

Sistemi esperti e soft computing

Esercitazione obbligatoria propedeutica per l'ammissione all'esame scritto

Sommario

Testo dell'esercitazione	3
Introduzione.....	4
Fase di sviluppo	6
Azioni preliminari	6
<i>networkGenerator.py</i>	6
<i>networkTest.py</i>	11
<i>networkExecute.py</i>	15
Fase di test.....	18
Analisi dei risultati	19
example1	19
example2	23
Conclusioni.....	28

Testo dell'esercitazione

Esercitazione Obbligatoria propedeutica per l'ammissione all'esame scritto

Scrivere un programma che:

- 1) Legga un dataset di regressione da un file di testo e lo carichi in una matrice (sulle righe le osservazioni sulle colonne gli attributi che descrivono gli ingressi. Si ricorda che l'ultimo attributo è il valore da stimare considerando i valori degli altri).
- 2) Generi due matrici: una per il training set e una per il test set usando, rispettivamente, il 70% e il 30% delle righe della matrice del dataset.
- 3) Costruisca vari modelli di rete neurale feedforward ad uno strato e funzione di attivazione sigmoidale variando il numero di neuroni da 5 a 50 con passo 5 (utilizzando il training set ovviamente).
- 4) Calcoli il valore dell'errore quadratico medio dell'approssimazione sia sul training sia sul test set per ogni modello.
- 5) Confronti i risultati ottenuti creando un grafico che abbia sulle ascisse il numero di neuroni e sulle ordinate i corrispondenti errori sul training e test set.

Vengono forniti due esempi di dataset (che rappresentano anche il formato di riferimento) su cui lo studente deve effettuare gli esperimenti.

Lo studente è libero di risolvere il problema utilizzando il linguaggio di programmazione o scripting preferito. Lo studente deve consegnare una relazione in cui vengono descritti tutte le scelte effettuate, i passi realizzati, i risultati ottenuti (importante organizzarli in tabelle e discuterli), discussioni e considerazioni finali. Tutti i codici e gli script prodotti devono essere consegnati e discussi in dettaglio nella relazione.

Introduzione

Per la risoluzione del problema proposto ho deciso di utilizzare i seguenti strumenti informatici:

1. **Python:** linguaggio di programmazione.
2. **pandas:** libreria software Python per la manipolazione e l'analisi dei dati.
3. **neurolab:** libreria software Python per la creazione e l'utilizzo di reti neurali.
4. **numpy:** libreria software Python per l'implementazione di calcoli matematici avanzati.
5. **pylab:** libreria software Python per la creazione di grafici.
6. **Visual studio code:** editor di testo avanzato con molte funzionalità extra improntate al mondo della programmazione.

Nell'esercitazione vengono forniti due esempi di dataset. Ho deciso di svolgere l'attività per entrambi gli esempi utilizzando le solite filosofie di data mining e machine learning. Per tale ragione discuterò in dettaglio la risoluzione di solo uno dei due dataset: "ele-2.txt" fornendo comunque risultati e informazioni importanti anche del secondo.

Il file .zip allegato alla relazione presenta la seguente struttura:

```
| esercitazione1.zip
|   Relazione esercitazione1.docx
|   Relazione esercitazione1.pdf
|   Results.xlsx
|   example1
|       | results
|       |     | *.net
|       |     | *.csv
|       |     | ele-2.csv
|       |     | networkExecute.py
|       |     | networkGenerator.py
|       |     | networkTest.py
|   example2
|       | results
|       |     | *.net
|       |     | *.csv
|       |     | networkExecute.py
|       |     | networkGenerator.py
|       |     | networkTest.py
|       |     | wizmir.csv
```

- **esercitazione1.zip**: file compresso contenente i sorgenti e i risultati delle esercitazioni svolte.
- **Relazione esercitazione1.docx**: relazione finale sull'esercitazione in formato word.
- **Relazione esercitazione1.pdf**: relazione finale sull'esercitazione in formato pdf.
- **Results.xlsx**: foglio di calcolo contenente i risultati dei test effettuati e i grafici richiesti dal testo del problema .
- **/example1**: cartella contenente sorgenti e risultati dell'esercitazione1, relativa al dataset "ele-2.txt".
- **/example2**: cartella contenente sorgenti e risultati dell'esercitazione2, relativa al dataset "wizmir.txt".
- **/results**: cartella contenente i vari risultati (reti neurali e file .csv) dell'esecuzione dei vari script Python.
- **networkGenerator.py**: script Python che, dato in input un certo dataset, genera e addestra varie reti neurali feedforward ad uno strato sottoforma di files .net all'interno della cartella /results.
- **networkTest.py**: script Python che date tutte le reti contenute nella cartella /results, fornisce informazioni salienti, sottoforma di file .csv, sui risultati ottenuti nella fase di test.
- **networkExecute.py**: script Python che consente all'utente di selezionare una delle reti presenti all'interno della cartella /results e di fornirgli in input una serie di attributi. Stampa a video il risultato della rete conseguentemente agli input forniti.
- **ele-2.csv**: dataset fornito in formato .txt al quale è stata cambiata l'estensione per essere utilizzato con la libreria pandas.
- **Wizmir.csv**: dataset fornito in formato .txt al quale è stata cambiata l'estensione per essere utilizzato con la libreria pandas.

Fase di sviluppo

Azioni preliminari

Il dataset fornito per l'esercitazione presenta estensione .txt ma analizzandone il contenuto è possibile notare che è scritto in formato CSV (Comma Separated Values). Ho cambiato l'estensione di tale file in .csv per agevolare la gestione dei dati attraverso l'utilizzo della libreria pandas.

È stato necessario eliminare manualmente l'ultima riga contenente valore null poiché si incorreva in problemi durante la fase di training delle reti neurali.

Sono state estratte casualmente dal dataset originale alcune tuple in modo che non entrino né nel training set né nel test set. Tali tuple verranno utilizzate per eseguire alcuni test di validazioni sulle reti neurali.

networkGenerator.py

Il primo dei tre script Python svolge i seguenti compiti:

1. Import dei dati da dataset in formato CSV.
2. Data cleaning.
3. Normalizzazione dei dati.
4. Splitting casuale del dataset normalizzato in due subset distinti: trainingSet e testSet, rispettivamente utilizzati per l'addestramento e per il test.
5. Creazione e addestramento di 10 reti neurali feedforward ad uno strato che differiscono tra loro per il numero di percettroni.
6. Calcolo dei tempi di addestramento per ogni rete neurale.

```
networkGenerator.py X
example1 > networkGenerator.py > ...
1 import numpy as np
2 import pandas as pd
3 import neurolab as nl
4 import time
5
```

Nelle prime righe di codice (righe 1-4) importo tutte le librerie necessarie.

```
networkGenerator.py X
example1 > networkGenerator.py > ...
5
6 # Estraggo i dati dal file .csv e li inserisco in un dataset pandas.
7 dataset = pd.read_csv("ele-2.csv", header=None, names=["attr1", "attr2", "attr3", "attr4", "result"])
8
9 # Elimino eventuali tuple contenenti MISSING VALUE
10 dataset.dropna()
11
12 # Normalizzazione dei dati
13 datasetNorm = dataset.copy()
14 features = ["attr1", "attr2", "attr3", "attr4", "result"]
15 toNorm = dataset[features]
16 datasetNorm[features] = (toNorm - toNorm.min())/(toNorm.max() - toNorm.min())
17
18 # Elimino eventuali duplicati
19 datasetNorm.drop_duplicates()
20
```

Estraggo il dataset contenuto nel file “ele-2.csv” attraverso l’utilizzo della libreria pandas (riga 7).

Elimino tutte le tuple contenenti missing values (riga 10).

Normalizzo il dataset (righe 13-16) applicando ad ogni elemento di ogni colonna la seguente formula

$$x'_{norm} = \frac{x' - x_{min}}{x_{max} - x_{min}}$$

Elimino eventuali duplicati presenti all’interno del dataset normalizzato (riga 19).

```

networkGenerator.py X
example1 > networkGenerator.py > ...
20
21 # Estraggo e salvo su file informazioni importanti sul dataset
22 datasetInfo = pd.DataFrame(columns=["min", "max"],
23                               data=[[dataset["attr1"].min(), dataset["attr1"].max()],
24                                     [dataset["attr2"].min(), dataset["attr2"].max()],
25                                     [dataset["attr3"].min(), dataset["attr3"].max()],
26                                     [dataset["attr4"].min(), dataset["attr4"].max()],
27                                     [dataset["result"].min(), dataset["result"].max()]])
28
29 datasetInfo.to_csv("results/datasetInfo.csv")
30

```

Estraggo per ogni colonna del dataset informazioni per quanto riguarda i minimi e massimi (righe 22-29). Saranno fondamentali per il successivo utilizzo delle reti neurali.

```

networkGenerator.py X
example1 > networkGenerator.py > ...
30
31 # Splitto il dataset in due dataset, uno per il training (contenente il 70% delle tuple pescate casualmente) e uno per il test (contenente il 30% delle tuple restanti)
32 trainingSet = datasetNorm.sample(frac=0.7, random_state = 200)
33 testSet = datasetNorm.drop(trainingSet.index)
34
35 #Stampo a video le principali informazioni su training set e test set
36 print("dataset size: " + str(len(dataset)) + " tuples")
37 print("trainingSet absolute size: " + str(len(trainingSet)) + " tuples")
38 print("trainingSet percentage size: " + str(np.round((len(trainingSet) / len(dataset))*100, 0)) + " %")
39 print("testSet absolute size: " + str(len(testSet)) + " tuples")
40 print("testSet percentage size: " + str(np.round((len(testSet) / len(dataset))*100, 0)) + " %")
41
42 # Stampo in csv il testSet per verificare i risultati del training
43 testSet.to_csv("results/testSet.csv", index=False)
44
45 # Stampo in csv il trainingSet per una seconda verifica del training
46 trainingSet.to_csv("results/trainingSet.csv", index=False)
47

```

Assegno alle due variabili *trainingSet* e *testSet* rispettivamente il 70% (riga 32) e il 30% (riga 33) dell'intero dataset.

Il metodo *.sample()* di pandas consente di estrarre le tuple in maniera casuale dal dataset di partenza, il parametro *random_state* rappresenta il seme del generatore di numeri casuali. Il *testSet* viene popolato effettuando la sottrazione tra l'intero dataset e il *trainingSet* appena creato, attraverso il metodo *.drop()*.

Stampo a video le principali informazioni sulle dimensioni dei subset creati (righe 36-40).

Creo due files testSet.csv e trainingSet.csv (righe 43, 46) contenente I due subset estratti dal dataset originale, saranno necessari per la successiva analisi delle reti neurali create.


```

networkGenerator.py X
example1 > networkGenerator.py > ...
47
48 # Splitto il training in inputs e targets
49 datasetInputs = trainingSet[["attr1", "attr2", "attr3", "attr4"]]
50 datasetTargets = trainingSet[["result"]]
51
52 # Inizializzo la variabile contatore del numero di neuroni utilizzati dalla rete feed-forward
53 neuronsStartNumber = 5
54 neuronsMaxNumber = 50
55 neuronsIncrement = 5
56
57 # Inizializzo una lista di timer per appuntare il tempo di addestramento delle reti al variare del numero di neuroni
58 timeList = []
59 neuronsNumberList = []
60

```

Mi preparo per il training delle reti neurali, separo i dati di input del *trainingSet* dalle etichette creando due strutture dati distinte: *datasetInputs* e *datasetTargets* (righe 49, 50).

Inizializzo qualche variabile per la gestione del ciclo di creazione delle reti neurali:

- neuronStartNumber* (riga 53) numero di percettroni nella rete neurale di partenza,
- neuronMaxNumber* (riga 54) numero di percettroni della rete neurale finale,
- neuronsIncrement* (riga 55) incremento del numero di percettroni per ogni ciclo. Creo due strutture dati di supporto di tipo lista per il calcolo del tempo di addestramento delle reti neurali: *timeList[]* e *neuronsNumberList[]* (righe 58, 59).

```

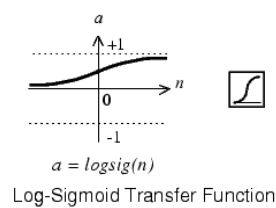
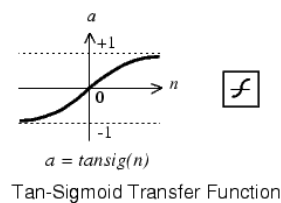
networkGenerator.py X
example1 > networkGenerator.py > ...
60
61 # Fintanto che la variabile neuronsNumber è minore di 50
62 while (neuronsStartNumber <= neuronsMaxNumber):
63
64     tStart = time.time()
65
66     # Inizializzo la rete assegnandogli i parametri necessari
67     # [[0,1], [0,1], [0,1], [0,1]] -> Numero di inputs e relativo insieme di appartenenza
68     # [neuronsNumber, 1] -> numero di neuroni e numero di outputs
69     # Successivamente cambio la funzione di trasferimento da quella di default (TanSig) a quella sigmoideale (LogSig)
70     net = nl.net.newff([[0,1], [0,1], [0,1], [0,1]], [neuronsStartNumber, 1])
71     net.layers[0].transf = nl.trans.LogSig()
72     net.layers[1].transf = nl.trans.LogSig()
73
74     # Avvio l'addestramento con il trainingSet assegnandogli i parametri necessari
75     # datasetInputs -> n tuple composte da 4 attributi tutti appartenenti all'insieme [0,1]
76     # datasetTargets -> n tuple composte da 1 attributo che rappresenta l'obiettivo del training
77     # epochs=1000 -> numero di epoche di addestramento massimo
78     # show=100 -> ogni quante epoche visualizzare l'errore attuale a video
79     # goal=0.01 -> criterio di arresto basato sul valore dell'errore
80     err = net.train(datasetInputs, datasetTargets, epochs=1500, show=50, goal=0.01)
81
82     tStop = time.time()
83
84     #Aggiungo alla lista del numero di neuroni le informazioni relative alla n-esima iterazione
85     neuronsNumberList.append(neuronsStartNumber)
86
87     # Aggiungo alla lista la misurazione del tempo necessario ad addestrare la rete neurale corrente
88     timeList.append(np.round(tStop - tStart,2))
89     print(str(neuronsStartNumber) + " neurons training duration: " + str(np.round(tStop - tStart,2)) + " seconds")
90
91     # Salvo la rete neurale su file
92     net.save("results/" + str(neuronsStartNumber) + "NeuronsNet.net")
93
94     #incremento il numero di neuroni della rete
95     neuronsStartNumber += neuronsIncrement
96

```

Il corpo del ciclo si ripete finché il numero di percettroni non ha raggiunto il massimo numero, contenuto nell'apposita variabile precedentemente inizializzata (riga 62). Creo e

assegno alla variabile **net** una rete neurale di tipo feedforward (riga 70) attraverso il metodo **.newff()** della libreria neurolab che richiede in input: numero di ingressi di ogni percettore con relativo insieme di appartenenza (parametro **minmax**), numero di percettroni e numero di uscite (parametro **size**).

La funzione di soglia di default è *TanSig*, il testo dell'esercitazione richiede esplicitamente una funzione di soglia di tipo sigmoidale. È necessario variare la funzione del primo strato e quella di uscita da *default* a *LogSig* (righe 71, 72)



Assegno alla variabile **err** l'errore in output dal training della rete (riga 80) attraverso il metodo **.train()** che richiede in input: dati di addestramento, etichette, massimo di epoche di addestramento (parametro **epochs**), ogni quante epoche mostrare l'errore attuale a video (parametro **show**), obiettivo sul valore dell'errore (parametro **goal**). Tale metodo prevede un addestramento ciclico sul training set sino al raggiungimento del valore obiettivo dell'errore o sino al raggiungimento del massimo numero di epoche specificato.

Salvo la rete neurale su file (riga 92) assegnandogli la seguente nomenclatura:

“<n>NeuronsNet.net” dove n è il numero di percettroni della rete neurale.

Incremento di un valore precedentemente dichiarato il numero di percettroni della successiva rete (riga 95).

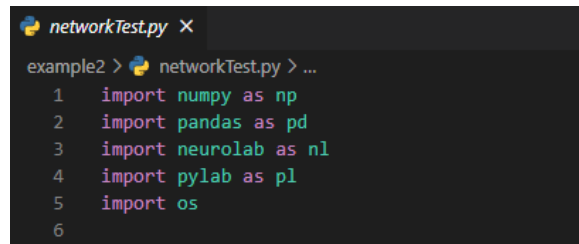
networkTest.py

Una volta eseguito lo script *networkGenerator.py*, possiamo trovare nella cartella /results una serie di files con estensione .net che rappresentano le reti neurali generate. Per ciascuna di queste è necessario eseguire una serie di test per validarne l'affidabilità.

La teoria ci insegna che è possibile validare la rete utilizzando il testSet precedentemente creato come verifica. La presente esercitazione vuole confrontare i risultati dei test sia sul testSet che sul trainingSet. Entrambi i subset sono presenti nella cartella /results poiché salvati dal precedente script.

Il secondo dei tre script Python svolge i seguenti compiti:

1. Testa la rete utilizzando il testSet, calcola per ogni tupla l'errore quadratico e stampa un report in csv.
2. Testa la rete utilizzando il trainingSet, calcola per ogni tupla l'errore quadratico e stampa un report in csv.
3. Calcola l'errore quadratico medio di entrambi i subset e stampa un report in csv.
4. Costruisce un grafico scatter con i valori degli errori a confronto, inserendo sulle ascisse il numero di perceptron della rete neurale e sulle ordinate il valore dell'errore quadratico medio.

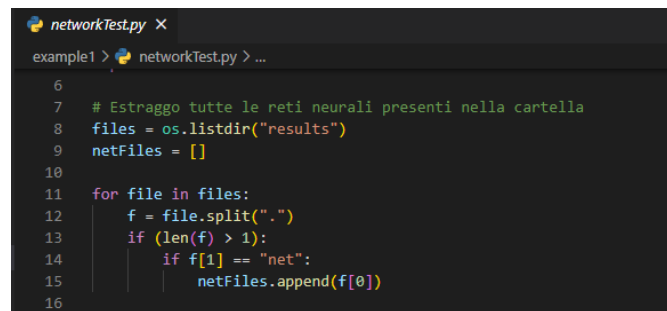


```

networkTest.py X
example2 > networkTest.py > ...
1  import numpy as np
2  import pandas as pd
3  import neurolab as nl
4  import pylab as pl
5  import os
6

```

Nelle prime righe di codice (righe 1-5) importo tutte le librerie necessarie.

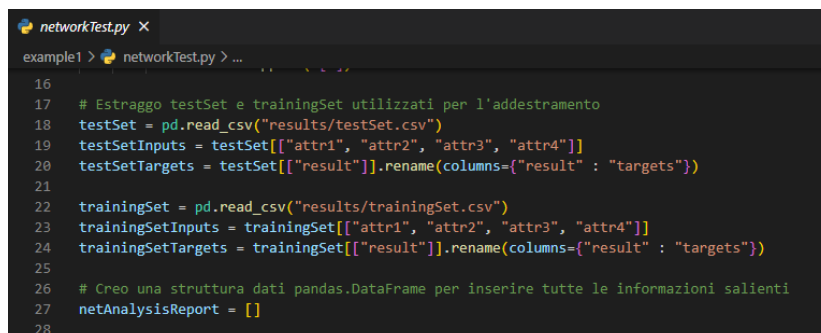


```

networkTest.py X
example1 > networkTest.py > ...
6
7  # Estraggo tutte le reti neurali presenti nella cartella
8  files = os.listdir("results")
9  netFiles = []
10
11 for file in files:
12     f = file.split(".")
13     if (len(f) > 1):
14         if f[1] == "net":
15             netFiles.append(f[0])
16

```

Ricerco all'interno della cartella /results tutti i files con estensione .net e li inserisco in una struttura dati di tipo lista *netFiles* (righe 8-15).



```

networkTest.py X
example1 > networkTest.py > ...
16
17 # Estraggo testSet e trainingSet utilizzati per l'addestramento
18 testSet = pd.read_csv("results/testSet.csv")
19 testSetInputs = testSet[["attr1", "attr2", "attr3", "attr4"]]
20 testSetTargets = testSet[["result"]].rename(columns={"result" : "targets"})
21
22 trainingSet = pd.read_csv("results/trainingSet.csv")
23 trainingSetInputs = trainingSet[["attr1", "attr2", "attr3", "attr4"]]
24 trainingSetTargets = trainingSet[["result"]].rename(columns={"result" : "targets"})
25
26 # Creo una struttura dati pandas.DataFrame per inserire tutte le informazioni salienti
27 netAnalysisReport = []
28

```

Estraggo *testSet* e *trainingSet* in formato csv dalla cartella /results precedentemente generati dallo script *networkGenerator.py* separando in distinte strutture dati input ed etichette (righe 18-24).

Creo una struttura dati di tipo lista *netAnalysisReport* nella quale saranno riportati gli errori quadratici medi relativi ad ogni rete neurale per *testSet* e *trainingSet* (riga 27).

```

networkTest.py X
example1 > networkTest.py > ...
28 |
29 | # Scorro tutte le reti neurali presenti nella cartella
30 | for neuralNet in netfiles:
31 |
32 |     # Inserisco un nuovo elemento nella struttura dati addetta all'analisi finale
33 |     netAnalysisReport.append([int(neuralNet.split("Neurons")[0]),0,0])
34 |
35 |     # Carico l'n-esima rete neurale
36 |     net = n1.load("results/" + neuralNet + ".net")
37 |
38 |     # Eseguo una simulazione sul testSet
39 |     testSetResults = pd.DataFrame(net.sim(testSetInputs)).rename(columns={0 : "results"})
40 |
41 |     result = pd.concat([testSetTargets, testSetResults], axis=1)
42 |
43 |     # Calcolo l'errore quadratico sulle singole tuple tra il risultato ottenuto e il risultato atteso
44 |     squaredError = []
45 |
46 |     for i in result.index:
47 |         squaredError.append(pow((result["targets"][i] - result["results"][i]),2))
48 |
49 |     netAnalysisReport[-1][1] = sum(squaredError)/len(squaredError)
50 |
51 |     result = pd.concat([result, pd.DataFrame(squaredError).rename(columns={0 : "squaredErr"})], axis=1)
52 |
53 |     # Stampo il risultato in csv
54 |     result.to_csv("results/" + str(neuralNet) + "_testOnTestSet.csv", index=False)
55 |
56 |     # Eseguo una simulazione sul trainingSet
57 |     trainingSetResults = pd.DataFrame(net.sim(trainingSetInputs)).rename(columns={0 : "results"})
58 |
59 |     result = pd.concat([trainingSetTargets, trainingSetResults], axis=1)
60 |
61 |     # Calcolo l'errore quadratico sulle singole tuple tra il risultato ottenuto e il risultato atteso
62 |     squaredError = []
63 |
64 |     for i in result.index:
65 |         squaredError.append(pow((result["targets"][i] - result["results"][i]),2))
66 |
67 |     netAnalysisReport[-1][2] = sum(squaredError)/len(squaredError)
68 |
69 |     result = pd.concat([result, pd.DataFrame(squaredError).rename(columns={0 : "squaredErr"})], axis=1)
70 |
71 |     # Stampo il risultato in csv
72 |     result.to_csv("results/" + str(neuralNet) + "_testOnTrainingSet.csv", index=False)
73 |

```

Il corpo del ciclo si ripete per ogni rete neurale presente nella cartella /results (riga 30).

Nella struttura dati **netAnalysisReport** inserisco una nuova tupla aggiungendo l'informazione relativa al numero dei percettroni della rete (riga 33).

Carico l'n-esima rete neurale e la assegno alla variabile **net** (riga 36).

Eseguo una simulazione della rete (riga 39) attraverso il metodo **.sim()** che richiede in input un set di dati e restituisce un set di risultati che salvo su una struttura dati **testSetResults**.

Calcolo l'errore quadratico sulle singole tuple (riga 46, 47) attraverso la seguente formula:

$$SquaredError_i = (target_i - result_i)^2$$

Inserisco tale informazione all'interno della struttura dati **result** (riga 51) che già contiene informazioni sui risultati attesi e calcolati. Salvo questo file in formato CSV per il testSet di ogni rete neurale (riga 54).

Calcolo successivamente l'errore quadratico medio su tutta la struttura attraverso la seguente formula:

$$MeanSquaredError = \frac{1}{n} * \sum_{i=1}^n squaredError_i$$

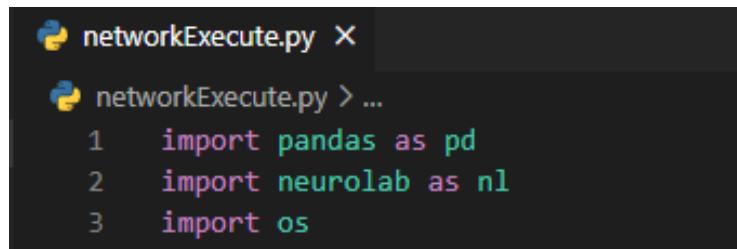
Aggiungo tale informazione alla struttura dati ***netAnalysisReport*** (riga 49).

Ripeto il solito procedimento sull'analisi dei risultati sul testSet anche per il trainingSet (righe 57-72).

networkExecute.py

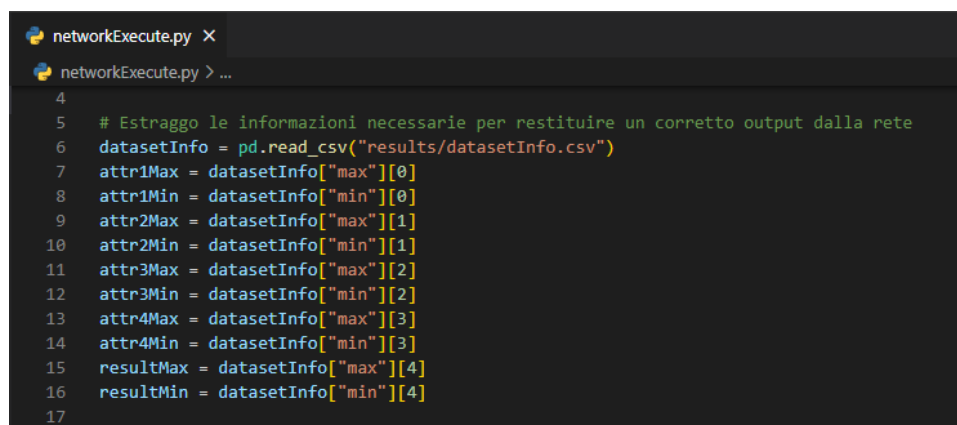
Il terzo dei tre script Python svolge i seguenti compiti:

1. Consente all'utente di selezionare una tra le reti neurali presenti nella cartella /results.
2. Richiede in input i dati necessari.
3. Restituisce in output il risultato della rete.



```
networkExecute.py X
networkExecute.py > ...
1  import pandas as pd
2  import neurolab as nl
3  import os
```

Nelle prime righe di codice (righe 1-3) importo tutte le librerie necessarie.



```
networkExecute.py X
networkExecute.py > ...
4
5  # Estraggo le informazioni necessarie per restituire un corretto output dalla rete
6  datasetInfo = pd.read_csv("results/datasetInfo.csv")
7  attr1Max = datasetInfo["max"][0]
8  attr1Min = datasetInfo["min"][0]
9  attr2Max = datasetInfo["max"][1]
10 attr2Min = datasetInfo["min"][1]
11 attr3Max = datasetInfo["max"][2]
12 attr3Min = datasetInfo["min"][2]
13 attr4Max = datasetInfo["max"][3]
14 attr4Min = datasetInfo["min"][3]
15 resultMax = datasetInfo["max"][4]
16 resultMin = datasetInfo["min"][4]
17
```

Estraggo dal file databaseInfo.csv, generato dallo script *networkGenerator.py*, le informazioni sugli attributi del dataset originale per correggere i dati in input e ricostruire il valore corretto dell'uscita della rete neurale (righe 6-16).

```

networkExecute.py X
networkExecute.py > ...
17
18 # Estraggo tutte le reti neurali presenti nella cartella
19 files = os.listdir("results")
20 netFiles = []
21
22 for file in files:
23     f = file.split(".")
24     if (len(f) > 1):
25         if f[1] == "net":
26             netFiles.append(f[0])
27
28 # Elenco a video tutte le reti neurali disponibili nella cartella
29 print("There are " + str(len(netFiles)) + " neural network ready to load:")
30
31 for index, net in enumerate(netFiles):
32     print(str(index+1) + " ) " + net)
33
34 # Chiedo all'utente di specificare una rete tra quelle disponibili
35 print("Which network to load?")
36 selectedNetwork = input()
37

```

Ricerco all'interno della cartella /results tutti i files con estensione .net e li inserisco in una struttura dati di tipo lista *netFiles* (righe 22-26).

Elenco a video tutte le reti neurali disponibili nella cartella (righe 29-32) e chiedo all'utente di selezionarne una (righe 35, 36)

```

networkExecute.py X
networkExecute.py > ...
37
38 try:
39
40     if(int(selectedNetwork) > len(netFiles)):
41         raise
42
43     else:
44
45         # Carico la rete selezionata
46         net = n1.load("results/" + netFiles[int(selectedNetwork)] + ".net")
47
48         while(True):
49             # Chiedo all'utente di inserire i quattro attributi di input
50             print("Inserire i quattro attributi di input")
51             print("attr1:")
52             attr1 = (float(input()) - attr1Min) / (attr1Max - attr1Min)
53             print("attr2:")
54             attr2 = (float(input()) - attr2Min) / (attr2Max - attr2Min)
55             print("attr3:")
56             attr3 = (float(input()) - attr3Min) / (attr3Max - attr3Min)
57             print("attr4:")
58             attr4 = (float(input()) - attr4Min) / (attr4Max - attr4Min)
59
60             # Calcolo l'output relativo ai quattro attributi e lo mostro a video
61             res = net.sim([[attr1, attr2, attr3, attr4]])[0][0]
62             res = res*resultMax - res*resultMin + resultMin
63             print("network output: " + str(res))
64
65 except:
66
67     print("Invalid input!")
68     exit
69

```

Racchiudo tutto il codice successivo all'interno di un *try-except* per gestire gli input invalidi (righe 38, 65).

Carico la rete neurale selezionata dall'utente e la assegno alla variabile **net** (riga 46).

Entro in un ciclo infinito (riga 48) dove richiedo all'utente gli input della rete neurale (righe 50-58).

Per rendere validi gli input quest'ultimi andranno normalizzati nel medesimo modo con cui è stato normalizzato il dataset di addestramento e test. È possibile normalizzare tali valori utilizzando i minimi e massimi del dataset originale utilizzando la seguente formula:

$$x'_{norm} = \frac{x' - x_{min}}{x_{max} - x_{min}}$$

Attraverso il metodo **.sim()** fornisco in input alla rete neurale la struttura dati di un elemento contenente i quattro valori specificati dall'utente e successivamente normalizzati. Assegno il risultato restituito alla variabile **res** (riga 61).

Il valore contenuto in **res** è ovviamente normalizzato, attraverso la seguente formula, inversa della precedente, posso ottenere il valore originale (riga 62):

$$x' = x'_{norm} * x_{max} - x'_{norm} * x_{min} + x_{min}$$

Stampo a video tale valore (riga 63).

Fase di test

Dopo la scrittura del codice e i vari test per verificarne il corretto funzionamento ho deciso di procedere come segue:

1. Creazione di 10 reti neurali composte da un numero incrementale di percettroni: 5, 10, 15, 20, 25, 30, 35, 40, 45, 50 (come da testo dell'esercitazione) su diverse epoche di addestramento: 50, 100, 200, 500, 1000 con un obiettivo di errore di 0.01, utilizzando lo script *networkGenerator.py*.
2. Misurazione dei tempi di addestramento di ogni rete neurale per ogni classe di epoche di addestramento, utilizzando lo script *networkGenerator.py*.
3. Misurazione degli errori quadratici medi sui test effettuati su testSet e trainingSet di ogni rete neurale per ogni classe di epoche di addestramento, utilizzando lo script *networkTest.py*.
4. Creazione di un foglio di calcolo che confronti tutti i risultati attraverso grafici.
5. Confronto dei risultati ottenuti applicando lo stesso input tra le reti neurali migliore e peggiore (in termini di errori sul testSet) generate precedentemente, utilizzando lo script *networkExecute.py*.

Analisi dei risultati

example1

example1 si riferisce al dataset “ele-2.txt” che, dopo la prima fase di data mining, presenta le seguenti caratteristiche:

```
dataset size: 1055 tuples
trainingSet absolute size: 738 tuples
trainingSet percentage size: 70.0 %
testSet absolute size: 317 tuples
testSet percentage size: 30.0%
```

Riporto di seguito i dati relativi ad una sessione di addestramento della rete, estratto della tabella presente nel foglio di calcolo Results.xlsx.

Training Epochs [-]	requested training time [s]	Neurons Number [-]	testSet Mean Squared Error [-]	trainingSet Mean Squared Error [-]	Percentage distance [%]	Mean percentage distance [%]
200	1.76E+01	5	1.66E-04	1.63E-04	2.04E+00	1.32E+01
	1.72E+01	10	1.60E-04	1.41E-04	1.34E+01	
	1.97E+01	15	1.53E-04	1.28E-04	1.89E+01	
	1.75E+01	20	1.42E-04	1.32E-04	7.25E+00	
	1.77E+01	25	1.38E-04	1.22E-04	1.28E+01	
	1.78E+01	30	1.41E-04	1.22E-04	1.62E+01	
	1.82E+01	35	1.28E-04	1.11E-04	1.54E+01	
	2.02E+01	40	1.29E-04	1.17E-04	9.56E+00	
	1.87E+01	45	1.30E-04	1.12E-04	1.55E+01	
	1.90E+01	50	1.38E-04	1.14E-04	2.13E+01	

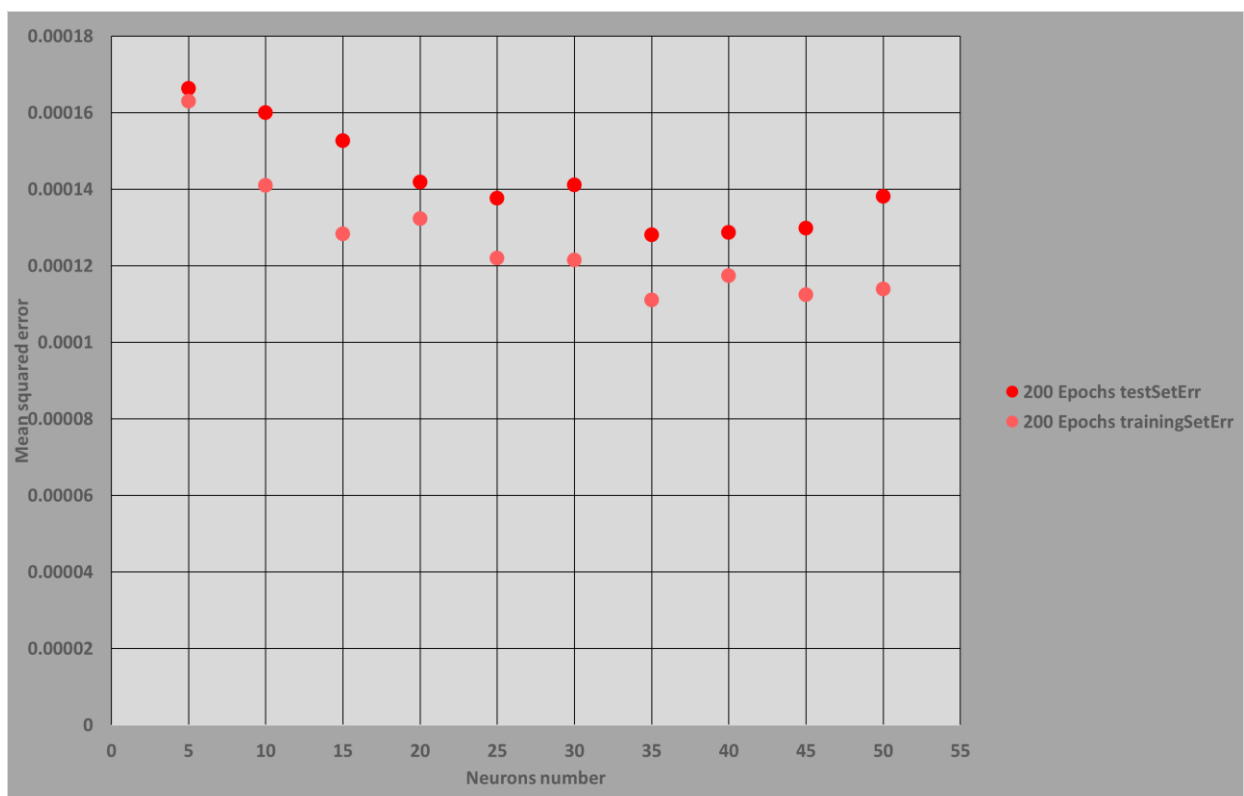
1. **Training Epochs [-]**: numero di epoche massime di addestramento.
2. **Requested training time [s]**: tempo in secondi necessario per l’addestramento.
3. **Neurons Number [-]**: numero di percettroni della rete.
4. **testSet Mean Squared Error [-]**: errore quadratico medio sul test set.
5. **trainingSet Mean Squared Error [-]**: errore quadratico medio sul training set.

6. **Percentage distance [%]**: distanza percentuale tra l'errore quadratico medio sul test set e l'errore quadratico medio sul training set, calcolato attraverso la seguente formula:

$$distance_{\%} = \left(\frac{MSE_{testSet}}{MSE_{trainSet}} - 1 \right) * 100$$

7. **Mean percentage distance [%]**: distanza percentuale media tra l'errore quadratico medio sul test set e l'errore quadratico medio sul training set.

Costruisco un grafico scatter utilizzando i dati contenuti della precedente tabella:

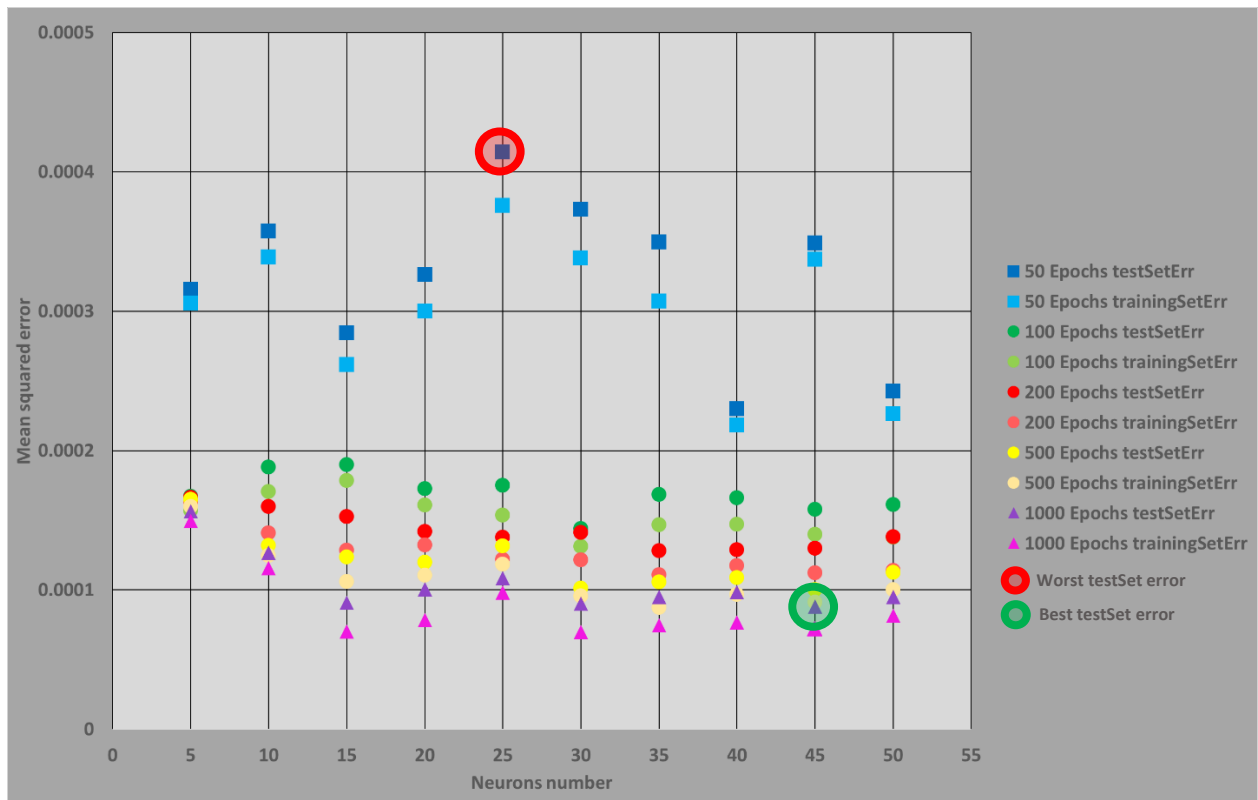


Riporto le seguenti osservazioni:

1. Per ogni rete neurale, l'errore dei risultati sul set di addestramento è sempre inferiore all'errore dei risultati sul set di test.

2. All'aumentare del numero di percettroni della rete neurale l'errore dei risultati tende a decrescere sia per il test set che per il training set.
3. All'aumentare del numero di percettroni della rete neurale la distanza percentuale tra l'errore del risultato sul test set e l'errore del risultato sul training set tende ad aumentare (tendenza all'overfitting dei dati).

Costruisco un grafico scatter con tutti i dati raccolti nella presente esercitazione:



Riporto le seguenti osservazioni:

1. All'aumentare delle epoche di addestramento gli errori sul test set e sul training set diminuiscono.
2. All'aumentare delle epoche di addestramento la distanza media percentuale tra l'errore quadratico medio sul test set e l'errore quadratico medio sul training set tende a crescere (tendenza all'overfitting dei dati).

Utilizzando una tupla estratta preliminarmente dal dataset originale e non appartenente né al training set né al test set, provo a mettere in relazione i risultati ottenuti utilizzando la rete con il migliore e il peggiore risultato in termini di errore sul test set.

La tupla in esame è la seguente:

attr1	attr2	attr3	attr4	result
2.5	0.75	12.48	5	544.48999

Attraverso lo script *networkExecute.py* fornisco in input alle reti neurali i dati sopra riportati e confronto i risultati:

```
There are 2 neural network ready to load:
1) 25NeuronsNet(50epochsworstNetwork)
2) 45NeuronsNet(1000 epochsbestNetwork)
Witch network to load?
1
Put four attributes:
attr1:
2.5
attr2:
0.75
attr3:
12.48
attr4:
5
network output: 778.1871197820251
```

```
There are 2 neural network ready to load:
1) 25NeuronsNet(50epochsworstNetwork)
2) 45NeuronsNet(1000 epochsbestNetwork)
Witch network to load?
2
Put four attributes:
attr1:
2.5
attr2:
0.75
attr3:
12.48
attr4:
5
network output: 538.4291504546237
```

Di seguito riporto la tabella con le informazioni importanti:

net [-]	Result [-]	Target [-]	Error [%]
45NeuronsNet(1000epochs)	538.429	544.48999	1.13
25NeuronsNet(50epochs)	778.187		30.03

example2

example2 si riferisce al dataset “wizmir.txt” che, dopo la prima fase di data mining, presenta le seguenti caratteristiche:

```
dataset size: 1459 tuples
trainingSet absolute size: 1021 tuples
trainingSet percentage size: 70.0 %
testSet absolute size: 438 tuples
testSet percentage size: 30.0%
```

Riporto di seguito i dati relativi ad una sessione di addestramento della rete neurale, estratto della tabella presente nel foglio di calcolo Results.xlsx.

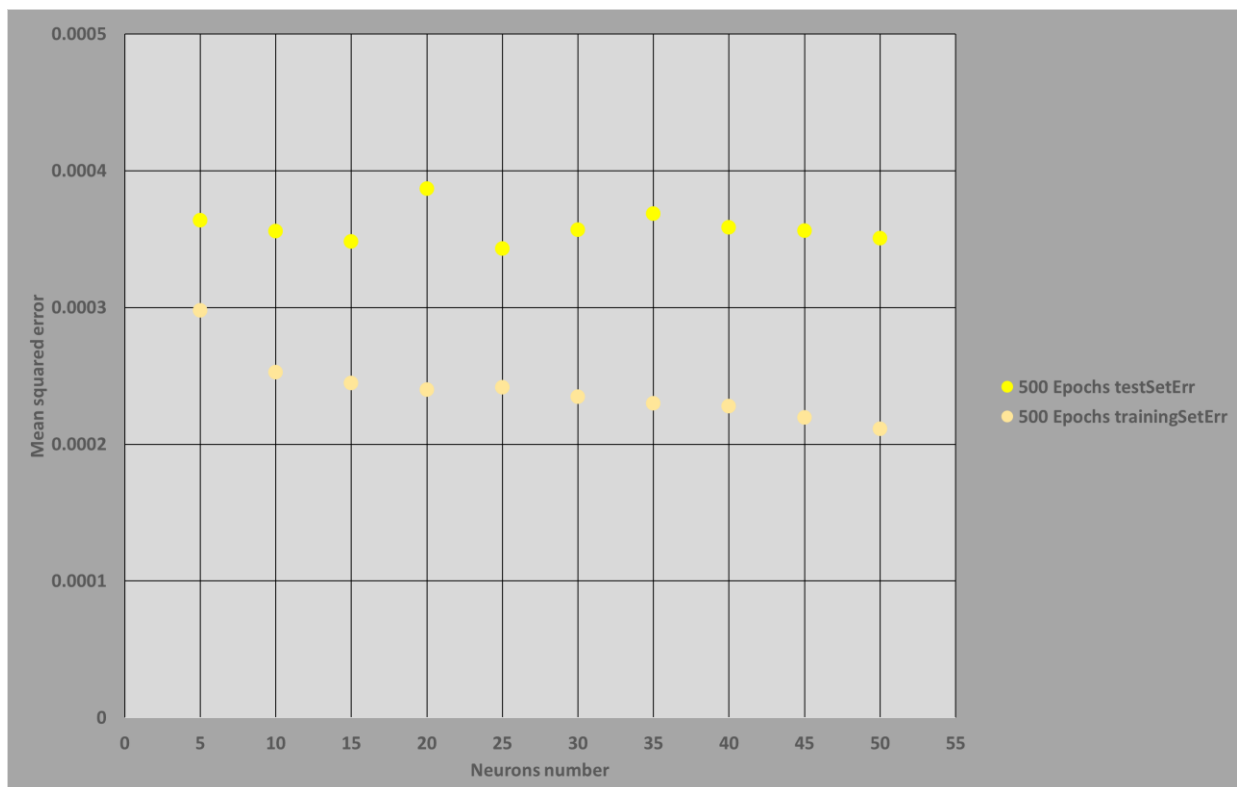
Training Epochs [-]	requested training time [s]	Neurons Number [-]	testSet Mean Squared Error [-]	trainingSet Mean Squared Error [-]	Percentage distance [%]	Mean percentage distance [%]
500	5.94E+01	10	3.64E-04	2.98E-04	2.22E+01	5.06E+01
	6.03E+01	15	3.56E-04	2.53E-04	4.08E+01	
	6.02E+01	20	3.48E-04	2.45E-04	4.23E+01	
	6.22E+01	25	3.87E-04	2.40E-04	6.13E+01	
	6.34E+01	30	3.43E-04	2.42E-04	4.19E+01	
	6.37E+01	35	3.57E-04	2.35E-04	5.21E+01	
	6.67E+01	40	3.68E-04	2.30E-04	6.03E+01	
	6.98E+01	45	3.58E-04	2.28E-04	5.73E+01	
	7.24E+01	50	3.56E-04	2.20E-04	6.22E+01	
	7.59E+01	5	3.51E-04	2.11E-04	6.59E+01	

8. **Training Epochs [-]**: numero di epoche massime di addestramento.
9. **Requested training time [s]**: tempo in secondi necessario per l’addestramento.
10. **Neurons Number [-]**: numero di percettroni della rete.
11. **testSet Mean Squared Error [-]**: errore quadratico medio sul test set.
12. **trainingSet Mean Squared Error [-]**: errore quadratico medio sul training set.
13. **Percentage distance [%]**: distanza percentuale tra l’errore quadratico medio sul test set e l’errore quadratico medio sul training set, calcolato attraverso la seguente formula:

$$distance_{\%} = \left(\frac{MSE_{testSet}}{MSE_{trainingSet}} - 1 \right) * 100$$

14. **Mean percentage distance [%]:** distanza percentuale media tra l'errore quadratico medio sul test set e l'errore quadratico medio sul training set.

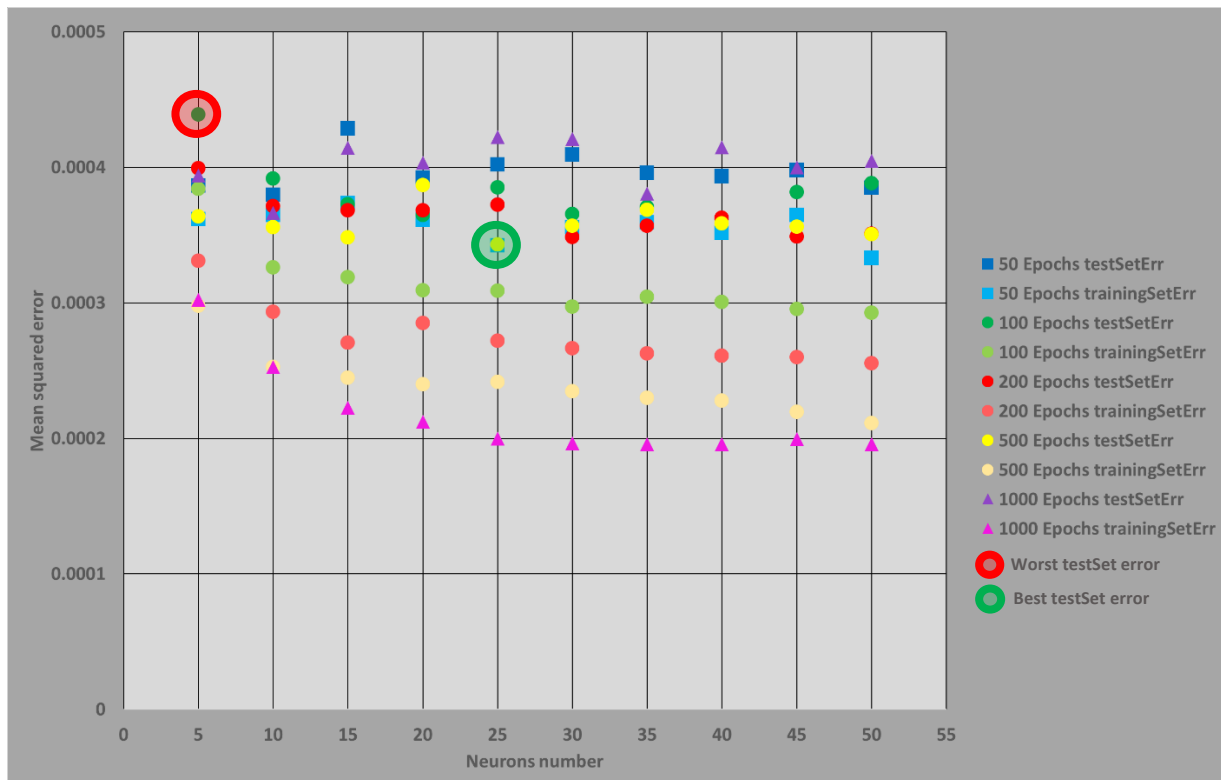
Costruisco un grafico scatter utilizzando i dati contenuti della precedente tabella:



Riporto le seguenti osservazioni:

- Per ogni rete neurale, l'errore dei risultati sul set di addestramento è sempre inferiore all'errore dei risultati sul set di test.
- All'aumentare del numero di percettroni della rete neurale l'errore dei risultati tende a decrescere per il training set ed è altalenante per il test set.
- All'aumentare del numero di percettroni della rete neurale la distanza percentuale tra l'errore del risultato sul test set e l'errore del risultato sul training tende ad aumentare.

Costruisco un grafico scatter con tutti i dati raccolti nella presente esercitazione:



Riporto le seguenti osservazioni:

3. All'aumentare delle epoche di addestramento gli errori sul training set diminuiscono mentre gli errori sul test set, dopo una prima fase di discesa tendono ad aumentare.
4. All'aumentare delle epoche di addestramento la distanza media percentuale tra l'errore quadratico medio sul test set e l'errore quadratico medio sul training set tende a crescere (tendenza all'overfitting dei dati).

Utilizzando due tuple estratte preliminarmente dal dataset originale e non appartenente né al training set né al test set, provo a mettere in relazione i risultati ottenuti utilizzando la rete con il migliore e il peggiore risultato in termini di errore sul test set.

Le tuple in esame sono le seguenti:

attr1	attr2	attr3	attr4	attr5	attr6	attr7	attr8	attr9	result
57.2	31.5	39.0	0.0	30.22	7.0	7.6	18.3	34.28	46.8
93.2	63.0	56.5	0.0	29.94	7.4	10.9	20.8	34.28	80.3

Attraverso lo script networkExecute.py fornisco in input alle reti neurali i dati sopra riportati e confronto i risultati:

```
There are 2 neural network ready to load:
1) 25NeuronsNet(500epochsbestNetwork)
2) 5NeuronsNet(100epochsworstNetwork)
Witch network to load?
1
Put nine attributes:
attr1:
57.2
attr2:
31.5
attr3:
39
attr4:
0
attr5:
30.22
attr6:
7
attr7:
7.6
attr8:
18.3
attr9:
34.28
network output: 45.6369596473536
```

```
There are 2 neural network ready to load:
1) 25NeuronsNet(500epochsbestNetwork)
2) 5NeuronsNet(100epochsworstNetwork)
Witch network to load?
2
Put nine attributes:
attr1:
57.2
attr2:
31.5
attr3:
39
attr4:
0
attr5:
30.22
attr6:
7
attr7:
7.6
attr8:
18.3
attr9:
34.28
network output: 44.906680441258565
```

There are 2 neural network ready to load:

1) 25NeuronsNet(500epochsbestNetwork)

2) 5NeuronsNet(100epochsworstNetwork)

Witch network to load?

1

Put nine attributes:

attr1:

93.2

attr2:

63

attr3:

56.5

attr4:

0

attr5:

29.94

attr6:

7.4

attr7:

10.9

attr8:

20.8

attr9:

34.28

network output: 79.41032787655729

There are 2 neural network ready to load:

1) 25NeuronsNet(500epochsbestNetwork)

2) 5NeuronsNet(100epochsworstNetwork)

Witch network to load?

2

Put nine attributes:

attr1:

93.2

attr2:

63

attr3:

56.5

attr4:

0

attr5:

29.94

attr6:

7.4

attr7:

10.9

attr8:

20.8

attr9:

34.28

network output: 78.97162968065656

Di seguito riporto la tabella con le informazioni importanti:

input [-]	net [-]	Result [-]	Target [-]	Error [%]
Tuple1	25NeuronsNet(500epochs)	45.637	46.8	2.48
	5NeuronsNet(100epochs)	44.907		4.04
Tuple2	25NeuronsNet(500epochs)	79.410	80.3	1.12
	5NeuronsNet(100epochs)	78.972		1.68

Conclusioni

I due test set esaminati hanno caratteristiche molto diverse:

Nel primo (ele-2.txt) all'aumentare delle epoche di addestramento si assiste ad un costante miglioramento dei risultati della rete neurale e sembra non esserci una grossa tendenza all'overfitting, probabilmente perché il training set utilizzato è molto ben rappresentativo del contesto.

Nel secondo (wizmir.txt) sono necessarie poche epoche di addestramento per ottenere dei risultati accettabili. All'aumentare di quest'ultime si nota chiaramente un'affezione della rete neurale ai dati di training con un conseguente fenomeno di overfitting.

Le reti neurali migliori e peggiori da me selezionate per ogni dataset sono disponibili nelle corrispondenti cartelle /results. Possono essere eventualmente testate attraverso l'utilizzo dello script `networkExecute.py`.