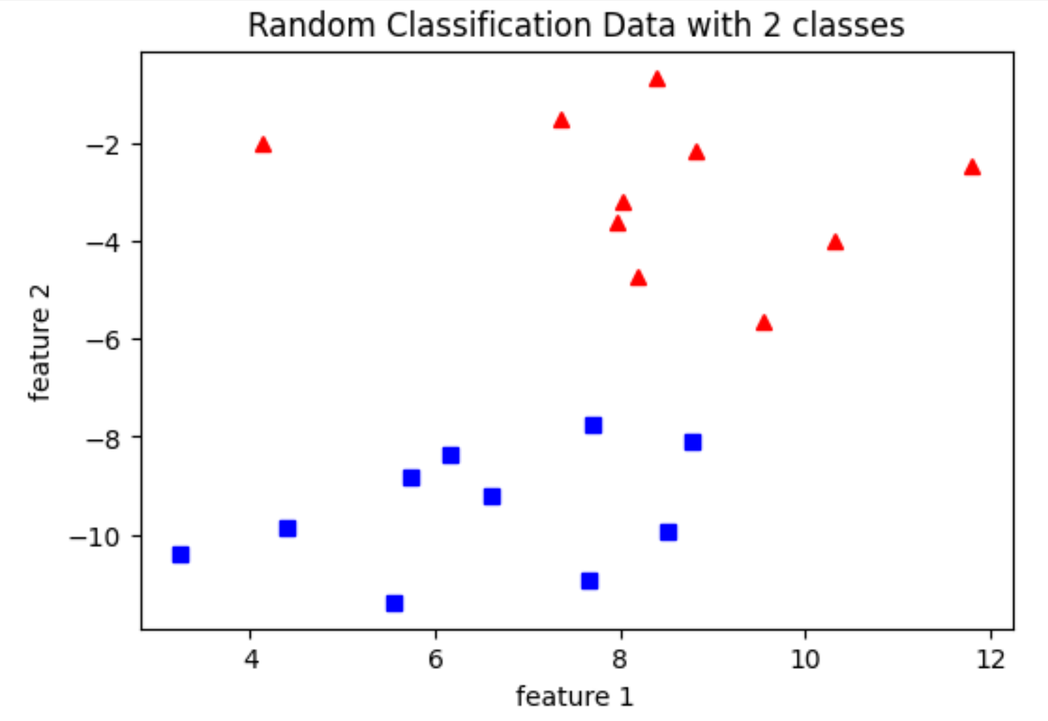
Natural and Artificial Cognition Lab Orazio Miglino
University of Naples Federico II

RETI NEURALI ARTIFICIALI

Teoria e implementazione

PROBLEMA DI CLASSIFICAZIONE



2 feature -> 2 caratteristiche delle classi

```
[[  8.77131695  -8.09700984]
 [  5.72632     -8.8155016 ]
 [  7.66156323 -10.93071471]
 [  8.18393421  -4.70904159]
 [  3.23386564 -10.41767869]
 [  7.97139445  -3.59309613]
 [ 10.31613965  -4.0052443 ]
 [  8.02389528  -3.16667702]
 [  5.54466302 -11.38884739]
 [ 11.80412386  -2.45767052]
 [  8.5021353   -9.94550812]
 [  7.71045389  -7.751419  ]
 [  6.59624461  -9.19900251]
 [  9.54779862  -5.63154374]
 [  4.12703205  -1.99052661]
 [  8.38941594  -0.64151842]
 [  6.15307256  -8.3647682 ]
 [  8.80854104  -2.14442139]
 [  4.38986875  -9.86151543]
 [  7.35338061  -1.50379713]]
```

20 samples

2 classi -> 0, 1

```
[[1]
 [1]
 [1]
 [0]
 [1]
 [0]
 [0]
 [0]
 [1]
 [0]
 [1]
 [1]
 [1]
 [0]
 [0]
 [0]
 [1]
 [0]
 [1]
 [0]]
```

PROBLEMA DI CLASSIFICAZIONE

Generazione del dataset

```
import numpy as np
from sklearn import datasets
X, t = datasets.make_blobs(n_samples=150,n_features=2,
                           centers=2,cluster_std=1.5,
                           random_state=6)
```

2 Array:
X -> features
t –> target

n_samples -> numero di sample
n_features -> numero di feature
centers -> numero di cluster / classi
cluster_std -> deviazione standard dei cluster
random_state -> inizializzazione della generazione di numeri pseudo-casuali

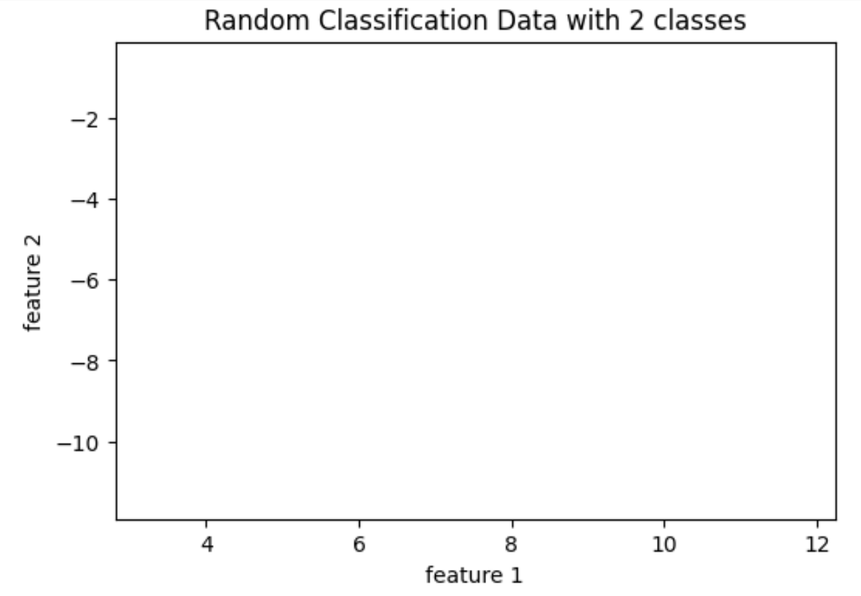
PROBLEMA DI CLASSIFICAZIONE

Visualizzazione del dataset

```python
import matplotlib.pyplot as plt

def plot_data(a,b):
  #Plotting
  fig = plt.figure(figsize=(6,4))
  plt.plot(a[:, 0][b == 0], a[:, 1][b == 0], 'r^')
  plt.plot(a[:, 0][b == 1], a[:, 1][b == 1], 'bs')
  plt.xlabel("feature 1")
  plt.ylabel("feature 2")
  plt.title('Random Classification Data with 2 classes')
  plt.show()
```

Nome della funzione e parametri



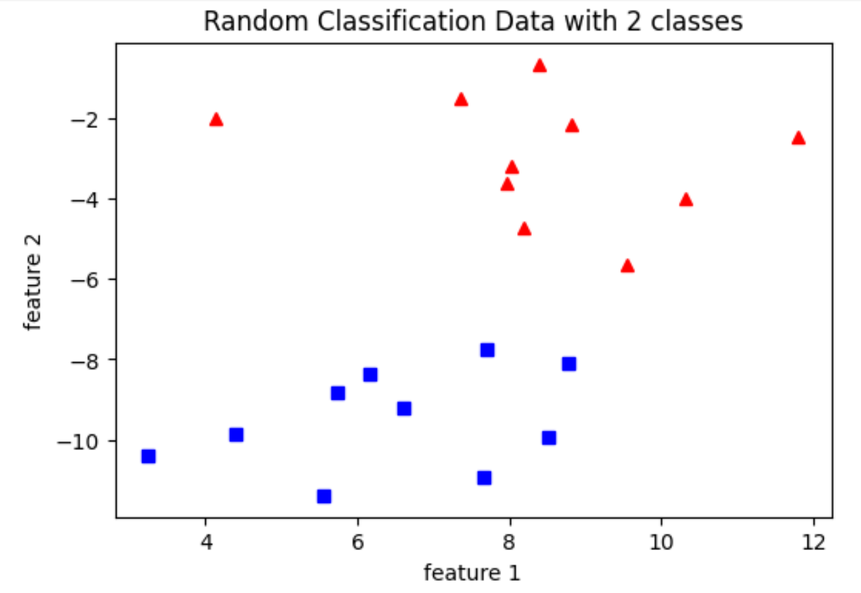
PROBLEMA DI CLASSIFICAZIONE

Generazione e visualizzazione del dataset

```
import numpy as np
from sklearn import datasets
X, t = datasets.make_blobs(n_samples=150,n_features=2,
                           centers=2,cluster_std=1.5,
                           random_state=6)

plot_data(X,t)
```

Chiamata della funzione plot_data con i parametri X e t



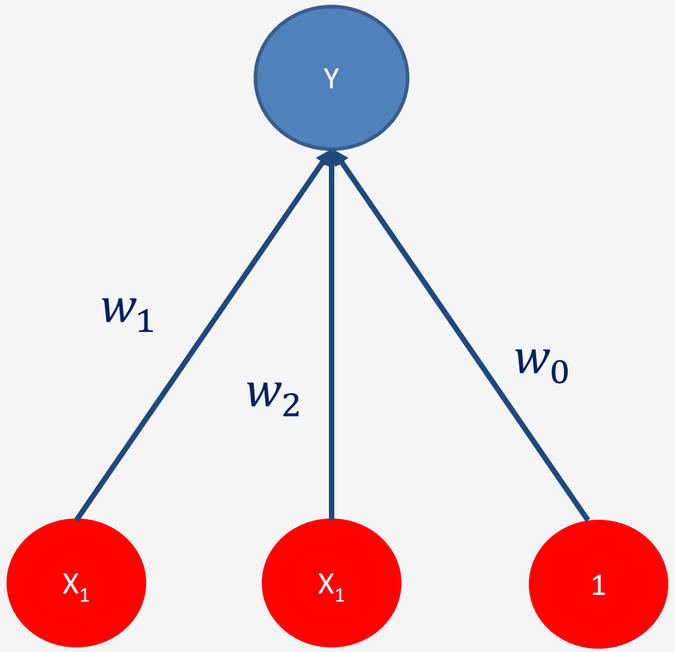
CLASSE PERCEPTRON
Metodi

Aggiungere il bias all'input

```
[[  8.77131695  -8.09700984]
 [  5.72632     -8.8155016 ]
 [  7.66156323 -10.93071471]
 [  8.18393421  -4.70904159]
 [  3.23386564 -10.41767869]
 [  7.97139445  -3.59309613]
 [ 10.31613965  -4.0052443 ]
 [  8.02389528  -3.16667702]
 [  5.54466302 -11.38884739]
 [ 11.80412386  -2.45767052]
 [  8.5021353   -9.94550812]
 [  7.71045389  -7.751419  ]
 [  6.59624461  -9.19900251]
 [  9.54779862  -5.63154374]
 [  4.12703205  -1.99052661]
 [  8.38941594  -0.64151842]
 [  6.15307256  -8.3647682 ]
 [  8.80854104  -2.14442139]
 [  4.38986875  -9.86151543]
 [  7.35338061  -1.50379713]]
```

→

```
[[  8.77131695  -8.09700984   1.        ]
 [  5.72632     -8.8155016    1.        ]
 [  7.66156323 -10.93071471   1.        ]
 [  8.18393421  -4.70904159   1.        ]
 [  3.23386564 -10.41767869   1.        ]
 [  7.97139445  -3.59309613   1.        ]
 [ 10.31613965  -4.0052443    1.        ]
 [  8.02389528  -3.16667702   1.        ]
 [  5.54466302 -11.38884739   1.        ]
 [ 11.80412386  -2.45767052   1.        ]
 [  8.5021353   -9.94550812   1.        ]
 [  7.71045389  -7.751419     1.        ]
 [  6.59624461  -9.19900251   1.        ]
 [  9.54779862  -5.63154374   1.        ]
 [  4.12703205  -1.99052661   1.        ]
 [  8.38941594  -0.64151842   1.        ]
 [  6.15307256  -8.3647682    1.        ]
 [  8.80854104  -2.14442139   1.        ]
 [  4.38986875  -9.86151543   1.        ]
 [  7.35338061  -1.50379713   1.        ]]
```



CLASSE PERCEPTRON
Metodi

Aggiungere il bias all'input

```python
#Add bias to the input
def add_bias(self, x):
  # input -> input for the ANN
  if x.ndim > 1:
    inp = np.insert(x,self.n,1,axis=1)
  else:
    inp = np.insert(x,2,1)
  return inp
# END of function
```

1 dimensione

```
[ 8.77131695 -8.09700984]
```

```
[ 8.77131695 -8.09700984  1.        ]
```

2 dimensioni

```
[[  8.77131695  -8.09700984]
 [  5.72632     -8.8155016 ]
 [  7.66156323 -10.93071471]
 [  8.18393421  -4.70904159]
 [  3.23386564 -10.41767869]
 [  7.97139445  -3.59309613]
 [ 10.31613965  -4.0052443 ]
 [  8.02389528  -3.16667702]
 [  5.54466302 -11.38884739]
 [ 11.80412386  -2.45767052]
 [  8.5021353   -9.94550812]
 [  7.71045389  -7.751419  ]
 [  6.59624461  -9.19900251]
 [  9.54779862  -5.63154374]
 [  4.12703205  -1.99052661]
 [  8.38941594  -0.64151842]
 [  6.15307256  -8.3647682 ]
 [  8.80854104  -2.14442139]
 [  4.38986875  -9.86151543]
 [  7.35338061  -1.50379713]]
```

```
[[  8.77131695  -8.09700984   1.        ]
 [  5.72632     -8.8155016    1.        ]
 [  7.66156323 -10.93071471   1.        ]
 [  8.18393421  -4.70904159   1.        ]
 [  3.23386564 -10.41767869   1.        ]
 [  7.97139445  -3.59309613   1.        ]
 [ 10.31613965  -4.0052443    1.        ]
 [  8.02389528  -3.16667702   1.        ]
 [  5.54466302 -11.38884739   1.        ]
 [ 11.80412386  -2.45767052   1.        ]
 [  8.5021353   -9.94550812   1.        ]
 [  7.71045389  -7.751419     1.        ]
 [  6.59624461  -9.19900251   1.        ]
 [  9.54779862  -5.63154374   1.        ]
 [  4.12703205  -1.99052661   1.        ]
 [  8.38941594  -0.64151842   1.        ]
 [  6.15307256  -8.3647682    1.        ]
 [  8.80854104  -2.14442139   1.        ]
 [  4.38986875  -9.86151543   1.        ]
 [  7.35338061  -1.50379713   1.        ]]
```

CLASSE PERCEPTRON
Metodi

Attivazione del neurone

```
#Calculates the activation of the neuron
def activation(self,x):
  net_input = np.dot(x,self.weights)
  output = self.step_func(net_input)
  return output
```

$$netinput_1 = \sum_{i=0}^{N} x_i w_{i1}$$

$$activation = \begin{cases} 1 \; if \;\; netinput \geq 0 \\ 0 \; if \;\; netinput < 0 \end{cases}$$

```
[[1.54891636 0.74589865 1.        ]]
```

dim[1,3]

X

```
[[0.77419012]
 [0.78337744]
 [4.5       ]]
```

dim[3,1]

1.5489 x 0.7741 + 0.7458 x 0.7833 + 1 x 4.5 =
= 6.2834

step_function(6.2834) =
= 1

CLASSE PERCEPTRON
Metodi

Funzione di attivazione

```
#defining the activation function
def step_func(self, x):
  if (x > 0):
    return 1.0
  else:
    return 0.0
#END of function
```

Esistono diverse funzioni di attivazione

In questo modo possiamo facilmente modificare la funzione di attivazione

CLASSE PERCEPTRON
Metodi

Metodo per l'addestramento della rete neurale

- Per addestrare la rete neurale utilizziamo la regola delta
- Presentiamo molte volte, in sequenza, i pattern di input del training set
- Attiviamo la rete neurale per ogni pattern di input
- Confrontiamo la previsione della rete neurale con il rispettivo target output (calcolo dell'errore)
- Modifichiamo i pesi della rete neurale

```
Per numero di epoche

        Per ogni pattern del training set

                Attivazione la rete neurale
                Calcolo dell'errore

                Se errore diverso da 0
                        modifica dei pesi con la regola delta
                Fine se
        Fine training set
Fine epoche
```

CLASSE PERCEPTRON
Metodi

Metodo per l'addestramento della rete neurale

```
# Training the ANN
def train(self):

  for epoch in range(self.epochs):

      # variable to store misclassified example
      n_miss = 0

      # looping for every example.
      for idx, i in enumerate(self.inputANN):

          # reshape the array
          i = i.reshape(1,self.n+1)
          # Calculating prediction - outpurt of the ANN
          output = self.activation(i)

          if (t[idx] - output) != 0:
              # Updating if the example is misclassified
              n_miss += 1
              #delta rule
              self.weights += self.lr*((t[idx] - output)*i.T)
          # end of if

      print(epoch, n_miss)  #end of epoch

  return self.weights
  #END of function
```

Per numero di epoche

Per ogni pattern del training set

Attivazione la rete neurale
Calcolo dell'errore

Se errore diverso da 0
modifica dei pesi con la regola delta
Fine se
Fine training set
Fine epoche

CLASSE PERCEPTRON
Metodi

Metodo per l'addestramento della rete neurale

```
# Training the ANN
def train(self):

for epoch in range(self.epochs):

    # variable to store misclassified example
    n_miss = 0

    # looping for every example.
    for idx, i in enumerate(self.inputANN):

        # reshape the array
        i = i.reshape(1,self.n+1)
        # Calculating prediction - output of the ANN
        output = self.activation(i)

        if (t[idx] - output) != 0:
            # Updating if the example is misclassified
            n_miss += 1
            #delta rule
            self.weights += self.lr*((t[idx] - output)*i.T)
        # end of if

    print(epoch, n_miss)  #end of epoch

return self.weights
#END of function
```

Dim 1

Dim 2

self.activation(x)

```
[[1.54891636 0.74589865 1.        ]]
```

dim[1,3]

X

```
[[0.77419012]
 [0.78337744]
 [4.5       ]]
```

dim[3,1]

E' necessario modificare la shape di x e farla passare da 1 a 2

Ulizziamo la funzione *reshape()*

CLASSE PERCEPTRON
Metodi

Metodo per l'addestramento della rete neurale

```
# Training the ANN
def train(self):

  for epoch in range(self.epochs):

      # variable to store misclassified example
      n_miss = 0

      # looping for every example.
      for idx, i in enumerate(self.inputANN):

          # reshape the array
          i = i.reshape(1,self.n+1)
          # Calculating prediction - outpurt of the ANN
          output = self.activation(i)

          if (t[idx] - output) != 0:
              # Updating if the example is misclassified
              n_miss += 1
              #delta rule
              self.weights += self.lr*((t[idx] - output)*i.T)
          # end of if

      print(epoch, n_miss)  #end of epoch

  return self.weights
  #END of function
```

Regola delta

$$w_0^t = w_0^{t-1} + \eta(d - y)x_1$$
$$w_1^t = w_1^{t-1} + \eta(d - y)x_2$$
$$w_2^t = w_2^{t-1} + \eta(d - y)bias$$

Calcolo dell'errore - LOSS

CLASSE PERCEPTRON
Metodi

Metodo per l'addestramento della rete neurale

```python
# Training the ANN
def train(self):

  for epoch in range(self.epochs):

      # variable to store misclassified example
      n_miss = 0

      # looping for every example.
      for idx, i in enumerate(self.inputANN):

          # reshape the array
          i = i.reshape(1,self.n+1)
          # Calculating prediction - outpurt of the ANN
          output = self.activation(i)

          if (t[idx] - output) != 0:
              # Updating if the example is misclassified
              n_miss += 1
              #delta rule
              self.weights += self.lr*((t[idx] - output)*i.T)
          # end of if

      print(epoch, n_miss)  #end of epoch

  return self.weights
  #END of function
```

```
[[1.54891636 0.74589865 1.        ]]
```
i -> dim[1,3]

```
[[0.77419012]
 [0.78337744]
 [4.5       ]]
```
self.weights -> dim[3,1]

$$w_0^t = w_0^{t-1} + \eta(d-y)x_1$$
$$w_1^t = w_1^{t-1} + \eta(d-y)x_2$$
$$w_2^t = w_2^{t-1} + \eta(d-y)bias$$

```
[[0.77419012]          [[1.54891636]
 [0.78337744]     →     [0.74589865]
 [4.5       ]]          [1.        ]]
```

dim[3,1] dim[3,1]

Trasposizione di i

```
[[1.54891636 0.74589865 1.        ]]   →   [[1.54891636]
                                            [0.74589865]
                                            [1.        ]]
```

dim[1,3] dim[3,1]

CLASSE PERCEPTRON
Metodi

Metodo per attivare la rete neurale per una sola predizione

```
#Activate the ANN
def predict(self, x):
  x = self.add_bias(x)
  output = self.activation(x)
  return output
```

- Aggiungiamo il bias al patter di input
- Attiviamo la rete neurale

CLASSE PERCEPTRON

Utilizziamo la classe Percetron per fare apprendere il pattern dei dati che abbiamo generato

```
#create the instance of the class Perceptron

learning_rate = 0.5
epochs = 10

perc = Perceptron(X, t, learning_rate, epochs)

perc.train()

t_pred = np.empty(0)
for x in X:
  val = perc.predict(x)
  t_pred = np.append(t_pred, val)
```

Decidiamo il *learning rate* e il numero di *epoche*

Creiamo una istanza della classe *Perceptron* chiamata *perc*

Chiamiamo la funzione *train()* della classe Perceptron

Attiviamo la rete neural per fare previsioni sugli input dopo l'apprendimento chiamando il metodo *predict()*

Array che contiene tutte le risposte della rete neurale

Possiamo utilizzare la funzione `plot_data(X, t_pred)` per visualizzare le risposte della rete neurale