRLF
                       lenta_guarigione_ferite= False,CRLF
                       mal_di_testa= True,CRLF
                       sete_frequente= True,CRLF
                       sudorazione= True,CRLF
                       tremore_alla_mano= False,CRLF
                       volto_pallido= False,CRLF
                       vista_offuscata= False,CRLF
                       urina_spesso= False ))CRLF
engine.run()CRLF
CRLF
CRLF
#un altro test CRLF
engine.declare(Persona(eta=69,CRLF
                       glicemia=80,CRLF
                       bocca_asciutta= False,CRLF
                       fame_insaziabile= False,CRLF
                       formicolio_mani_iedi= False,CRLF
                       lenta_guarigione_ferite= False,CRLF
                       mal_di_testa= False,CRLF
                       sete_frequente= False,CRLF
                       sudorazione= False,CRLF
                       tremore_alla_mano= False,CRLF
                       volto_pallido= False,CRLF
                       vista_offuscata= False,CRLF
                       urina_spesso= False ))CRLF
engine.run()CRLF
CRLF
CRLF
#-------------------------------------------------------------------------------------------
----CRLF
#Capitolo 7 Fuzzy logic CRLF
CRLF
#installazione della libreria fuzzy-c-means CRLF
!git clone https://github.com/omadson/fuzzy-c-means.git CRLF
!pip install fuzzy-c-means CRLF
CRLF
from fcmeans import FCM  CRLF
#carica la funzione per fare algoritmo fuzzy clustering CRLF
from sklearn.datasets import make_blobs  CRLF
#carica i dati di esempio da libreria per apprendimento automatico CRLF
from matplotlib import pyplot as plt  CRLF
#carica la libreria per la creazione di grafici CRLF
from seaborn import scatterplot as scatter CRLF
#carica la funzione per fare scatter plot, grafico a dispersione in cui due variabili di un set di dati sono riportate su uno spazio cartesiano CRLF
CRLF
n_samples = 50000 #numero di oggetti da classificare CRLF
n_bins = 3  # 3 classi in cui classificare CRLF
centers = [(-5, -5), (0, 0), (5, 5)]  CRLF
#centri delle classi CRLF
X,_ = make_blobs(n_samples=n_samples, n_features=2, cluster_std=1.0,centers=centers, shuffle=False, random_state=42) CRLF
#creazione di grande gruppo contenente gli oggetti da classificare intorno ai 3 centri CRLF
CRLF
#esecuzione del fuzzy-c-means CRLF
fcm = FCM(n_clusters=n_bins) CRLF
fcm.fit(X) CRLF
CRLF
#generazione centri dei gruppi ottenuti output CRLF
fcm_centers = fcm.centers CRLF
```

```
fcm_labels  = fcm.u.argmax(axis=1)CRLF
CRLF
#creazione dei graficiCRLF
%matplotlib inlineCRLF
f, axes = plt.subplots(1, 2, figsize=(11, 5))CRLF
axes[0].scatter(X[:, 0], X[:, 1]) #subplot1CRLF
axes[0].set_title('Dati originali')CRLF
scatter = axes[1].scatter(X[:, 0], X[:, 1], c=fcm_labels) #subplot2CRLF
axes[1].scatter(fcm_centers[:, 0], fcm_centers[:, 1], marker='s', s=200, c='red')CRLF
axes[1].set_title('Cluster FCM')CRLF
plt.show()CRLF
CRLF
CRLF
CRLF
CRLF
CRLF
#installazione della libreria fuzzy-c-meansCRLF
!git clone https://github.com/omadson/fuzzy-c-means.gitCRLF
!pip install fuzzy-c-meansCRLF
CRLF
from fcmeans import FCM CRLF
#carica la funzione per fare algoritmo fuzzy clusteringCRLF
from sklearn.datasets import make_blobs CRLF
#carica i dati di esempio da libreria per apprendimento automaticoCRLF
from matplotlib import pyplot as plt CRLF
#carica la libreria per la creazione di graficiCRLF
from seaborn import scatterplot as scatterCRLF
#carica la funzione per fare scatter plot, grafico a dispersione in cui due variabili
di un set di dati sono riportate su uno spazio cartesianoCRLF
CRLF
#dati da classificareCRLF
n_bins = 2CRLF
X=np.array([[2,2],[2,4],[2,6],[3,3],[3,4],[3,5],[4,4],[5,4],[6,4],[7,3],[7,4],[7,5],
CRLF
        [8,2],[8,4],[8,6]])CRLF
centers = [(4,3), (4, 8)]CRLF
CRLF
# fuzzy-c-meansCRLF
fcm = FCM(n_clusters=2)CRLF
fcm.fit(X)CRLF
CRLF
# outputCRLF
fcm_centers = fcm.centersCRLF
fcm_labels  = fcm.u.argmax(axis=1)CRLF
CRLF
CRLF
#creazione dei graficiCRLF
%matplotlib inlineCRLF
f, axes = plt.subplots(1, 2, figsize=(11, 5))CRLF
axes[0].scatter(X[:, 0], X[:, 1]) #subplot1CRLF
axes[0].set_title('Dati originali')CRLF
scatter = axes[1].scatter(X[:, 0], X[:, 1], c=fcm_labels) #subplot2CRLF
axes[1].scatter(fcm_centers[:, 0], fcm_centers[:, 1], marker='s', s=200)CRLF
axes[1].set_title('Cluster FCM')CRLF
plt.show()CRLF
CRLF
CRLF
CRLF
CRLF
CRLF
!git clone https://github.com/scikit-fuzzy/scikit-fuzzy.gitCRLF
!pip install scikit-fuzzyCRLF
CRLF
import numpy as npCRLF
import skfuzzyCRLF
import skfuzzy as fuzzCRLF
CRLF
x = np.arange(11)CRLF
mfx = fuzz.trimf(x, [0, 5, 10])CRLF
xCRLF
mfxCRLF
 CRLF
```

```
CRLF
CRLF
 #crea le librerie, se non già eseguita in precedenzaCRLF
!git clone https://github.com/scikit-fuzzy/scikit-fuzzy.gitCRLF
!pip install scikit-fuzzyCRLF
CRLF
from __future__ import division, print_functionCRLF
import numpy as npCRLF
import matplotlib.pyplot as pltCRLF
import skfuzzy as fuzzCRLF
CRLF
colors = ['b', 'orange', 'g', 'r', 'c', 'm', 'y', 'k', 'brown', 'green']CRLF
CRLF
#definisce i centri per tre gruppiCRLF
centers = [[4, 2], [1, 7], [5, 6]]CRLF
CRLF
#definisce parametro sigmas in (x,y) per i tre clusterCRLF
sigmas = [[0.8, 0.3], [0.3, 0.5], [1.1, 0.7]]CRLF
CRLF
#genera dati casuali per fare testCRLF
np.random.seed(42)CRLF
xpts = np.zeros(1)CRLF
ypts = np.zeros(1)CRLF
labels = np.zeros(1)CRLF
for i, ((xmu, ymu), (xsigma, ysigma)) in enumerate(zip(centers, sigmas)):CRLF
    xpts = np.hstack((xpts, np.random.standard_normal(200) * xsigma + xmu))CRLF
    ypts = np.hstack((ypts, np.random.standard_normal(200) * ysigma + ymu))CRLF
    labels = np.hstack((labels, np.ones(200) * i))CRLF
CRLF
#visualizza i dati creati per fare testCRLF
fig0, ax0 = plt.subplots()CRLF
for l in range(3):CRLF
    ax0.plot(xpts[labels == l], ypts[labels == l], 'o', color=colors[l],CRLF
             label='gruppo '+str(l+1))CRLF
ax0.set_title('Test su 200 dati per 3 gruppi')CRLF
ax0.legend()CRLF
CRLF
#crea ciclo e grafico di riepilogo con sotto graficiCRLF
fig1, axes1 = plt.subplots(3, 3, figsize=(8, 8))CRLF
alldata = np.vstack((xpts, ypts))CRLF
fpcs = []CRLF
CRLF
for ncenters, ax in enumerate(axes1.reshape(-1), 2):CRLF
    cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(alldata, ncenters, 2, error=0.005
    , maxiter=1000, init=None)CRLF
——→#fuzz.cluster.cmeans la funzione relativa all'algoritmo da usareCRLF
CRLF
    #visualizza i dati secondo il numero di gruppi sceltoCRLF
    cluster_membership = np.argmax(u, axis=0)CRLF
    for j in range(ncenters):CRLF
        ax.plot(xpts[cluster_membership == j],CRLF
                ypts[cluster_membership == j], '.', color=colors[j])CRLF
CRLF
    #evidenzia il centro di ogni gruppo fuzzy per farlo risaltareCRLF
    for pt in cntr:CRLF
        ax.plot(pt[0], pt[1], 'rs')CRLF
CRLF
    ax.set_title('Gruppi = {0}; FPC = {1:.2f}'.format(ncenters, fpc))CRLF
    ax.axis('off')CRLF
CRLF
fig1.tight_layout()CRLF
CRLF
#scelta soluzione con 3 gruppi recupera i parametri con nuovo centro essendo una
scelta casualeCRLF
cntr, u_orig, _, _, _, _, _ = fuzz.cluster.cmeans(alldata, 3, 2, error=0.005, maxiter=
1000)CRLF
CRLF
CRLF
#genera nuovi punti intorno ai tre gruppi individuati con intervallo [0, 10] in x, y
CRLF
newdata = np.random.uniform(0, 1, (1100, 2)) * 10CRLF
CRLF
```

```
#usa di `cmeans_predict` e `cntr` sui nuovi dati
u, u0, d, jm, p, fpc = fuzz.cluster.cmeans_predict(newdata.T, cntr, 2, error=0.005, maxiter=1000)

#visualizza i dati, il valore fuzzy si ottiene da cmeans_predict e viene
#defuzzificato come valore massimo della funzione di appartenenza
#per scegliere il numero corrispondente al colore del gruppo associato

cluster_membership = np.argmax(u, axis=0) #scelta del valore massimo adeguato a visualizzare

fig3, ax3 = plt.subplots()
ax3.set_title('Nuovi punti assegnati ai 3 gruppi esistenti')
for j in range(3):
    ax3.plot(newdata[cluster_membership == j, 0],
             newdata[cluster_membership == j, 1], 'o',
             label='gruppo ' + str(j+1))
ax3.legend()

plt.show()



!git clone https://github.com/scikit-fuzzy/scikit-fuzzy.git
!pip install scikit-fuzzy
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control

#insiemi fuzzy antecedenti e funzioni appartenenza
qualità = control.Antecedent(np.arange(0, 11, 1), 'qualità')
servizio = control.Antecedent(np.arange(0, 11, 1), 'servizio')
#insieme fuzzy conseguente e funzione di appartenenza
mancia = control.Consequent(np.arange(0, 26, 1), 'mancia')

nomi = ['bassa','media','alta'] #parole per creare insiemi fuzzy
# popola le variabili fuzzy con le parole scelte
qualità.automf(names=nomi)
servizio.automf(names=nomi)

# creazione della funzione appartenenza per output usando la libreria
mancia['bassa'] = fuzz.trimf(mancia.universe, [0, 0, 13])
#tre valori di asse ascissa per definire forma e posizione dei triangoli
mancia['media'] = fuzz.trimf(mancia.universe, [0, 13, 25])
mancia['alta'] = fuzz.trimf(mancia.universe, [13, 25, 25])

# visualizzare le variabili fuzzy con .view()
qualità.view()
servizio.view()
mancia.view()

#definizione delle regole per realizzare il comportamento
regola1 = control.Rule(qualità['bassa'] | servizio['bassa'], mancia['bassa'])
regola2 = control.Rule(servizio['media'], mancia['media'])
regola3 = control.Rule(servizio['alta'] | qualità['alta'], mancia['alta'])

#visualizzazione delle regole
regola1.view()
regola2.view()
regola3.view()

#creazione del sistema di controllo
mancia_controllo = control.ControlSystem([regola1, regola2, regola3])
calcola_mancia = control.ControlSystemSimulation(mancia_controllo)

#fornisco input
calcola_mancia.input['qualità'] = 6.5
calcola_mancia.input['servizio'] = 9.8

#da cui calcolo output
```

```
calcola_mancia.compute()
#stampo del risultato e creazione grafico per defuzzificare
print ('Mancia: '+str(int(calcola_mancia.output['mancia']))+"%")
mancia.view(sim=calcola_mancia)
#-------------------------------------------------------------------------------
#Capitolo 8 Machine learning
#semplice esempio
#importazione librerie necessarie
import numpy as np
from sklearn import datasets #riferimento ai dati di esempio
from sklearn.model_selection import train_test_split #organizza dati learning
from sklearn.linear_model import Perceptron #modello di predizione scelto
from sklearn.metrics import accuracy_score #calcolo dell'accuratezza
#carica gli specifici dati Iris
iris = datasets.load_iris()
X = iris.data #input
y = iris.target #output
#divide in insiemi di training 80% e test 20% senza scelta casuale
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
#impostazione parametri del modello neural network Perceptron
ppn = Perceptron(max_iter=40, tol=0.001, eta0=0.01, random_state=0)
#supervised learning del modello scelto
ppn.fit(X_train, y_train)
#predizione sul test set
y_pred = ppn.predict(X_test)
#verifica errore sul test set
accuracy_score(y_test, y_pred)
#messa in esercizio su nuovi dati di un oggetto da classificare
print ("classe predetta",ppn.predict([[5.8, 1.8, 4.1, 2.4]]))
#-------------------------------------------------------------------------------
#Capitolo 9 Pre-elaborazione dei dati
import numpy as np
c = np.array([1, 2, np.NAN, 3, 4])
print("Dati caricati:",c)
print("Test su presenta NaN:",np.isnan(c))
print("Dati caricati senza NaN:",c[~np.isnan(c)])
print("Media dei dati caricati con NaN:",np.mean(c))
print("Media dei dati caricati senza NaN:",np.mean(c[~np.isnan(c)]))
#esempio di ispezione per conoscere il dataset
#caricamento dati e librerie
from sklearn.datasets import load_iris
import pandas as pd
iris = load_iris()
print("Descrizione dataset Iris",iris.data.shape)
print("\nQuantità di righe:",iris.target.shape)
print("\nChiavi:",iris.keys())
print("\nDimensioni:",iris.data.shape)
print("\nNomi feature:",iris.feature_names)
print("\nDescrizione:",iris.DESCR)
iris_pd = pd.DataFrame(iris.data)
print("\nDati nelle prime 5 righe")
print(iris_pd.head())
iris_pd.columns = iris.feature_names
print("\nDati nelle prime 5 righe con nomi feature su colonne\n",iris_pd.head())
```

```
CRLF
print("\nPrincipali statistiche sui dati contenuti\n",iris_pd.describe())CRLF
CRLF
# prepara librerieCRLF
import matplotlib.pyplot as pltCRLF
import seaborn as snsCRLF
CRLF
#carica dataset Iris da snsCRLF
iris = sns.load_dataset("iris")CRLF
iris.head()CRLF
CRLF
#grafico della relazione di ogni feature con altre specieCRLF
plt.xlabel('Features')CRLF
plt.ylabel('Specie')CRLF
pltX = iris.loc[:, 'sepal_length']CRLF
pltY = iris.loc[:,'species']CRLF
plt.scatter(pltX, pltY, color='blue', label='sepal_length', marker=".")CRLF
pltX = iris.loc[:, 'sepal_width']CRLF
pltY = iris.loc[:,'species']CRLF
plt.scatter(pltX, pltY, color='green', label='sepal_width', marker="s")CRLF
pltX = iris.loc[:, 'petal_length']CRLF
pltY = iris.loc[:,'species']CRLF
plt.scatter(pltX, pltY, color='red', label='petal_length', marker="p")CRLF
pltX = iris.loc[:, 'petal_width']CRLF
pltY = iris.loc[:,'species']CRLF
plt.scatter(pltX, pltY, color='black', label='petal_width', marker="x")CRLF
plt.legend(loc=4, prop={'size':8})CRLF
plt.show()CRLF
CRLF
CRLF
CRLF
CRLF
#-------------------------------------------------------------------------------
----CRLF
#Capitolo 10 Valutazione modelloCRLF
CRLF
CRLF
from sklearn.metrics import mean_absolute_errorCRLF
y_true = [3, -0.5, 2, 7]CRLF
y_pred = [2.5, 0.0, 2, 8]CRLF
print(mean_absolute_error(y_true, y_pred))CRLF
CRLF
y_true = [[0.5, 1], [-1, 1], [7, -6]]CRLF
y_pred = [[0, 2], [-1, 2], [8, -5]]CRLF
print(mean_absolute_error(y_true, y_pred))CRLF
CRLF
from sklearn.metrics import mean_squared_errorCRLF
y_true = [3, -0.5, 2, 7]CRLF
y_pred = [2.5, 0.0, 2, 8]CRLF
print(mean_squared_error(y_true, y_pred))CRLF
CRLF
y_true = [[0.5, 1],[-1, 1],[7, -6]]CRLF
y_pred = [[0, 2],[-1, 2],[8, -5]]CRLF
print(mean_squared_error(y_true, y_pred))CRLF
CRLF
CRLF
CRLF
from sklearn.metrics import mean_squared_errorCRLF
from math import sqrtCRLF
y_true = [3, -0.5, 2, 7]CRLF
y_pred = [2.5, 0.0, 2, 8]CRLF
print(sqrt(mean_squared_error(y_true, y_pred)))CRLF
CRLF
y_true = [[0.5, 1],[-1, 1],[7, -6]]CRLF
y_pred = [[0, 2],[-1, 2],[8, -5]]CRLF
print(sqrt(mean_squared_error(y_true, y_pred)))CRLF
CRLF
from sklearn.metrics import r2_scoreCRLF
y_true = [3, -0.5, 2, 7]CRLF
y_pred = [2.5, 0.0, 2, 8]CRLF
print(r2_score(y_true, y_pred))CRLF
CRLF
```

```
y_true = [[0.5, 1], [-1, 1], [7, -6]]CRLF
y_pred = [[0, 2], [-1, 2], [8, -5]]CRLF
print(r2_score(y_true, y_pred))CRLF
CRLF
CRLF
CRLF
from sklearn.metrics import log_lossCRLF
print(log_loss(["spam", "no spam", "no spam", "spam"],[[.0, 1.0], [1.0, .0], [1.0, .0], [.0, 1.0]])) #tutto correttoCRLF
print(log_loss(["spam", "no spam", "no spam", "spam"],[[.1, .9], [.9, .1], [.9, .1], [.1, .9]])) #valore un poco basso ma correttoCRLF
print(log_loss(["spam", "no spam", "no spam", "spam"],[[.1, .9], [.9, .1], [.8, .2], [.45, .55]])) #un basso e un quasi indecisoCRLF
print(log_loss(["spam", "no spam", "no spam", "spam"],[[.5, .5], [.5, .5], [.5, .5], [.5, .5]])) #tutti indecisiCRLF
print(log_loss(["spam", "no spam", "no spam", "spam"],[[1.0, .0], [.0, 1.0], [.0, 1.0], [1.0, .0]])) #tutto sbagliatoCRLF
CRLF
CRLF
# classi individuate con numeriCRLF
from sklearn.metrics import confusion_matrixCRLF
y_true = [2, 0, 2, 2, 0, 1]CRLF
y_pred = [0, 0, 2, 2, 0, 2]CRLF
print("Classi con numeri\n", confusion_matrix(y_true, y_pred))CRLF
CRLF
# classi individuate con etichetteCRLF
y_true = ["cat", "ant", "cat", "cat", "ant", "bird"]CRLF
y_pred = ["ant", "ant", "cat", "cat", "ant", "cat"]CRLF
print("Classi con etichette\n", confusion_matrix(y_true, y_pred, labels=["ant", "bird", "cat"]))CRLF
CRLF
#classificatore binario e calcolo metriche da matrice confusioneCRLF
print("Classificatore binario\n", confusion_matrix([0, 1, 0, 1], [1, 1, 1, 0]))CRLF
tn, fp, fn, tp = confusion_matrix([0, 1, 0, 1], [1, 1, 1, 0]).ravel()CRLF
print("Metriche da classificatore binario", tn, fp, fn, tp)CRLF
CRLF
CRLF
CRLF
#-------------------------------------------------------------------------------------------CRLF
#Capitolo 11 k-NEAREST NEIGHBORSCRLF
CRLF
CRLF
#librerie necessarieCRLF
from sklearn import datasetsCRLF
from sklearn.model_selection import train_test_splitCRLF
from sklearn.neighbors import KNeighborsClassifierCRLF
CRLF
#prepara training e test set di IrisCRLF
iris = datasets.load_iris()CRLF
X, y = iris.data, iris.targetCRLF
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)CRLF
CRLF
#prova con diversi valori di k e scelta dei pesiCRLF
scores_list=[]CRLF
for k in range(1, 5): #da 0 a 3CRLF
    #fornire: iperparametro k, tipi di pesi da usare uniform distance, algoritmo da usare CRLF
    kNN = KNeighborsClassifier(n_neighbors=k, weights='uniform', algorithm = 'auto')CRLF
    kNN.fit(X_train, y_train) #apprendimento training setCRLF
    scores=kNN.score(X_test, y_test) #accuratezza su test setCRLF
    scores_list.append(scores)CRLF
    print ("%d-NN accuratezza: %0.2f" % (k, scores.mean()))CRLF
CRLF
#messa in esercizio prova su nuovo inputCRLF
k=3 #scegliere il parametro kCRLF
kNN = KNeighborsClassifier(n_neighbors=k, weights='uniform', algorithm = 'auto')CRLF
kNN.fit(X_train, y_train) CRLF
#ri-training set solo una volta per prepararlo con k sceltoCRLF
new=[5.0, 2.0, 5.0, 2.0]CRLF
```

```
print("Input ", new)CRLF
print("Output probabilità ", kNN.predict_proba([new]))CRLF
print("Output indice classe predetta ", kNN.predict([new]))CRLF
print("Distanza e indici punti vicini ", kNN.kneighbors([new]))CRLF
CRLF
CRLF
import matplotlib.pyplot as pltCRLF
plt.title('Relazione tra k e accuratezza')CRLF
plt.plot(range(1, 5), scores_list)CRLF
plt.xlabel('k'); plt.ylabel('Accuratezza');CRLF
plt.show()CRLF
CRLF
import numpy as npCRLF
import matplotlib.pyplot as pltCRLF
from matplotlib.colors import ListedColormapCRLF
from sklearn import neighbors, datasetsCRLF
CRLF
k = 5 #iperparametroCRLF
CRLF
#prepara training e test set di IrisCRLF
iris = datasets.load_iris()CRLF
CRLF
#scelta prime due feature per fare grafico bidimensionaleCRLF
X = iris.data[:, :2]CRLF
y = iris.targetCRLF
CRLF
h = .02  # step nella creazione di mesh grafica CRLF
CRLF
#mappe di colore da usareCRLF
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])CRLF
cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])CRLF
#cmap_light = ListedColormap(['orange', 'cyan', 'cornflowerblue']) CRLF
#cmap_bold = ListedColormap(['darkorange', 'c', 'darkblue']) CRLF
weight='distance'CRLF
#prepara il modello e apprendimentoCRLF
kNN = KNeighborsClassifier(k, weights='distance')CRLF
kNN.fit(X, y)CRLF
CRLF
# disegno delle linee di confine (decision boundary) e assegna colore del punto CRLF
# appartenente alla zona colorata con mesh [x_min, x_max]x[y_min, y_max]. CRLF
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1CRLF
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1CRLF
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))CRLF
Z = kNN.predict(np.c_[xx.ravel(), yy.ravel()])CRLF
CRLF
# crea il graficoCRLF
Z = Z.reshape(xx.shape)CRLF
plt.figure(); plt.pcolormesh(xx, yy, Z, cmap=cmap_light)CRLF
CRLF
# inserisce i punti del trainingCRLF
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold, s=25, edgecolor='k')CRLF
plt.xlim(xx.min(), xx.max()); plt.ylim(yy.min(), yy.max())CRLF
plt.title("Classificazione Iris con %i-NN e weights = 'distance'" % k)CRLF
plt.xlabel('Lunghezza sepalo (cm)'); plt.ylabel('Larghezza sepalo (cm)')CRLF
plt.text(4, 5, "setosa"); plt.text(8, 5, "virginica"); plt.text(5, 1.5, "versicolor")
CRLF
plt.show() #mostra il grafico finaleCRLF
CRLF
CRLF
CRLF
CRLF
# prepara librerieCRLF
from sklearn.datasets import load_iris CRLF
from sklearn.model_selection import train_test_split CRLF
from sklearn.neighbors import KNeighborsClassifierCRLF
CRLF
# carica datasetCRLF
iris = load_iris()CRLF
X = iris.dataCRLF
y = iris.targetCRLF
# prepara training e test set con random_state=4 per la scelta casuale CRLF
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=4)CRLF
```

```
# crea KNN con n_neighbors = 5
knn = KNeighborsClassifier(n_neighbors = 5)
# il modello apprende
knn.fit(X_train, y_train)
# predizione valori per X_test test set
y_pred = knn.predict(X_test)
print('Accuratezza: ', knn.score(X_test, y_test))

# prepara libreria per k-folder
from sklearn.model_selection import cross_val_score
# usa il modello preparato prima
knn = KNeighborsClassifier(n_neighbors = 5)
# X,y divisi in 5 folder, scoring basato su accuratezza
scores = cross_val_score(knn, X, y, cv=5, scoring='accuracy')
print("Score :", scores)

# media su cv punteggi per ottenere un punteggio di precisione
print("Media score: ", scores.mean())

import matplotlib.pyplot as plt
# %matplotlib inline

k_range = range(1, 31)  #intervallo per valori di k
k_scores = []
for k in k_range: #calcolo modello con valori di k
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X, y, cv=5, scoring='accuracy')
    k_scores.append(scores.mean())#prepara dati per il grafico
plt.plot(k_range, k_scores)
plt.xlabel('Valore di K per K-NN')
plt.ylabel('Cross-Validated accuratezza')
plt.show()


#-------------------------------------------------------------------------------
#Capitolo 14 Support Vector Machine

#prepara librerie
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import datasets
from sklearn import svm

#crea dataset
X, y = datasets.load_iris(return_X_y=True)

#organizza training e test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=0)

print(X.shape, y.shape, X_train.shape, y_train.shape, X_test.shape, y_test.shape)

#prepara modello con divisione dataset in training e test set
clf = svm.SVC(kernel='linear', C=1).fit(X_train, y_train)
print("Score :", clf.score(X_test, y_test))

#uso di cross validation
from sklearn.model_selection import cross_val_score
clf = svm.SVC(kernel='linear', C=1)
scores = cross_val_score(clf, X, y, cv=5) # 5 folder
print("Accuratezza cross validation: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

#-------------------------------------------------------------------------------
#Capitolo 15 Decision tree


# classificazione Iris
import pandas as pd
from sklearn.datasets import load_iris
```

```
from sklearn import treeCRLF
CRLF
# carica dataset e prepara i dataframe per il trainingCRLF
iris = load_iris()CRLF
X = pd.DataFrame(iris.data[:, :], columns = iris.feature_names[:])CRLF
y = pd.DataFrame(iris.target, columns =["Species"])CRLF
CRLF
# crea il modelloCRLF
dtree = tree.DecisionTreeClassifier(max_depth = 2)CRLF
dtree.fit(X,y)CRLF
print("Importanza delle feature:",dtree.feature_importances_)CRLF
print("Numero livelli creati:",dtree.tree_.max_depth)CRLF
print("Numero nodi creati:",dtree.tree_.node_count)CRLF
print("Classe:",dtree.predict([[5.1,→3.5,→1.4,→0.2]])) #classe vincenteCRLF
print("Probabilità:",dtree.predict_proba([[5.1,→3.5,→1.4,→0.2]])) #probabilità assegnateCRLF
tree.plot_tree(dtree.fit(X,y),filled='true')CRLF
#rappresentazione semplice del decision tree, con testo e grafico coloratoCRLF
CRLF
import graphviz  #libreria con varie possibilità graficheCRLF
dot_data = tree.export_graphviz(dtree, out_file=None, CRLF
                    feature_names=iris.feature_names, class_names=iris.target_names, CRLF
                    filled=True, rounded=True, special_characters=True)CRLF
graph = graphviz.Source(dot_data)CRLF
graph #rappresentazione grafica del decision treeCRLF
CRLF
# in Google CoLab mettere in altra cella ed eseguirla dopo il graficoCRLF
graph.render("decision tree iris")CRLF
#crea un file formato PDF con la rappresentazione graficaCRLF
CRLF
from sklearn.tree import export_textCRLF
r = export_text(dtree, feature_names=iris['feature_names'])CRLF
print(r) #rappresentazione del decision tree con testoCRLF
CRLF
nomef=["sepal length", "sepal width", "petal length", "petal width"]CRLF
importanza = pd.DataFrame({'feature':nomef,'importanza':dtree.feature_importances_})CRLF
importanza = importanza.sort_values('importanza',ascending=False)CRLF
print(importanza)CRLF
print("verifica totale importanza feature",sum(importanza['importanza']))CRLF
CRLF
CRLF
CRLF
#-------------------------------------------------------------------------------------CRLF
#Capitolo 16 Bayes e naive BayesCRLF
CRLF
from sklearn.datasets import load_irisCRLF
iris= load_iris()CRLF
X = iris.data[:, :] #selezione colonne con featureCRLF
y = iris.target #etichette di classi in ultima colonnaCRLF
from sklearn.model_selection import train_test_splitCRLF
#80% training e 20% testCRLF
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)CRLF
from sklearn.preprocessing import StandardScalerCRLF
scaler = StandardScaler()CRLF
scaler.fit(x_train)CRLF
x_train = scaler.transform(x_train)CRLF
x_test = scaler.transform(x_test)CRLF
from sklearn.naive_bayes import GaussianNBCRLF
classifier = GaussianNB() # Gaussian Naive BayesCRLF
classifier.fit(x_train, y_train)CRLF
y_pred = classifier.predict(x_test)CRLF
from sklearn.metrics import classification_report, confusion_matrixCRLF
print(confusion_matrix(y_test, y_pred))CRLF
print(classification_report(y_test, y_pred))CRLF
# 2 righe con dati casuali per altra provaCRLF
x_random = [[-1.56697667 , 1.22358774, -1.56980273, -1.33046652],[-2.21742620 , 3.08669365, -1.29593102,-1.07025858]]CRLF
y_random=(classifier.predict(x_random))CRLF
```

```
print("Classi",y_random)CRLF
CRLF
CRLF
#-------------------------------------------------------------------------------
----CRLF
#Capitolo 17 Neural networkCRLF
CRLF
#PerceptronCRLF
# carica librerieCRLF
from sklearn import datasetsCRLF
from sklearn.preprocessing import StandardScalerCRLF
from sklearn.linear_model import PerceptronCRLF
from sklearn.model_selection import train_test_splitCRLF
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
CRLF
import numpy as npCRLF
CRLF
# carica Iris dataset e prepara i dati per trainingCRLF
iris = datasets.load_iris()CRLF
X = iris.dataCRLF
y = iris.targetCRLF
CRLF
# divisione dataset in 70% training set e 30% test setCRLF
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)CRLF
CRLF
# prepara StandardScaler e trasforma dati X per avere media 0 e varianza 1CRLF
sc = StandardScaler()CRLF
sc.fit(X_train)CRLF
X_train_std = sc.transform(X_train)CRLF
X_test_std = sc.transform(X_test)CRLF
CRLF
# crea Perceptron con 40 iterazioni (epoche), 0.001 tolleranza sufficiente per
l'uscita, learning rate 0.1CRLF
# genera pesi casuali nei pesi iniziali, verbosità per stampare cosa calcolaCRLF
ppn = Perceptron(max_iter=40, tol=0.001, eta0=0.1, random_state=1, verbose=10)CRLF
CRLF
print("\nTipo: ",type(ppn))CRLF
CRLF
# training del PerceptronCRLF
ppn.fit(X_train_std, y_train)CRLF
CRLF
print("\nDizionario attributi con parametri di Perceptron: \n",ppn.__dict__)CRLF
print("\nValori dei pesi dopo training:\n", ppn.coef_)CRLF
print("\nValori dei bias dopo training:\n", ppn.intercept_)CRLF
CRLF
# predizione del Perceptron per test set e indici di prestazioneCRLF
y_pred = ppn.predict(X_test_std)CRLF
print("\nRisultati sul test set:\n",y_pred)CRLF
print("\nValori corretti del test set:\n",y_test)CRLF
print("\nAccuracy: %.2f" % accuracy_score(y_test, y_pred))CRLF
print("\nConfusion Matrix:\n",confusion_matrix(y_test,y_pred))CRLF
print("\nRiepilogo classificazione:\n",classification_report(y_test,y_pred))CRLF
print("\nRighe test set solo per output con classe 1\n", X_test[y_pred==1])CRLF
CRLF
#salva nel file l'oggetto con valori ottimizzati ottenuti dal trainingCRLF
import pickle #vedere https://docs.python.org/3/library/pickle.htmlCRLF
with open("perceptron.pkl", "wb") as f:CRLF
    pickle.dump(ppn,f, pickle.HIGHEST_PROTOCOL)CRLF
CRLF
#salva i valori dei pesi in file CSVCRLF
np.savetxt("PPNcoef.csv", ppn.coef_, delimiter=",")CRLF
np.savetxt("PPNintercept.csv", ppn.intercept_, delimiter=",")CRLF
CRLF
#esempio messa in esercizio di Perceptron preparato precedentementeCRLF
import pickle CRLF
with open("perceptron.pkl", "rb") as f:CRLF
    ppn = pickle.load(f)CRLF
CRLF
CRLF
CRLF
#-------------------------------------------------------------------------------
----CRLF
```

```
#Capitolo 18 Deep Neural NetworkCRLF
#crea un piccolo tensoreCRLF
from numpy import arrayCRLF
T = array([CRLF
    [[1,2,3],    [4,5,6],    [7,8,9]],CRLF
    [[11,12,13], [14,15,16], [17,18,19]],CRLF
    [[21,22,23], [24,25,26], [27,28,29]],CRLF
    ])CRLF
print(T.shape)CRLF
print(T)CRLF
CRLF
CRLF
CRLF
%tensorflow_version 2.x #attiva esecuzione della versione 2.0CRLF
import tensorflow as tfCRLF
print(tf.__version__)CRLF
CRLF
#crea un tensoreCRLF
x = [[2.]]CRLF
m = tf.square(x)CRLF
print("Descrizione di m:",m)CRLF
print("Contenuto di m:",m.numpy())CRLF
#moltiplica due tensoriCRLF
a = tf.constant([[2, 2],[3, 4]])CRLF
b = tf.constant([[2, 1], #separare le righe aumenta leggibilitàCRLF
                 [3, 2]])CRLF
ab = tf.matmul(a, b)CRLF
print('a * b = \n', ab.numpy())CRLF
CRLF
@tf.function #definisce operazione su tensoreCRLF
def f(x): return tf.add(x, 1.) #aggiunge 1 a tensore xCRLF
CRLF
a = tf.constant("Benvenuto TensorFlow 2.0!") #tensore con una stringaCRLF
print(a)CRLF
print(a.numpy())CRLF
CRLF
scalar = tf.constant(1.0) #tensore contiene una costanteCRLF
vector = tf.constant([1.0, 1.0]) #tensore come array con 2 costantiCRLF
matrix = tf.constant([[3.0, 2.0], [6.0, 4.0]]) #tensore come matrice 2x2 con 4
costantiCRLF
print(f(scalar))CRLF
print(f(vector))CRLF
print(f(matrix))CRLF
CRLF
#calcolo del grafo in figura 19.1CRLF
a=tf.constant(5)CRLF
b=tf.constant(8)CRLF
c=tf.constant(2)CRLF
d=tf.constant(3)CRLF
e=tf.constant(6)CRLF
f=((a+b+c)*d)/eCRLF
print(f)CRLF
print(f.numpy())CRLF
CRLF
#crea un computational graph con i due approcciCRLF
import numpy as npCRLF
data = np.array([[ 1.764,  0.400],CRLF
       [ 0.978,  2.240],CRLF
       [ 1.867, -0.977],CRLF
       [ 2.467, -0.107]])CRLF
CRLF
input = tf.keras.layers.InputLayer(input_shape=(2)) #input layer con 2 valoriCRLF
hidden = tf.keras.layers.Dense(3, activation='relu') #1 hidden layer denso con 3
neuroniCRLF
output = tf.keras.layers.Dense(1, activation='sigmoid') #output layer denso con 1
neuroneCRLF
#viene usato Keras per creare il modello e il contenuto delle variabiliCRLF
CRLF
#crea modello con approccio di chiamata a funzione ed esecuzione dinamicaCRLF
#i dati di input vanno nel primo livello, che è il primo nodoCRLF
#quando aggiunge il secondo nodo, l'output del primo nodo viene inserito nel secondo
nodoCRLF
```

```
# e viene calcolato l'output del secondo nodo e così via
#consente di stampare stati intermedi del modello, ma rallenta molto i calcoli
def model_1(data):
    x = input(data)
    x = hidden(x)
    print('Dopo il primo layer:', x)
    out = output(x)
    print('Dopo il secondo layer:', out)
    return out

print('Output come tensore:', model_1(data))
print('Output come valori:\n', model_1(data).numpy())

@tf.function  #crea modello con approccio creazione grafo statico
#prima collega tutti i nodi facendo una grande operazione computazionale
#quindi segue il flusso del grafo per fare i calcoli, non mostra stati intermedi
#del modello né aggiungere nodi al volo, ma è più veloce dell'altro approccio
def model_2(data):
    x = input(data)
    x = hidden(x)
    print('Dopo il primo layer:', x)
    out = output(x)
    print('Dopo il secondo layer:', out)
    return out

print('Output come tensore:', model_2(data))

for i, j in enumerate(data):
    print('batch ciclo esterno {} interno {}:'.format(i,j))
    model_1(d[np.newaxis, :])  # calcola loss con approccio eager

for i, j in enumerate(data):
    print('batch ciclo esterno {} interno {}:'.format(i,j))
    model_2(d[np.newaxis, :]) # calcola loss con approccio grafo statico

import time
start_time = time.time()
c=3
print("--- %s seconds ---" % (time.time() - start_time))

import tensorflow_datasets as tfds
print("Elenco data set in TensorFlow 2.o", tfds.list_builders())
data, info = tfds.load(name='fashion_mnist', as_supervised=True, split=None, with_info
=True)
print("Descrizione data set MNIST",info)


#-------------------------------------------------------------------------------
----
#Capitolo 20

#OpenCV e riconoscimento della faccia con occhi

# -*- coding: utf-8 -*-
import numpy as np
import pandas as pd
import cv2 as cv #a volte non si usa alias cv
from google.colab.patches import cv2_imshow # visualizzare immagini in Google Colab

from skimage import io
from PIL import Image
import matplotlib.pylab as plt

cv.__version__#mostra versione libreria installata

image = io.imread('IMG_3370.jpg') #caricare file
image= cv.cvtColor(image, cv.COLOR_BGR2RGB)
#convertire formato grafico colori in RGB, necessario per non usare colori sbabliati

cv2_imshow(image) #visualizzare immagine

print(image.dtype) #mostra tipo di dato di matrice con immagine
```

```
#mostra altezza, larghezza, profondità di matrice con immagine
print(image.shape[0],image.shape[1],image.shape[2] )
#se immagine non viene caricata il codice continua ad andare avanti, nom crea errore!

#manca immagine se la dimensione è 0x0x0 matrice tutta vuoto e bisogna ricaricare immagine

color = ('b','g','r')  #istogramma con analisi distribuzione dei colori
for i,col in enumerate(color):
    histr = cv.calcHist([image],[i],None,[256],[0,256])
    plt.plot(histr,color = col)
    plt.xlim([0,256])
plt.show()

gray_image = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
#passaggio al grigio per trovare i contorni usando 1 sola matrice di numeri
cv2_imshow(gray_image)

plt.title("Solo i contorni") #mostra i contorni delle aree colorate
plt.contour(gray_image, origin = "image")
plt.show()

ret, thresh = cv.threshold(gray_image,150,255,0)
contours, hierarchy = cv.findContours(thresh, cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)
cv.drawContours(image, contours, -1, (0, 255, 0), 3)
plt.imshow(image)#mostra i contorni trovati sull'immagine reale

face_cascade = cv.CascadeClassifier(cv.data.haarcascades + 'haarcascade_frontalface_default.xml')
faces = face_cascade.detectMultiScale(gray_image, 1.3, 5)
print(faces)
for (x,y,w,h) in faces:
    cv.rectangle(gray_image,(x,y),(x+w,y+h),(255,0,0),2)
plt.imshow(cv.cvtColor(gray_image, cv.COLOR_BGR2RGB))
#disegna rettangolo sulla parte di immagine contenente un volto umano

face_cascade_eye = cv.CascadeClassifier(cv.data.haarcascades +'haarcascade_eye.xml')
eyes = face_cascade_eye.detectMultiScale(gray_image, 1.3, 5)
print(eyes)
for (x,y,w,h) in eyes:
    cv.rectangle(gray_image,(x,y),(x+w,y+h),(255,0,0),2)
plt.imshow(cv.cvtColor(gray_image, cv.COLOR_BGR2RGB))
#disegna rettangolo sulla parte di immagine contenente gli occhi

```

Cosa è l'intelligenza artificiale? Quali problemi computazionali può risolvere? Quali passi fare per creare un algoritmo? Come organizzare i dati di input e interpretare l'output? Quale modello matematico scegliere e come programmarlo in linguaggio Python?

La nuova edizione del volume di Marmo intende rispondere a queste domande in modo pragmatico, per capire come funziona l'algoritmo, risolvere problemi tecnici e creare nuovi utilizzi. Ricca di esempi, consigli, link scelti, codice in linguaggio Python, l'opera è stata aggiornata inserendo alla fine di ogni capitolo una **raccolta di prompt da usare in ChatGPT**, con i quali sarà possibile approfondire di volta in volta l'argomento trattato.

Il libro tratta diversi temi, tra cui:

- definizioni e storia dell'intelligenza artificiale;
- progettazione dell'algoritmo;
- algoritmo evolutivo, logica fuzzy, sistema esperto;
- modelli di machine learning;
- neural network e deep learning;
- computer vision;
- contesto normativo;
- analisi serie temporali;
- programmare ChatGPT per creare testo e immagine.

RISORSE ONLINE
Su https://www.algoritmiia.it/ materiali gratuiti, integrazioni, codice in linguaggio Python.

ROBERTO MARMO è professore a contratto di informatica presso la Facoltà di Ingegneria dell'Università di Pavia, consulente e formatore sulla intelligenza artificiale per cercare e analizzare informazioni estratte da internet, social media, sensori, IoT e altre fonti. È autore di vari libri, fra cui *La matematica di Facebook* (Hoepli 2019).
Sito web **www.robertomarmo.net**

Ulrico Hoepli Editore S.p.A.
via Hoepli, 5 - 20121 Milano
www.hoeplieditore.it

€ 39,90

ebook disponibile
ISBN 978-88-360-1727-0
9 788836 017270