Applichiamo il modello su appena descritto ai dati che abbiamo collezionato con le modalità indicate nella sezione apposita.

1. Direttamente da Google Form è possibile scaricare il database generato dalle risposte al survey. Concentriamo l’analisi sulle prime 21 variabili che rappresentano il punteggio da 0 a 3 su quanto, ciascun individuo, è preparato sui requisiti delle inserzioni che verranno mostrate in seguito.
2. Importiamo la libreria *pandas* di cui eseguiamo il comando *read\_csv* per importare il file nell’ambiente Python e trasformarlo in dataframe.   
   Specifichiamo gli argomenti *sep = ‘;’* -che indica il tipo di separatore di dati- e *header = None*. L’ultimo argomento consente di aggirare il problema di intestazioni di colonna particolarmente complesse che renderebbero il codice poco leggibile ogni volta sia necessario richiamare la colonna stessa: l’intestazione di colonna diventa un’osservazione vera e propria che grazie al comando *drop* viene eliminata.   
   Grazie al comando *astype(float)*, le stringhe “numeriche” vengono trasformate in numeri decimali. Con *reset\_index()*, la riga 0, eliminata con l’operazione drop, torna ad indicizzare la prima osservazione relativa ai punteggi. Tale operazione è necessaria per consentire la corretta fusione di più dataframe.

import pandas as pd

dfa1 = pd.read\_csv(‘filepreprocessato.csv’, sep = ‘;’, header = False)

dfa1 = dfa1.drop(0)

dfa1 = dfa1.astype(float)

dfa1 = dfa1.reset\_index()

dfa1.head()

|  |  |  |  |
| --- | --- | --- | --- |
| **1** | **2** | **3** | **4** |
| 1.0 | 1.0 | 0.0 | 3.0 |
| 3.0 | 3.0 | 2.0 | 2.0 |
| 1.0 | 1.0 | 0.0 | 2.0 |

1. Vengono manipolati i dati, introducendo le variabili necessarie all’analisi. Per rendere lo script più leggero viene utilizzata la libreria *numpy* quando possibile. In particolare, la funzione *repeat,* che ripete il primo argomento tante volte quante specificate nel secondo, e *array* che genera una lista (o meglio, un array numpy). Le altre variabili sono ottenute da combinazioni lineari di altre. Tutte le variabili vengono inserite in un dataframe. Si noti l’argomento *.transpose()* che genera il dataframe trasposto *dfa2*. Ciò perché il comando *DataFrame* genera un dataframe impilando gli array e non concatenandoli orizzontalmente.

import numpy as np

needa1 = np.repeat(2.55,9)

needb1 = np.repeat(2,9)

avga1 = dfa1[[4, 5]].mean(axis=1)

avgb1 = dfa1[[1, 7, 9, 12]].mean(axis=1)

diffa1 = avga1/needa1

diffb1 = avgb1/needb1

q1 = np.array([0.0,1.0,0.0,0.0,0.0,1.0,1.0,0.0,1.0])

dfa2 = pd.DataFrame((needa1, needb1, avga1,

avgb1, diffa1, diffb1, q1)).transpose()

dfa1 = pd.read\_csv(‘filepreprocessato.csv’, sep = ‘;’, header = False)

dfa1 = dfa1.drop(0)

dfa1 = dfa1.astype(float)

dfa1 = dfa1.reset\_index()

dfa1.head()

1. Viene ripetuto il punto 3 per ciascuna inserzione.
2. Vengono concatenati i dataframe attraverso il comando *concat*, specificando l’*axis = 1* che concatena i dataframe orizzontalmente, non mantenendo gli indici di colonna dei dataframe originali *ignore\_index = False.* Vengono convertiti nuovamente i valori numerici in decimali.

|  |  |  |  |
| --- | --- | --- | --- |
| **21** | **22** | **needa1** | **needb1** |
| 1.0 | 2.0 | 2.55 | 2.0 |
| 3.0 | 2.0 | 2.55 | 2.0 |
| 3.0 | 2.0 | 2.55 | 2.0 |

dfa1 = pd.concat([dfa1,dfa2,dfa3,

dfa4,dfa5,dfa6],

axis = 1, ignore\_index=True)

1. Per la dimostrazione, si realizza un modello **Support Vector Classificator**, i cui predittori saranno *diffa1*, *diffa2* e la variabile target Y, che rappresenta la scelta, o meno, di inviare la propria candidatura alla prima inserzione. Grazie al comando *iloc* creiamo la variabile X e la variabile Y.

X = dfa1.iloc[:, [27,28]].values

Y = dfa1.iloc[:, 29].values

1. Importiamo il comando *train\_test\_split* della libreria *sk.learn.model\_selection* grazie al quale dividiamo il database randomicamente ordinato nella sua prima 75esima parte, dedicata al training del modello e l’ultima 25esima parte, dedicata al testing. Il comando *random\_state* consente di replicare la randomizzazione.

from sk.learn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,

train\_size = 0.75,

random\_state=42)

1. Importiamo la funzione *SVC* dalla libreria *sklearn.svm*. Il classificatore avrà un kernel *lineare*, quindi fittiamo il modello ai dati a disposizione. Realizziamo le predizioni delle y, con gli iperparametri appena realizzati, sulla base delle *X\_test*.

from sklearn.svm import SVC

classifier = SVC(kernel = 'linear', random\_state = 0)

classifier.fit(X\_train, y\_train)

y\_pred = classifier.predict(X\_test)

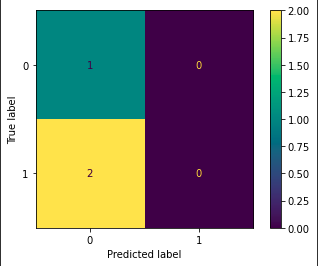
1. Viene mostrata la *confusion matrix* grazie all’apposita funzione nella libreria *sk.learn*. Ogni colonna della matrice rappresenta i valori predetti, ogni riga i valori reali. L'elemento sulla riga i e sulla colonna j è il numero di casi in cui il classificatore ha classificato la classe "vera" i come classe j. Attraverso questa matrice è osservabile se vi è "confusione" nella classificazione di diverse classi.

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

from sklearn.metrics import ConfusionMatrixDisplay

cm\_display = ConfusionMatrixDisplay(cm).plot()



1. Misuriamo l’accuracy, precision e recall delle predizioni attraverso le apposite funzioni della libreria *sklearn.metrics*.

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score

print(accuracy\_score(y\_test,y\_pred))

print(precision\_score(y\_test,y\_pred))

print(recall\_score(y\_test,y\_pred))

0.0

0.0

0.0

1. Per completezza, realizziamo un modello di **regressione logistica**, questa volta inserendo il sesso degli interrogati tra i predittori. La procedura legata alla divisione tra train e test, confusion matrix, misurazione delle performance predittive è identica. Le uniche differenza sono da ricercarsi nella scelta delle variabili e del modello.

X = dfa1.iloc[:, [27,28,58]].values

y = dfa1.iloc[:, 29].values

from sklearn.linear\_model import LogisticRegression

clf = LogisticRegression(random\_state=0).fit(X\_train, y\_train)