

Documentazione Progetto Machine Learning mod. B

Corso di Machine Learning mod. B - Prof. R. Prevete

Studenti: Andrea Sorrentino N97/221

Alessandro Serrapica N97/213

Orlando Aprea N97/226

14 dicembre 2016

Indice

1	Introduzione	1
2	Struttura dell'applicazione	1
2.1	Main GUI	1
2.2	Validazione dati e gestione degli errori	4
2.3	Output	5
3	Reti neurali	7
3.1	Cos'è una rete neurale	7
3.2	Training - backpropagation	8
4	Creazione di una rete	11
4.1	Strutture dati per la gestione di una rete	11
4.1.1	Dataset	11
4.1.2	Gradient	12
4.1.3	Results	13
4.1.4	Net	13
5	Testing	15
5.1	Test1	15
5.1.1	Batch	16
5.1.2	Online	17
5.2	Test2	19
5.2.1	Batch	19
5.2.2	Online	21
5.3	Test3	22
5.3.1	Batch	23
5.3.2	Online	24
5.4	Test4	26
5.4.1	Batch	26
5.4.2	Online	28
5.5	Test5	29
5.5.1	Batch	30
5.5.2	Online	31
6	Source code	34
6.1	NewNet	34
6.2	ForwardPropagation	35
6.3	BackPropagation	36
6.4	GradientDescent	38
6.5	Batch	38

6.6	Online	41
6.7	VerifyNet	43

1 Introduzione

Il progetto assegnato durante il corso é strutturato in due sezioni principali. La prima parte é comune a tutti i gruppi e riguarda la progettazione ed implementazione di funzioni per simulare la propagazione in avanti di una rete neurale multi-strato con almeno due strati di pesi, con la sigmoide come funzione di output dei nodi interni e l'identitá come funzione di output dei nodi di output.

(FACOLTATIVO: permettere all'utente di implementare reti con piú di uno strato di nodi interni e con qualsiasi funzione di output per ciascuno strato).

Oltre a ciò la prima parte prevede anche la progettazione ed implementazione di funzioni per la realizzazione della back-propagation per reti neurali multi-strato con almeno due strati di pesi, con la sigmoide come funzione di output dei nodi interni e l'identitá come funzione di output dei nodi di output, con la somma dei quadrati come funzione di errore.

(FACOLTATIVO: permettere all'utente di realizzare la back-propagation con piú di uno strato di nodi interni, con qualsiasi funzione di output per ciascun strato e con qualsiasi funzione di errore derivabile rispetto all'output).

La seconda parte, assegnata nello specifico agli studenti Aprea, Serrapica, Sorrentino, é inerente lo studio dell'apprendimento della rete neurale al variare dei nodi interni, fissato come algoritmo di aggiornamento dei pesi la discesa del gradiente con momento,(i.e. scegliere e mantenere invariati tutti gli altri "parametri" come dataset, funzioni di output e funzione di errore).

2 Struttura dell'applicazione

L'applicazione é stata progettata e sviluppata cercando di adottare quei requisiti di modularitá e basso accoppiamento tra le varie funzionalitá che sono alla base di qualsiasi software ben strutturato.

In particolare, si possono distinguere quattro moduli principali:

- Funzioni per la gestione dell'interfaccia utente.
- Funzioni per la creazione/modifica della rete.
- Funzioni per l'addestramento della rete.
- Funzioni per la verifica della bontá dei risultati della rete.

In questa sezione sará sviluppato il primo punto, perlopiú da un punto di vista qualitativo, dato che le funzionalitá inerenti la UI non erano richieste nella traccia iniziale. Inoltre, una spiegazione approfondita delle funzioni di validazione dei campi di un'interfaccia non farebbe altro che tediare il lettore con informazioni inutili allo scopo del progetto. In tal senso quanto segue puo' anche essere inteso come una sorta di manuale utente.

2.1 Main GUI

Per accedere alla Main GUI é necessario caricare il progetto in matlab e digitare nella console il comando '*run main*'. Il risultato di tale operazione é la visualizzazione schermo della seguente interfaccia:

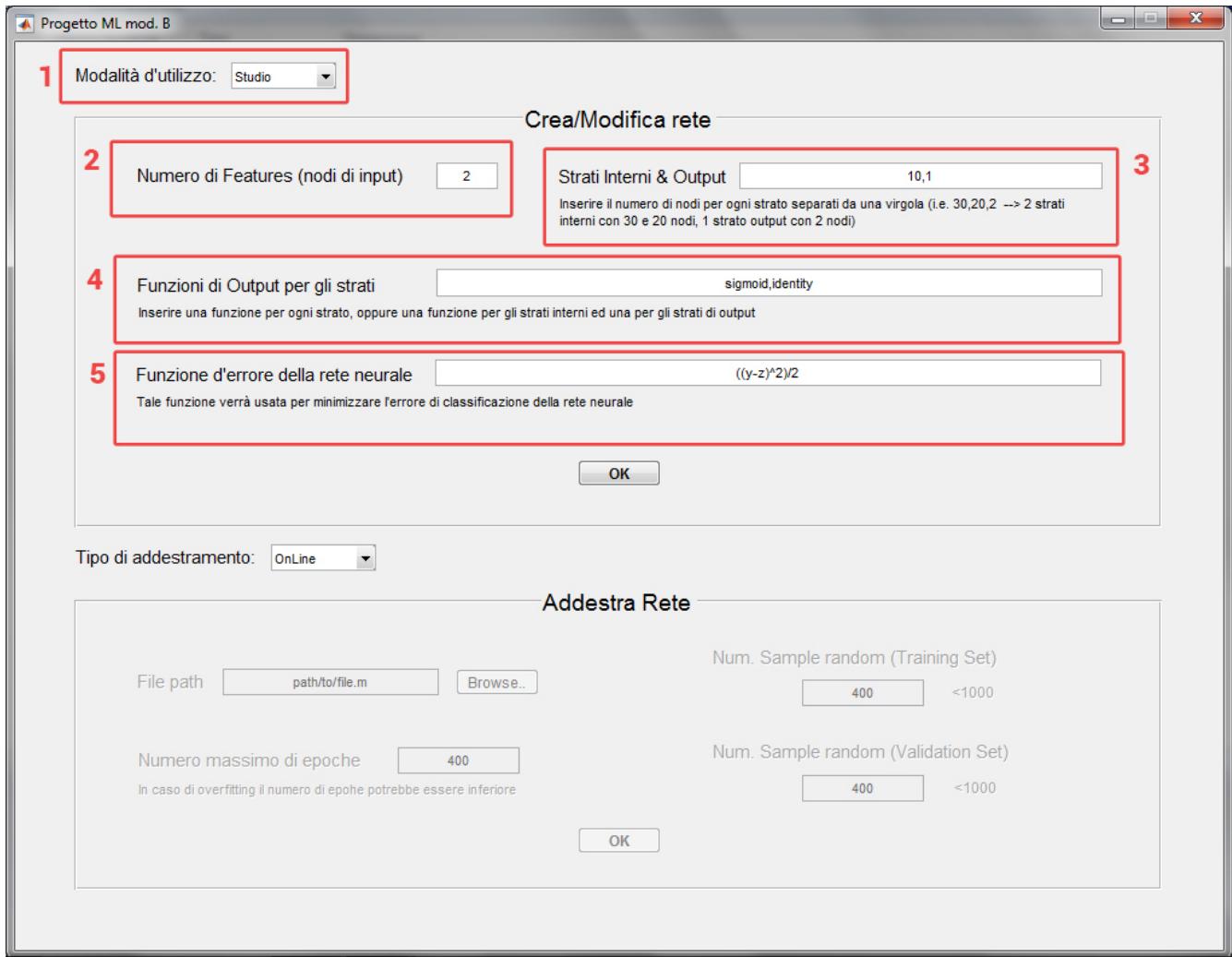


Figura 1: Main GUI - creazione rete

Il primo campo consente di alternare la modalità di utilizzo dell'applicazione tra due scelte possibili:

- Studio
- Utilizzo libero

Nello specifico, la modalità ‘Studio’ consente di effettuare test ripetuti sullo stesso dataset (lasciando training set e validation set invariati) e permette di modificare solo la conformazione interna della rete (i.e. strati interni e funzioni associate agli stessi), come previsto dalla traccia. ‘L’utilizzo libero’ consente invece di cambiare tutti i parametri della GUI in ogni momento, in questo caso il training set ed il validation set sono diversi ad ogni utilizzo. Il secondo parametro rappresenta il numero di nodi di input della rete, che deve essere coerente con il numero di features del dataset in ingresso.

Il terzo valore che è possibile inserire nella GUI riguarda il numero di strati della rete e, per ogni strato della stessa, il corrispondente numero di nodi (i.e. un valore 10,1 genera una rete con 10 nodi per lo strato interno ed uno per lo strato di output).

Il quarto campo indica le funzioni di output da associoare ad ogni strato della rete, é possibile inserire due funzioni (una per gli strati interni e una per quello di output), oppure una funzione per ogni strato della rete.

L'ultimo parametro della prima parte della GUI consente all'utente di specificare la funzione di errore da usare durante il processo di apprendimento, la derivata di tale funzione sará poi calcolata in automatico da matlab.

Una volta impostati tutti i parametri, con un click sul pulsante ‘OK’, si da luogo alla creazione della rete e si riceve il seguente feedback:

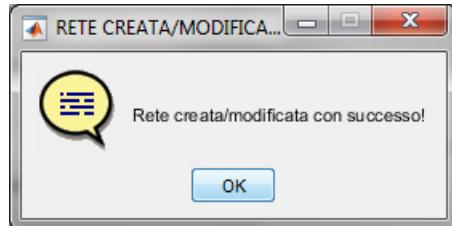


Figura 2: Main GUI - feedback creazione rete

A questo punto si attiva la seconda parte della GUI, quella relativa all'apprendimento, come é possibile osservare nell'immagine seguente.

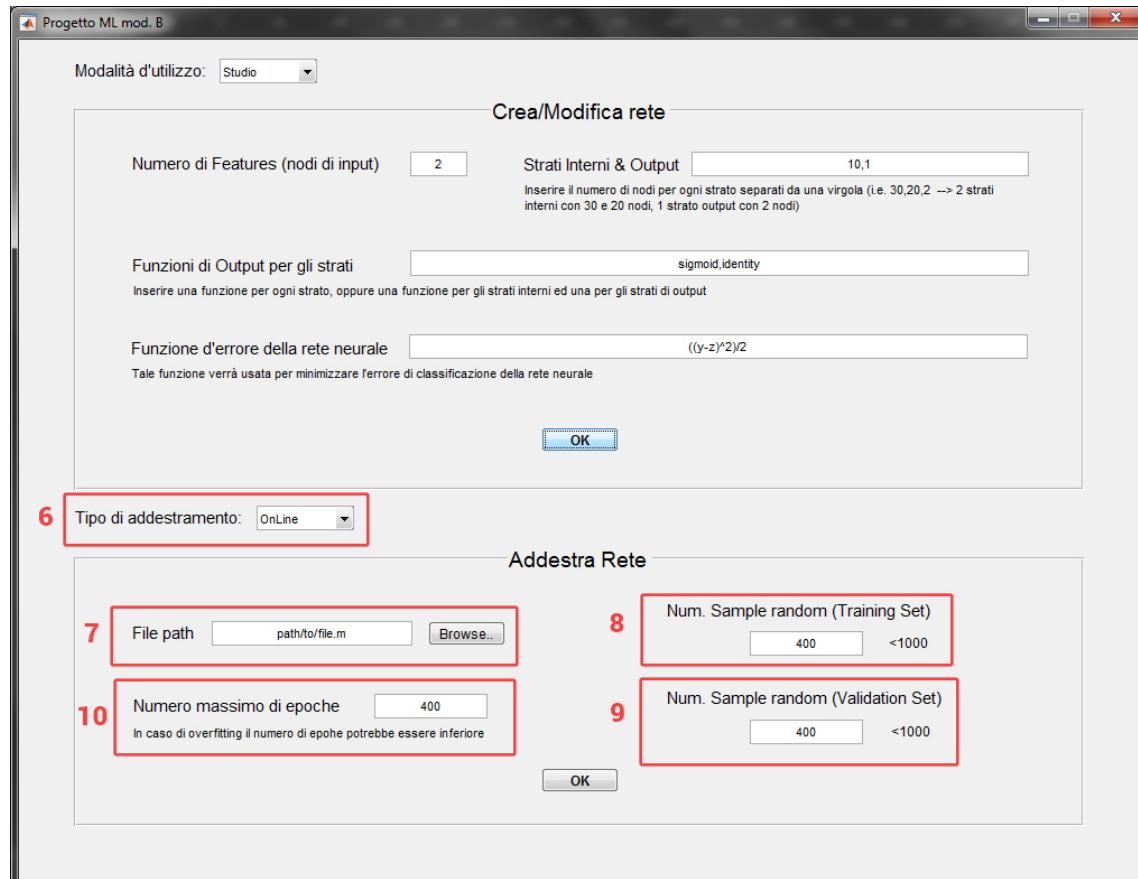


Figura 3: Main GUI - addestramento rete

Il parametro numero 6 é un menú a tendina che consente di selezionare la modalità di apprendimento tra:

- Online
- Batch

In ogni caso, come richiesto dalla traccia, l'applicazione userá un approccio all'apprendimento basato sulla discesa del gradiente con momento.

Il settimo campo é un semplice file browser che consente di selezionare il dataset.

I valori 8 e 9 sono rispettivamente il numero di elementi che vengono estratti come permutazione random dal dataset per formare il training set ed il validation set.

L'ultimo parametro (il decimo), invece, indica il numero di epoches da utilizzare per l'apprendimento.

Una volta avviato l'addestramento, viene mostrata a video una progress bar che consente all'utente di comprendere lo stato di avanzamento del processo. Al termine del training della rete sono mostrate altre due schermate con i risultati ottenuti, e sulle quali si fornisce un approfondimento nella sezione 2.3.

2.2 Validazione dati e gestione degli errori

Cosa succede se si inseriscono dei dati non corretti, oppure non coerenti, nella GUI?

A questo proposito sono state effettuate le opportune verifiche. In particolare, grazie all'uso di espressioni regolari, é stato possibile controllare che tutti i campi siano non vuoti e che l'utente rispetti il formato dell'input dei singoli parametri. Nello specifico i campi 2,8,9,10 devono essere degli interi, mentre il terzo deve essere del tipo '*numero1,...,numeroN*'. Il quarto parametro della GUI deve essere del tipo '*funzione1,...,funzioneN*' (con un numero di funzioni pari a due, oppure pari al numero di strati specificati nel secondo campo).

Oltre ai controlli di tipo sintattico é stato necessario assicurarsi che i parametri, spesso collegati fra loro a livello logico, fossero coerenti. Un esempio lampante di questa situazione si ha quando il numero di nodi di input (campo 2) non corrisponde al numero di features del dataset caricato (campo 7), in questo caso é necessario che l'utente provveda ad un cambiamento di uno dei due valori in modo da soddisfare il requisito logico di base. Similmente, il check sul campo 4 consente di comprendere se l'utente abbia inserito una funzione valida (built-in o file.m) o meno. Di seguito sono state raggruppate le varie schermate di errore che possono apparire all'interno del programma.

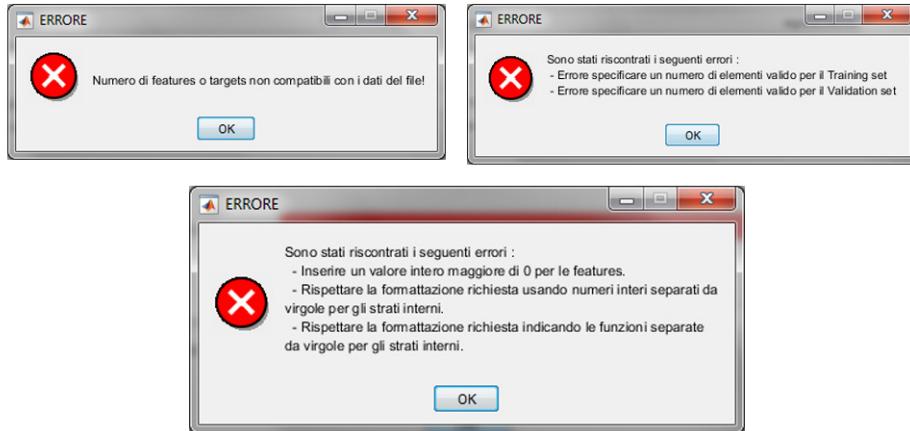


Figura 4: Main GUI - addestramento rete

Ovviamente per fornire all'utente un feedback visivo opportuno, i campi errati sono colorati di rosso, come si puo' notare nell'immagine seguente.

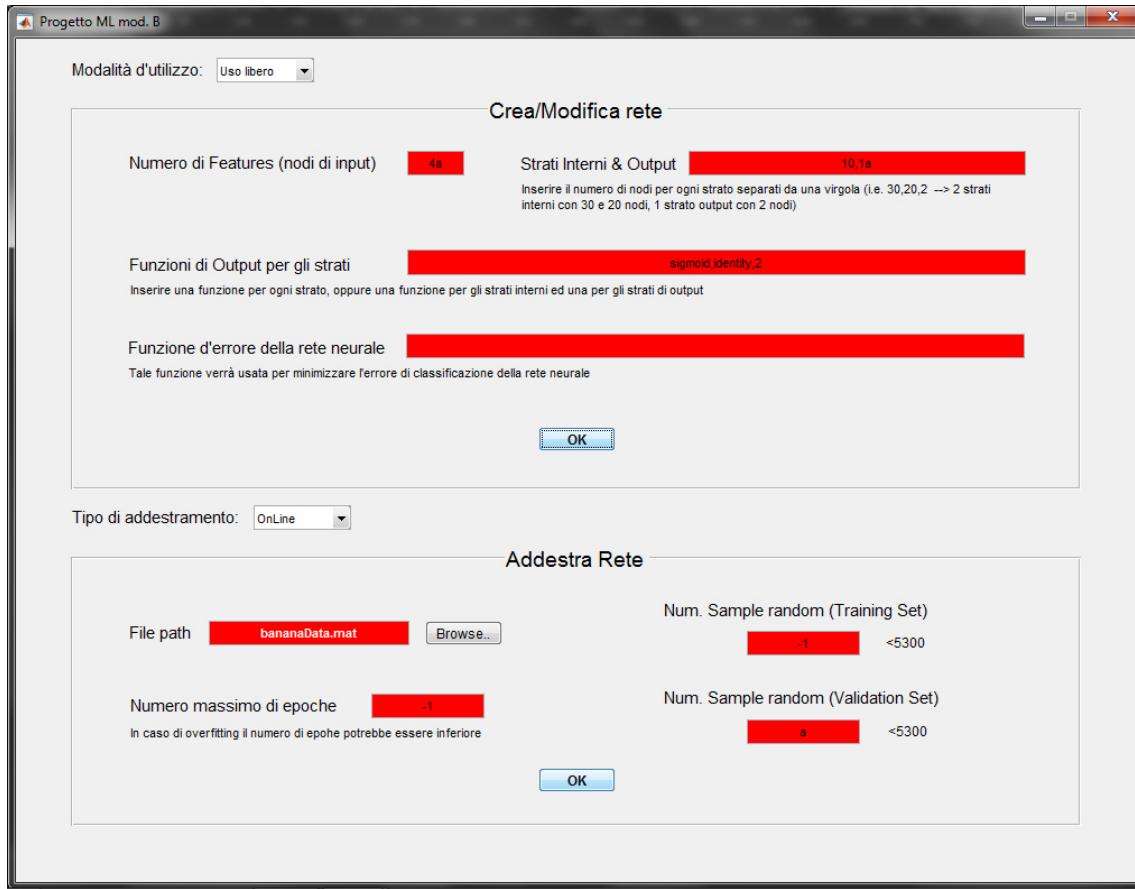


Figura 5: Main GUI - schermata di errore

2.3 Output

Una volta terminato il processo di apprendimento, l'applicazione mostrerà a video i risultati dell'operazione. In particolare, sarà visualizzata una figura contenente il plot dei punti del training set e del validation set, oltre alla curva dell'errore che decresce all'aumentare delle epoche(Figura 6). Per comprendere la bontá dell'apprendimento però non basta soltanto visualizzare la curva dell'errore, é necessario analizzare anche dei dati statistici piú concreti (Figura 7), tra cui:

- Precision
- Recall
- Accuracy
- Specificity

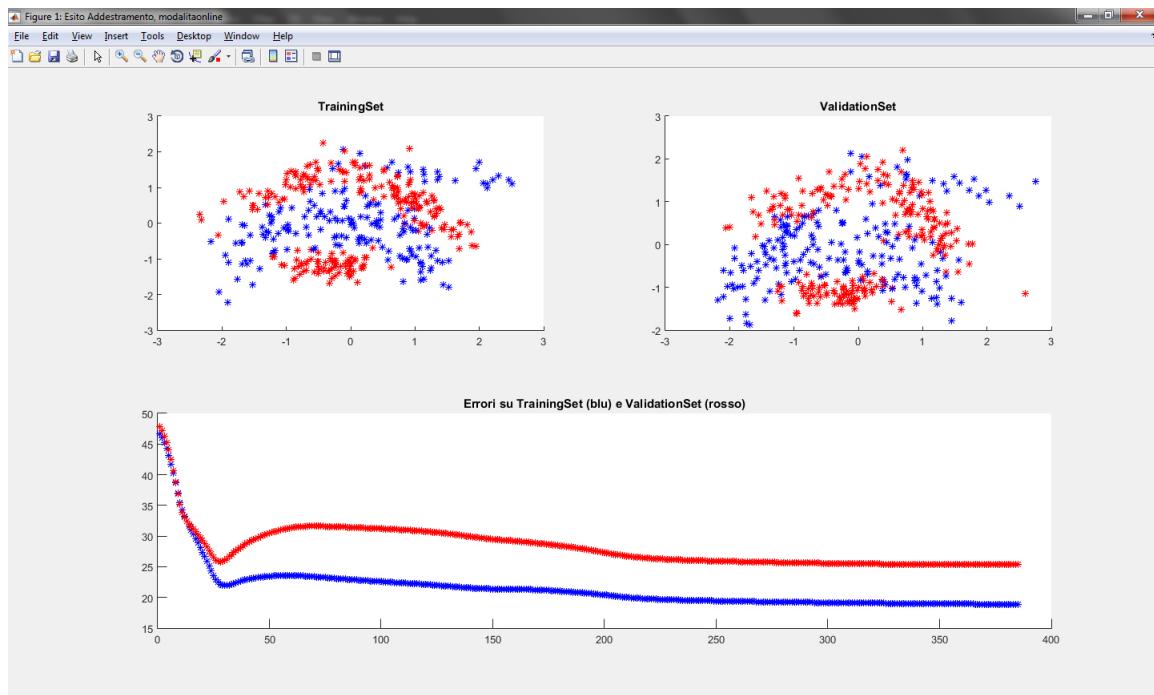


Figura 6: Plot figure - schermata di output

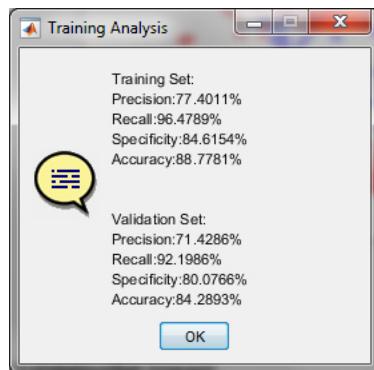


Figura 7: finestra dati statistici

Per fornire un indicazione, anche visiva, che faccia comprendere ad un primo sguardo l'effettiva qualità della classificazione che la rete addestrata è in grado di effettuare, verranno mostrati gli elementi di un test set, mettendo a confronto l'etichettatura originaria con la classificazione effettuata dalla rete(Figura 8).

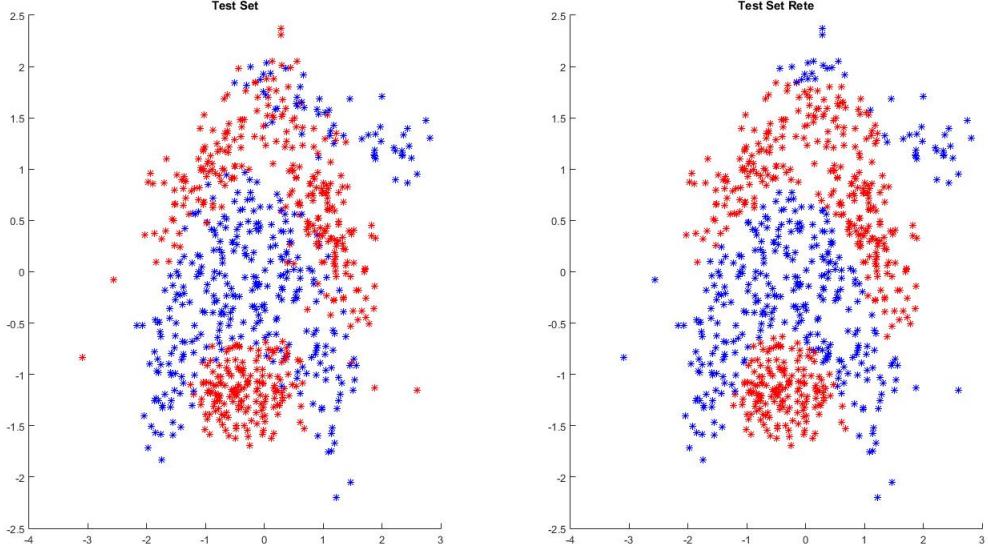


Figura 8: finestra confronto classificazione

3 Reti neurali

3.1 Cos'è una rete neurale

Una rete neurale consente di risolvere una moltitudine di problemi, tra cui riveste un ruolo di primo piano quello della classificazione di elementi di un determinato dominio rispetto ad un insieme di classi. Possiamo dunque definire un problema di classificazione come l'individuazione di una funzione che sia in grado di mappare elementi di uno spazio d -dimensionale su elementi di uno spazio c -dimensionale. Nello specifico vogliamo quindi individuare una funzione che abbia le seguenti caratteristiche:

$$f : x \in R^d \rightarrow f(x) \in R^c$$

In generale però, tale funzione è solo parzialmente conosciuta, o addirittura, del tutto sconosciuta.

Quindi lo scopo di un problema di classificazione si riduce all'approssimazione della funzione $f(x)$ descritta, mediante una funzione parametrizzata $g(x; w)$ così descritta:

$$g : x \in R^d, w \in R^p \rightarrow g(x; w) \in R^c$$

Dunque per approssimare f è necessario riuscire a comprendere la funzione g , e ciò equivale ad individuare i parametri w . Il procedimento che consente di ottenere i valori dei parametri w è detto training. Dunque, come è fatta nello specifico una rete neurale (feed forward, full connected, con funzione di attivazione per i nodi interni di tipo sigmoidale)? La risposta è nella seguente figura riassuntiva. La descrizione nello specifico dei dettagli di una rete neurale esula dallo scopo della presente documentazione, ci si soffermerà maggiormente sulle modalità di training.

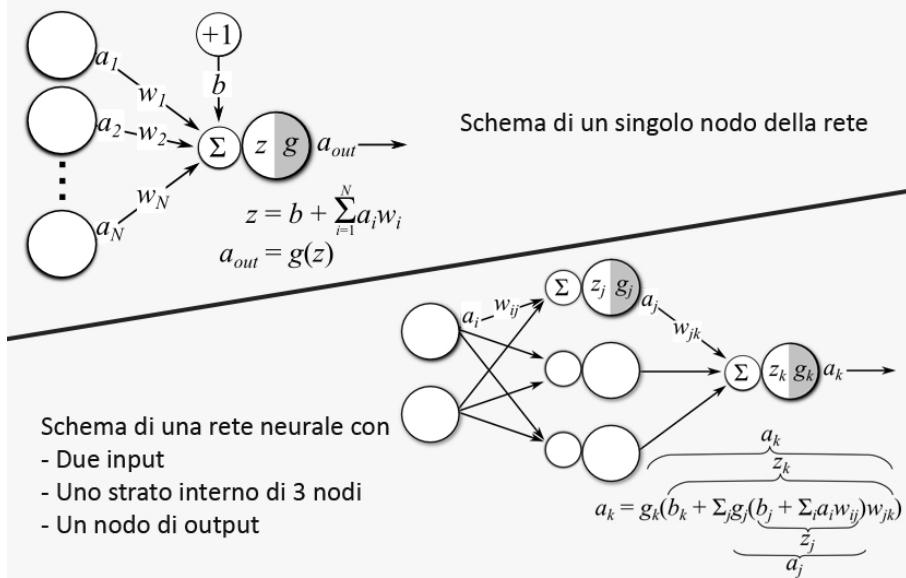


Figura 9: Schemi del singolo nodo e di una rete d'esempio

3.2 Training - backpropagation

Una volta fatta questa breve disamina dei concetti alla base di una rete neurale é possibile passare a discutere la fase di training della rete. Addestrare una rete neurale significa sostanzialmente determinare i parametri della stessa che minimizzano l'errore commesso. Quindi c'è bisogno, innanzitutto, di un modo per quantificare l'errore commesso. Un modo standard di operare in questo senso, é quello di affidarsi ad una funzione di errore già ampiamente usata in letteratura, ad esempio l'errore quadratico medio:

$$E = 1/N \sum (\text{output} - \text{target})^2$$

Data una funzione di errore, l'obiettivo é trovare i parametri che la minimizzano. Un metodo efficiente per raggiungere questo scopo é quello di utilizzare la discesa del gradiente. Per applicare la discesa del gradiente ad una rete neurale si fa uso di un algoritmo noto come backpropagation. Tale algoritmo si puó suddividere in 3 step principali:

- Forward-propagation dei valori di input
- Back-propagation dei valori di errore
- Discesa del gradiente

Forward-propagation Per poter determinare l'errore della rete si necessita in primo luogo di propagare i valori di input attraverso gli strati della rete. Questo ci consente di ottenere i valori di output della rete (utili poi in seguito), come nella figura seguente. Da quanto visto in precedenza si deduce che una rete neurale effettua sostanzialmente un mapping non lineare tra uno spazio d-dimensionale ed uno spazio c-dimensionale, con parametri dati dai pesi della rete.

Back-propagation Il secondo step dell'algoritmo consiste nel calcolare l'errore di output e propagarlo indietro fino allo strato dei valori di input. Questo é fatto calcolando le derivate della funzione di errore rispetto ai pesi.

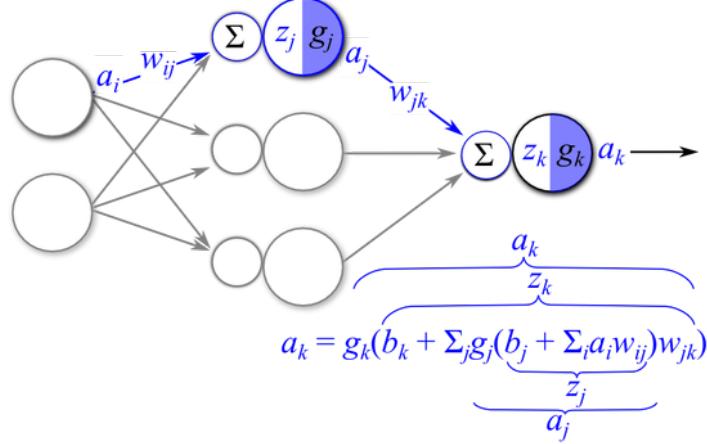


Figura 10: Forward-propagation

Per calcolare $\frac{\delta E}{\delta w_{ij}}$, siccome $E = \Sigma_n E^n$, possiamo concentrarci sul calcolo di $\frac{\delta E^n}{\delta w_{ij}}$. Assumiamo, quindi, di usare una funzione di errore valida (ovvero, una funzione differenziabile delle variabili di output), ad esempio l'errore quadratico medio.

$$E^n = 1/N \sum (a_k - t_k)^2$$

Per calcolare $\frac{\delta E^n}{\delta w_{ij}}$ osserviamo che E^n dipende da w_{ij} attraverso z_j . È possibile dimostrare che:

$$\frac{\delta E^n}{\delta w_{ij}} = \delta_j a_i$$

con

$$\delta_j = \frac{\delta E^n}{\delta z_j}$$

Quanto detto ci consente di affermare che è possibile ottenere la derivata richiesta semplicemente moltiplicando il valore di output del nodo da cui parte la connessione associata al peso w_{ij} per un certo valore δ_j associato al nodo in cui termina la connessione con peso w_{ij} . Come calcolare dunque δ_j ? Per i nodi di output della rete si ha:

$$\delta_k = \frac{\delta E^n}{\delta z_k} = g'_k(z_k) \frac{\delta E^n}{\delta a_k} = g'_k(z_k) E'(a_k, t_k) = g'_k(z_k)(a_k - t_k)$$

Per i nodi restanti abbiamo che:

$$\delta_j = \frac{\delta E^n}{\delta z_j} = g'_j(z_j) \sum_k \delta_k w_{jk}$$

Per le reti che hanno più di uno strato interno questo procedimento si applica in modo iterativo ai livelli $j - 1, j - 2, \dots, etc.$ Effettuando le opportune sostituzioni otteniamo che

$$\frac{\delta E^n}{\delta w_{ij}} = \delta_j a_i$$

Ciò ci consente di avere a disposizione una formula ricorsiva che a partire dai nodi di output ci consente di risalire al valore δ_j di tutti i nodi interni.

Discesa del gradiente La discesa del gradiente, si basa sui risultati ottenuti dalla back-propagation (ovvero sul calcolo delle derivate della funzione d'errore rispetto ai pesi) e li sfrutta per calcolare i nuovi pesi della rete. È

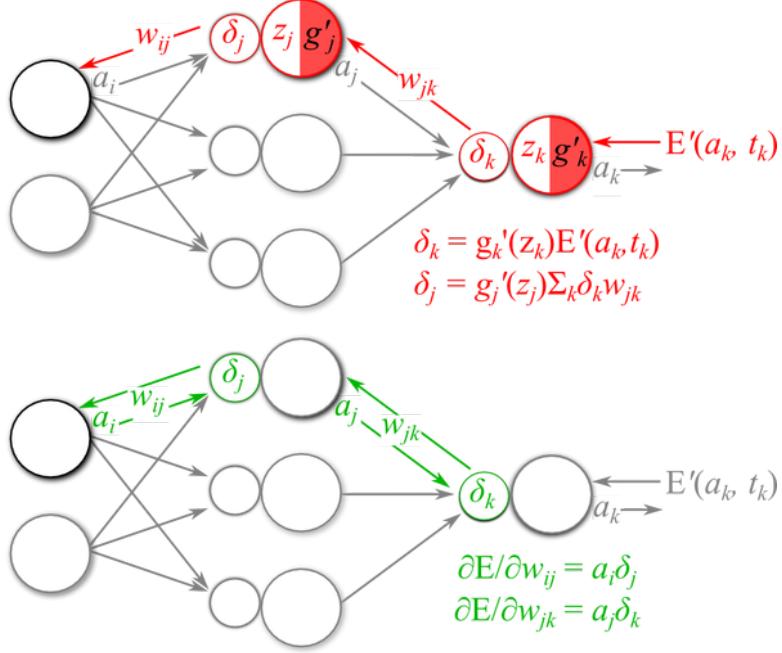


Figura 11: Back-propagation

fondamentale sottolineare che le due fasi sono distinte. In particolare l'aggiornamento dei pesi avviene nel seguente modo:

$$w_{ij} = w_{ij} - \eta \frac{\delta E}{\delta w_{ij}}$$

Discesa del gradiente con momento Per velocizzare il processo di addestramento dei parametri della rete è possibile applicare un momento alla discesa del gradiente. Possiamo, ad esempio, calcolare la variazione del peso come dipendente dalla variazione precedente, cioé:

$$\Delta w_{ij}^{(t)} = -\eta \delta E / \delta w_{ij} + \mu \Delta w_{ij}^{(t-1)}$$

Dove μ è detto coefficiente del momento e $\mu \Delta w_{ij}^{(t-1)}$ è il momento applicato al gradiente, dipendente dalla variazione di peso precedente. La discesa del gradiente con momento, pone il problema di come scegliere η . Per l'elaborato presentato si è scelto di non fissare η a priori, ciò per rendere l'algoritmo di apprendimento il più adattivo possibile. In particolare sono state utilizzate le due costanti ausiliarie ρ e σ (impostate rispettivamente con valori leggermente superiori a 1 e compreso tra 0 e 1) come parametri con i quali moltiplicare η ad ogni iterazione dell'algoritmo. Nel caso di variazione d'errore positiva (errore in aumento) l'influenza di η viene diminuita moltiplicandola per σ (l'apprendimento non è efficace), nel caso di variazione d'errore negativa (errore in diminuzione) l'influenza di η viene aumentata moltiplicandola per ρ (l'apprendimento è efficace e dunque se ne aumenta la velocità).

4 Creazione di una rete

Nelle sezioni seguenti viene esaminato il processo di creazione di una rete e delle strutture dati necessarie alla sua corretta gestione nel programma.

4.1 Strutture dati per la gestione di una rete

Per gestire e riportare tutte le informazioni utili relative allo svolgimento dell'addestramento della rete, sono state implementate le seguenti strutture dati:

- dataset
- trainingset
- validationset
- net
- gradient
- results

4.1.1 Dataset

Il dataset è la struttura chiave che risulta essere alla base del processo di addestramento della rete e della verifica della bontà della classificazione eseguita dalla stessa. Infatti, all'interno del dataset vengono immagazzinati tutti i dati (estrapolati dal file precedentemente caricato tramite GUI) su cui la rete dovrà addestrarsi a funzionare. Nel caso del bananadataset, utilizzato per tale progetto, i dati constano di due coordinate identificatrici di un punto e un'etichetta che discrimina il punto in base all'appartenenza o meno alla figura.

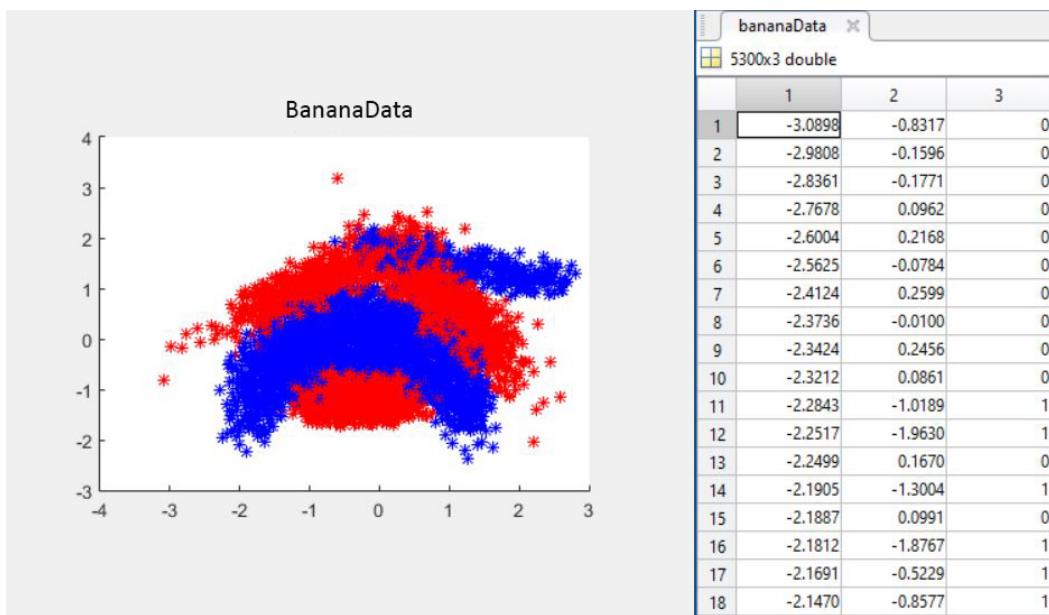


Figura 12: Sample del bananadata e corrispettivo plot

Trainingset e Validationset Dal dataset vengono estratti due sottoinsiemi, training e validation set. L'operazione di estrazione dei dati avviene nel seguente modo: All'utente è richiesto di inserire, prima della formazione di training set e validation set, un numero x ed un numero y di elementi che rappresentano la cardinalità dei due insiemi di dati.

- Viene generata una permutazione casuale di tutti gli elementi del dataset
- In base al numero x di elementi inseriti dall'utente nella GUI, sono selezionati i primi x elementi della permutazione in modo da formare il training set
- Partendo dall'elemento successivo nell'ordine della permutazione prescelta, vengono poi presi gli y elementi deputati a formare il validation set(gli y elementi sono sempre inseriti dall'utente nella GUI)

I sottoinsiemi generati, come risultato del suddetto algoritmo, saranno disgiunti, ciò per garantire la generalità della classificazione della rete. Sulla base dei dati del training set, verrà addestrata man mano la rete, tramite il validation set invece sarà possibile verificare che l'addestramento della rete avvenga correttamente e in modo più generale possibile per tutti i punti del dataset.

4.1.2 Gradient

Nella struttura gradient, sono contenuti tutti i dati necessari per la modifica della rete, tramite la metodologia della discesa del gradiente con momento. Di seguito è fornita una visione d'insieme sui principali elementi costitutivi di questa struttura.

ETA Il calcolo della variazione dei pesi e dei bias da effettuare sulla rete, si basa sul calcolo della derivata dell'errore relativa proprio a pesi e bias attuali. Tramite ETA tali derivate vengono normalizzate. In particolare nel caso del metodo batch, si è scelto di parametrizzare ETA sulla base del numero di elementi del trainingset cosicché la sommatoria delle derivate degli errori relativi ai pesi e ai bias risultasse proporzionata ai valori di peso e bias che volta per volta ci si trova a modificare nella rete.

MU L'idea del metodo della discesa del gradiente con momento è che se la precedente variazione di pesi e bias ha migliorato le capacità della rete, allora tale variazione debba essere tenuta in conto anche successivamente, allo scopo di velocizzare l'addestramento. La costante MU si occupa di regolare l'influenza che la variazione precedente deve avere sulla nuova modifica di pesi e bias.

RHO e SIGMA Tali costanti influiscono indirettamente nel calcolo della discesa del gradiente con momento tramite l'aggiornamento di ETA. RHO e SIGMA avranno rispettivamente valori leggermente superiori a 1 e compreso tra 0 e 1. Ad ogni aggiornamento della rete si valuta tramite il calcolo della variazione dell'errore, se le modifiche hanno avuto un effetto positivo. In caso positivo, ETA viene moltiplicata per RHO e dunque l'apprendimento viene leggermente velocizzato, in caso negativo vuol dire che la rete ha compiuto un passo d'apprendimento errato e dunque ETA viene moltiplicata per SIGMA e di conseguenza l'influenza delle successive modifiche è leggermente abbassata.

Variazione dei pesi e dei Bias In tali strutture vengono salvati i dati relativi alle variazioni effettuate dall’aggiornamento precedente, cosicché all’aggiornamento successivo, siano disponibili per contribuire alla nuova modifica della rete.

4.1.3 Results

Tale struttura é atta al contenimento di tutti i dati necessari ad un corretto studio del funzionamento della rete, mediante l’analisi dei risultati ottenuti. I dati sono memorizzati in doppia copia (una relativa al training set ed una al validation).

TruePositive Numero di elementi classificati come facenti parte della banana, che effettivamente si sono rivelati farne parte.

TrueNegative Numero di elementi classificati come **non** facenti parte della banana, che effettivamente si sono dimostrati **non** farne parte.

FalsePositive Numero di elementi classificati erroneamente come facenti parte della banana.

FalseNegative Numero di elementi classificati erroneamente come non facenti parte della banana.

Precision Tale valore rappresenta una stima percentuale di quante classificazioni positive fossero effettivamente corrette.

Recall Tale valore rappresenta una stima percentuale di quante classificazioni positive sono state effettivamente indovinate dalla rete.

Accuracy Tale valore rappresenta una stima percentuale del numero di classificazioni correttamente effettuate dalla rete.

Specificity Tale valore rappresenta sostanzialmente l’inverso della Precision, in quanto determina la percentuale di classificazioni che correttamente hanno ritenuto un punto come non facente parte della banana del bananadataset rispetto a quelle effettivamente corrette.

4.1.4 Net

É la struttura fondante di tutto il progetto software. Consente di contenere tutte le informazioni utili alla gestione della rete neurale. É strutturata in modo da garantire una buona efficienza delle operazioni di addestramento e testing della rete.

W W é una sottostruttura costituita da un vettore, in cui ciascun elemento rappresenta uno strato della rete. Ogni cella di tale vettore contiene la matrice dei pesi relativi agli archi tra lo strato di nodi precedente e quello corrente. Tali pesi sono quindi disposti in forma matriciale, in particolare l’indice ‘i’ della colonna individua l’iesimo nodo d’input per quello strato e l’indice ‘j’ di riga specifica il nodo dello strato corrente ai quali é relativo il peso dell’arco ‘Wji’.

B B é una struttura molto simile a W . Cosí come W anche B é composta da un vettore i cui elementi sono in corrispondenza uno ad uno con gli strati. In particolare per ogni strato (cella del vettore) la struttura contiene un sottovettore in cui ogni elemento corrisponde al bias associato al singolo nodo.

F e dF Cosí come le strutture precedenti anche F e dF sono organizzate per strati. Al loro interno contengono la funzione di output (e la derivata della funzione di output) relativa a ciascuno strato.

E e dE Sono stutture sostanzialmente identiche ad F e dF . Sono organizzate per strati ed al loro interno contengono la funzione di errore e la derivata della funzione di errore.

5 Testing

La fase di testing è stata sviluppata in modo programmatico, così da essere 'compliant' con quanto espressamente richiesto dalla traccia del progetto: (Gr.2) Si fissi la discesa del gradiente con momento come algoritmo di aggiornamento dei pesi, si studi l'apprendimento della rete neurale al variare dei nodi interni (Scegliere e mantenere invariati tutti gli altri 'parametri' come dataset, funzioni di output e funzione di errore).

A tal proposito si elencano i parametri che sono stati fissati, mentre in ogni sottosezione saranno esposte le singole variazioni della rete.

- Numero di nodi di input: 2
- Numero di nodi di output: 1
- Funzione di output dei nodi interni: sigmoide
- Funzione di output dei nodi di output: indentitá
- Funzione di errore della rete: errore quadratico medio. $E^n = 1/N \sum (a_k - t_k)^2$
- Numero di epoche per il training: 1000
- Numero di elementi del training set: 400
- Numero di elementi del validation set: 800
- Numero di elementi del test set: 800

Per quanto riguarda i parametri della rete relativi alla discesa del gradiente con momento facciamo una distinzione tra la modalitá 'Online' e 'Batch'.

Online:

- $\mu = 0.1$
- $\sigma = 0.5$
- $\rho = 1.1$
- Valore iniziale $\eta = 0.1$

Batch

- $\mu = 0.1$
- $\sigma = 0.5$
- $\rho = 1.01$
- Valore iniziale $\eta = (1/|TrainingSet|) * 0.1$

5.1 Test1

Struttura della rete utilizzata per il test: 2,20,30,1 .

Come si può vedere dalle curve presenti nei grafici sottostanti, nella prima fase, l'apprendimento online abbassa molto più rapidamente l'errore in quanto operando un aggiornamento sui pesi della rete neurale per ciascun punto (e non per ogni epoca come nel caso dell'aggiornamento batch), al termine delle prime epoche risulta più efficace. Man mano che la simulazione prosegue i risultati tendono invece a convergere. L'algoritmo, non termina per overfitting (esegue infatti tutte le 1000 epoche previste) sintomo questo che l'addestramento sulla rete se proseguito, avrebbe potuto apportare una maggiore capacità di classificare correttamente. Tale tesi viene avvalorata ulteriormente dal plotting dei punti del test set classificati dalla rete, i confini di decisione infatti tra la zona rossa e la zona blu appaiono molto ben definiti, stando sostanzialmente ad indicare che la classificazione è ancora grossolana e non molto particolareggiata.

5.1.1 Batch

Di seguito la schermata relativa all'apprendimento della rete, con l'andamento dell'errore rispetto alle epoche.

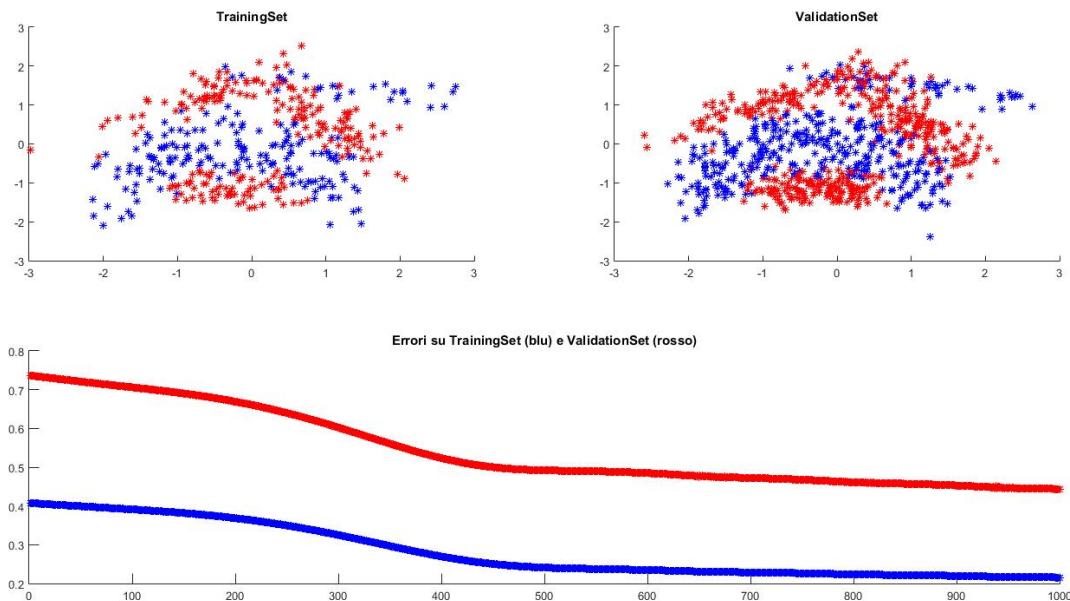


Figura 13: test1 - Apprendimento batch

Di seguito il confronto tra i dati del training set e quelli classificati dalla rete.

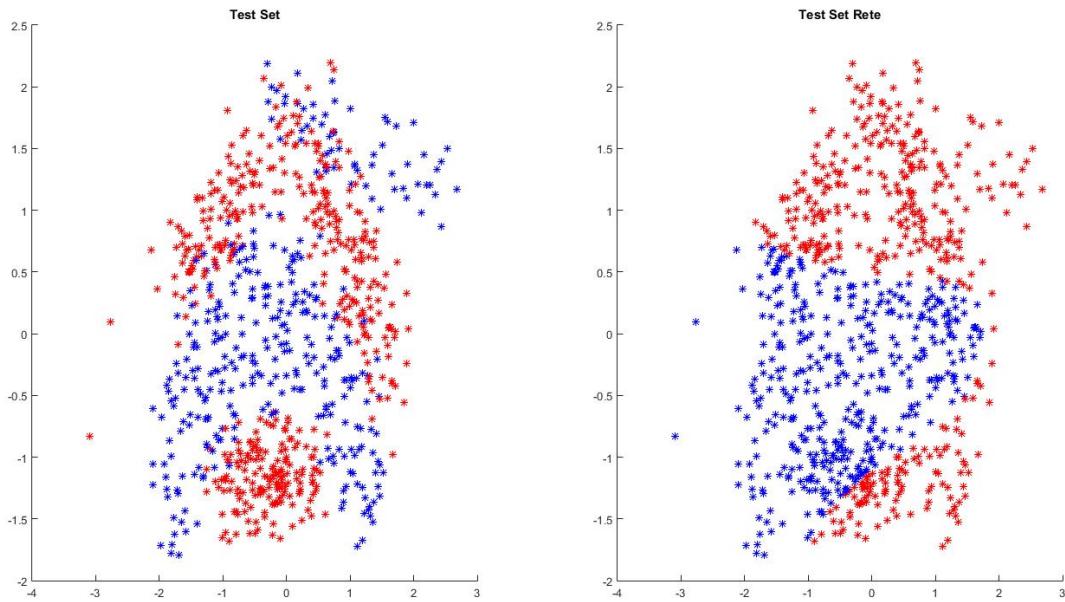


Figura 14: test1 - Confronto elementi classificati

Di seguito dei dati statistici per un'analisi più approfondita.

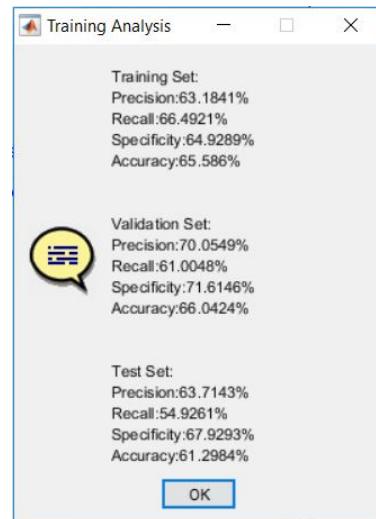


Figura 15: test1 - Dati statistici

5.1.2 Online

Di seguito la schermata relativa all'apprendimento della rete, con l'andamento dell'errore rispetto alle epoche.

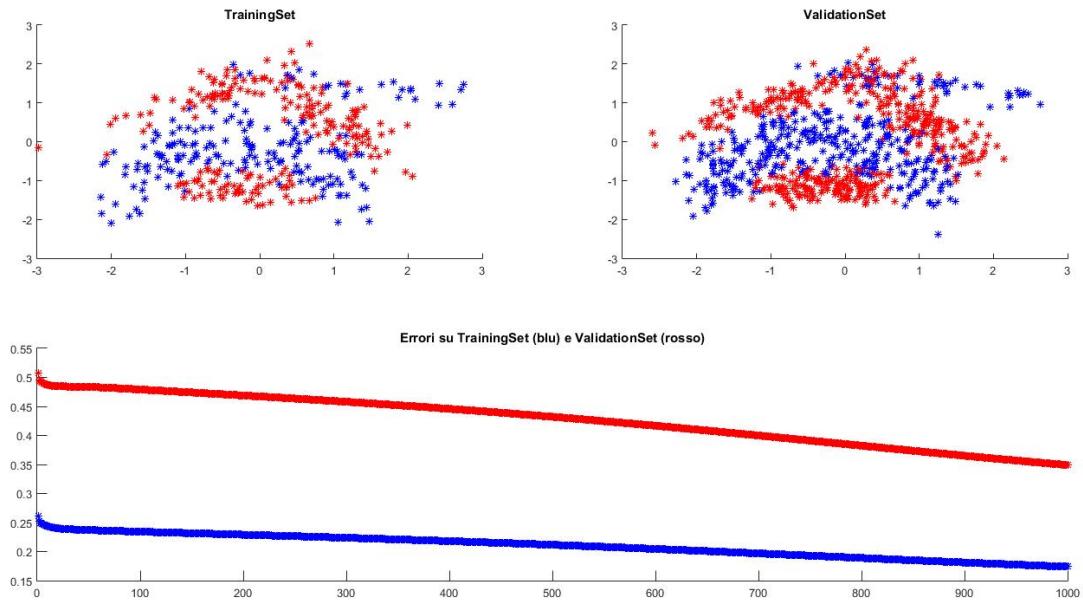


Figura 16: test1 - Apprendimento batch

Di seguito il confronto tra i dati del training set e quelli classificati dalla rete.

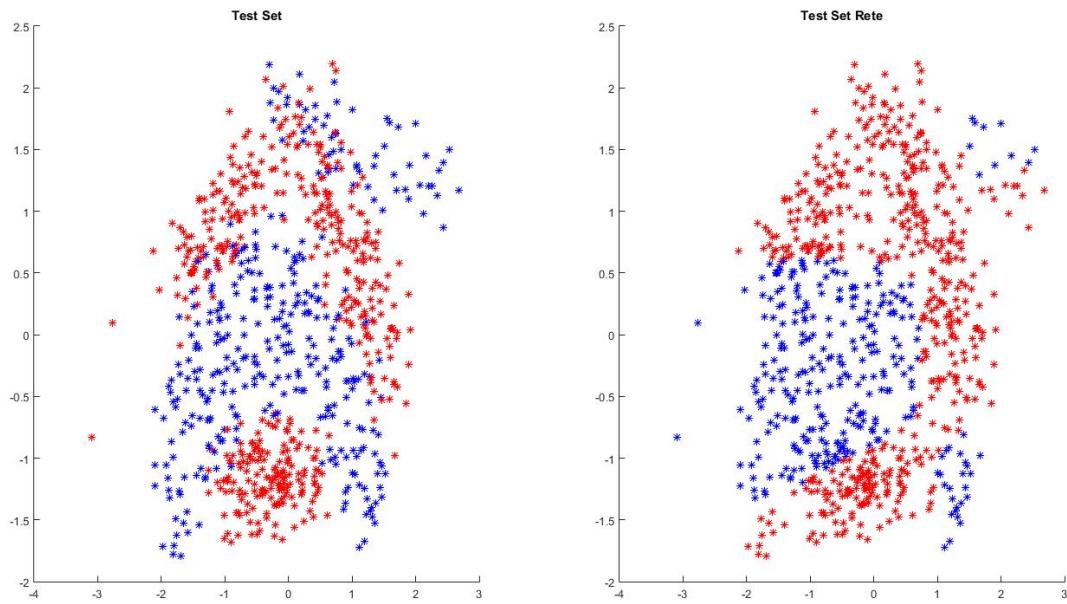


Figura 17: test1 - Confronto elementi classificati

Di seguito dei dati statistici per un'analisi più approfondita.

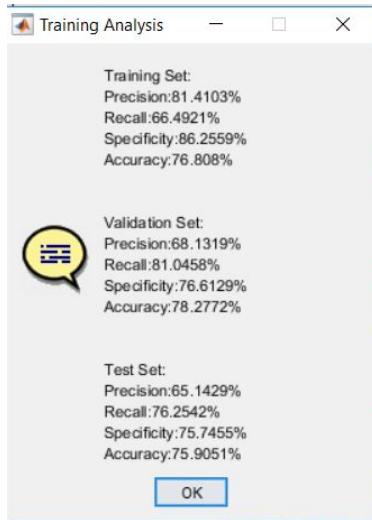


Figura 18: test1 - Dati statistici

5.2 Test2

Struttura della rete utilizzata per il test: 2,10,1 .

Rispetto al test precedente, l'errore calcolato dagli algoritmi batch e online risulta essere notevolmente superiore in partenza, ciò è certamente dovuto al minor numero di nodi e di strati della nuova rete neurale presa in esame. Il training set oggetto dell'addestramento però al contrario risulta avere dei punti separati in modo decisamente migliore. Ciò risulta in un addestramento più rapido e preciso della rete. Benché come nel caso precedente la rete non termini l'addestramento per overfitting e dunque sia ancora suscettibile di miglioramento. La classificazione in virtù degli insiemi di dati più 'favorevoli' appare decisamente migliore come testimoniato anche dai dati forniti dal 'Training Analysis' e dalla visualizzazione della classificazione sui test set.

5.2.1 Batch

Di seguito la schermata relativa all'apprendimento della rete, con l'andamento dell'errore rispetto alle epoche.

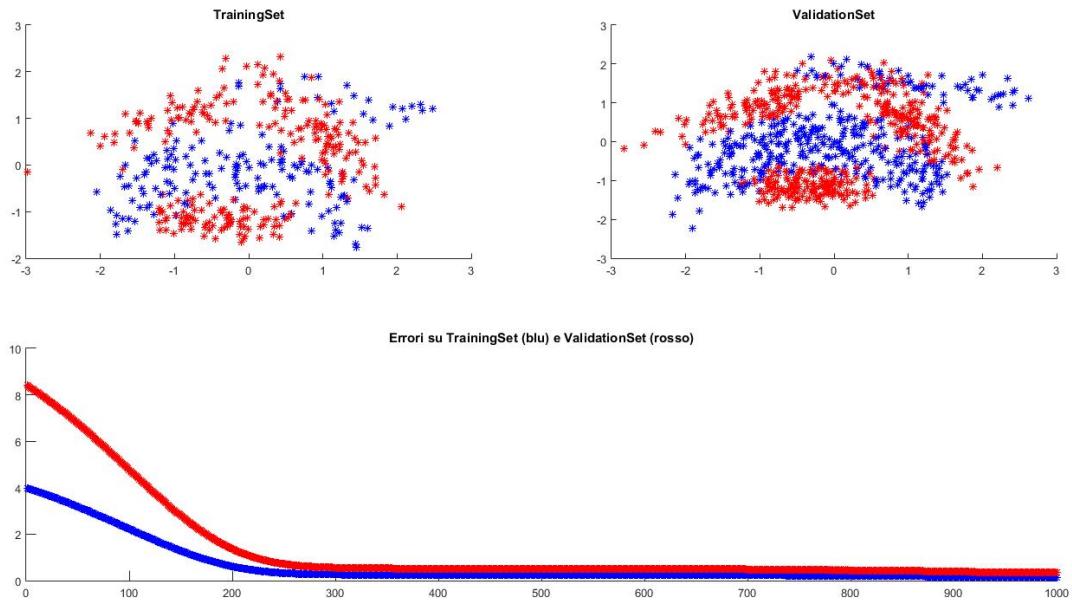


Figura 19: test1 - Apprendimento batch

Di seguito il confronto tra i dati del training set e quelli classificati dalla rete.

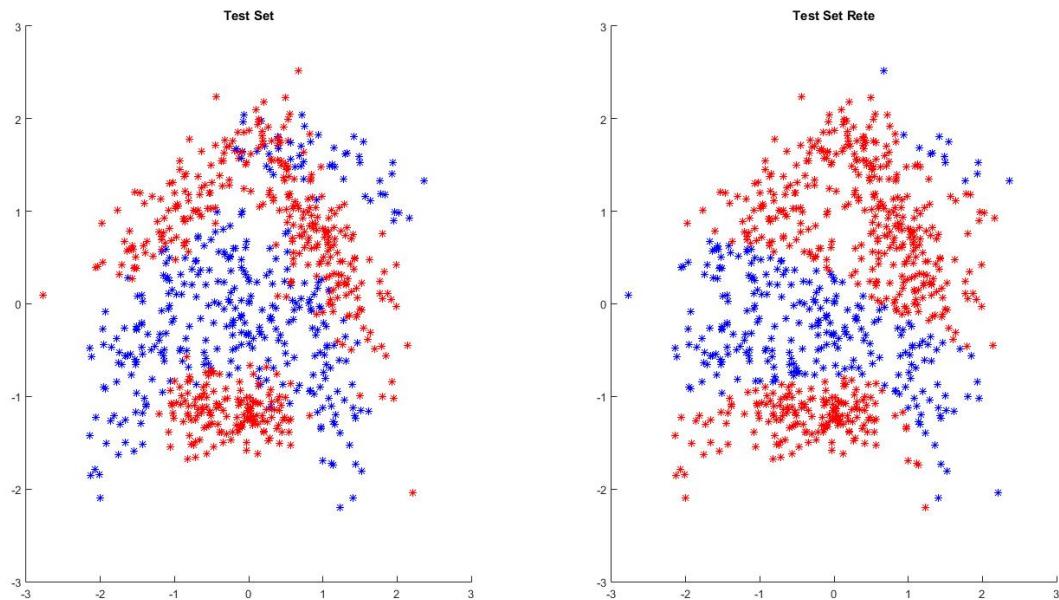


Figura 20: test1 - Confronto elementi classificati

Di seguito dei dati statistici per un'analisi più approfondita.

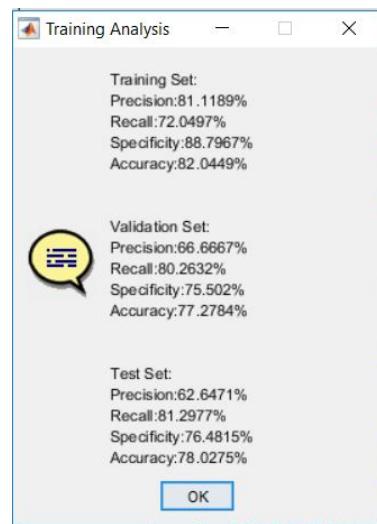


Figura 21: test1 - Dati statistici

5.2.2 Online

Di seguito la schermata relativa all'apprendimento della rete, con l'andamento dell'errore rispetto alle epoche.

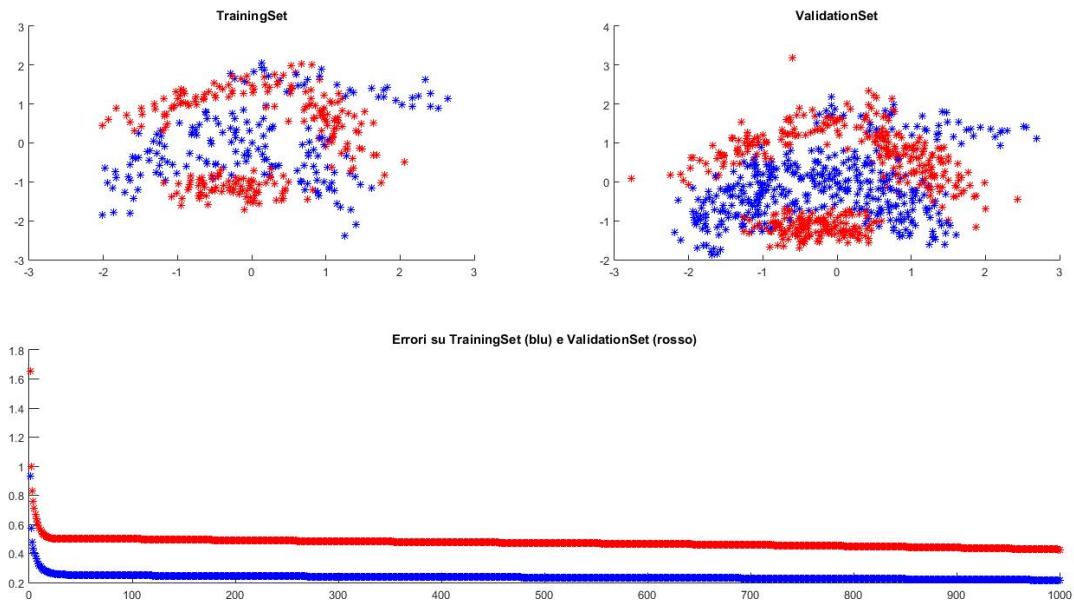


Figura 22: test1 - Apprendimento batch

Di seguito il confronto tra i dati del training set e quelli classificati dalla rete.

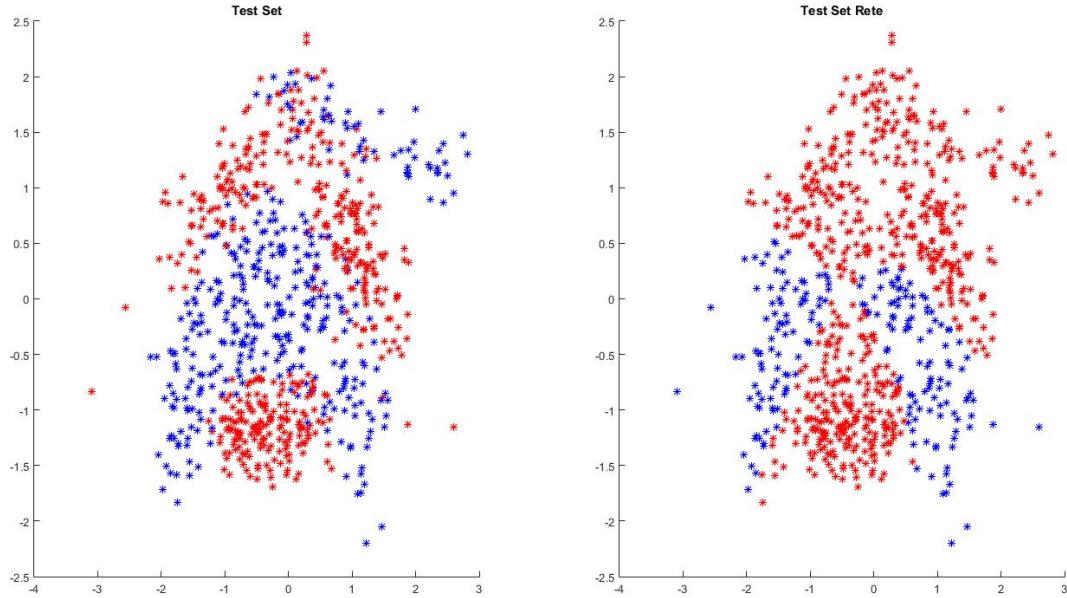


Figura 23: test1 - Confronto elementi classificati

Di seguito dei dati statistici per un'analisi più approfondita.

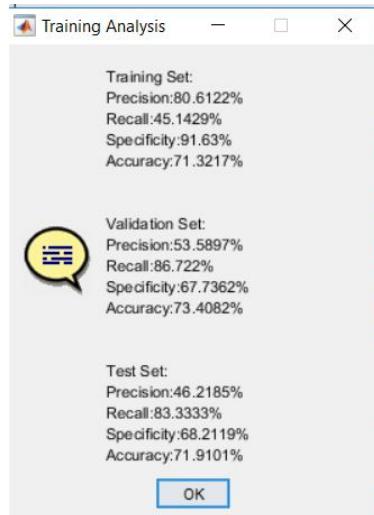


Figura 24: test1 - Dati statistici

5.3 Test3

Struttura della rete utilizzata per il test: 2,20,30,20,30,20,30,1 .

Avere una grande quantità di nodi e strati consente a tale rete di far calare l'errore sulla classificazione in modo drammatico nelle prime battute, soprattutto quando si utilizza l'algoritmo online. Successivamente però dover ricalibrare i pesi di così tanti archi tra i vari (pochi) neuroni per strato, risulta controproducente e la rete non

riesce a trovare un giusto equilibrio tra i vari parametri. Da ciò risulta una classificazione erronea e poco affidabile, nel caso dell'apprendimento online l'algoritmo termina addirittura per un presunto overfitting, sintomo questo, che lo squilibrio tra strati e neuroni risulta troppo penalizzante perch la rete possa addestrarsi in modo efficace.

5.3.1 Batch

Di seguito la schermata relativa all'apprendimento della rete, con l'andamento dell'errore rispetto alle epochhe.

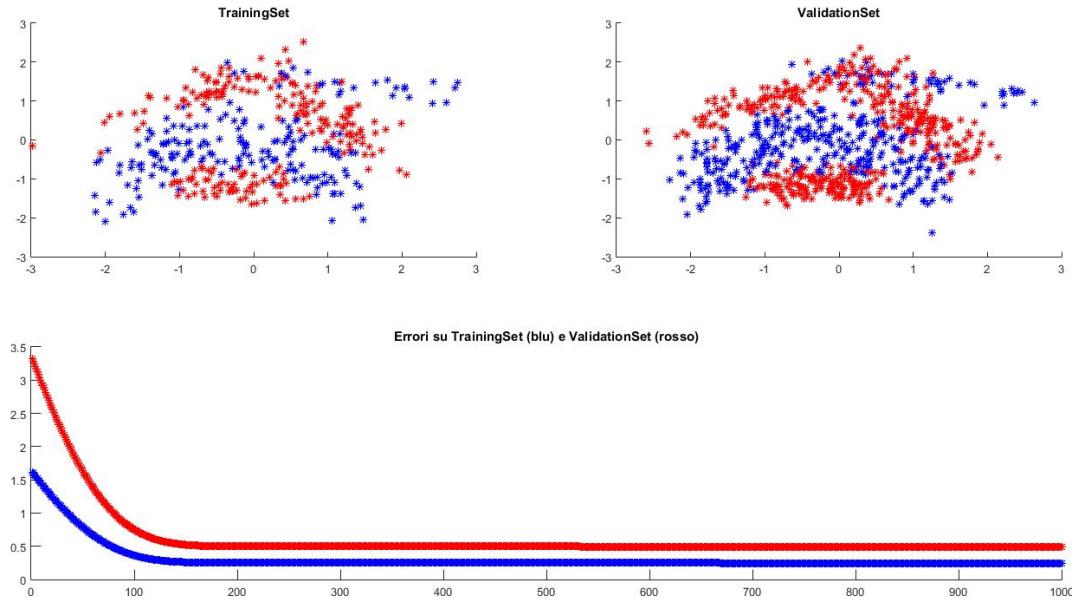


Figura 25: test1 - Apprendimento batch

Di seguito il confronto tra i dati del training set e quelli classificati dalla rete.

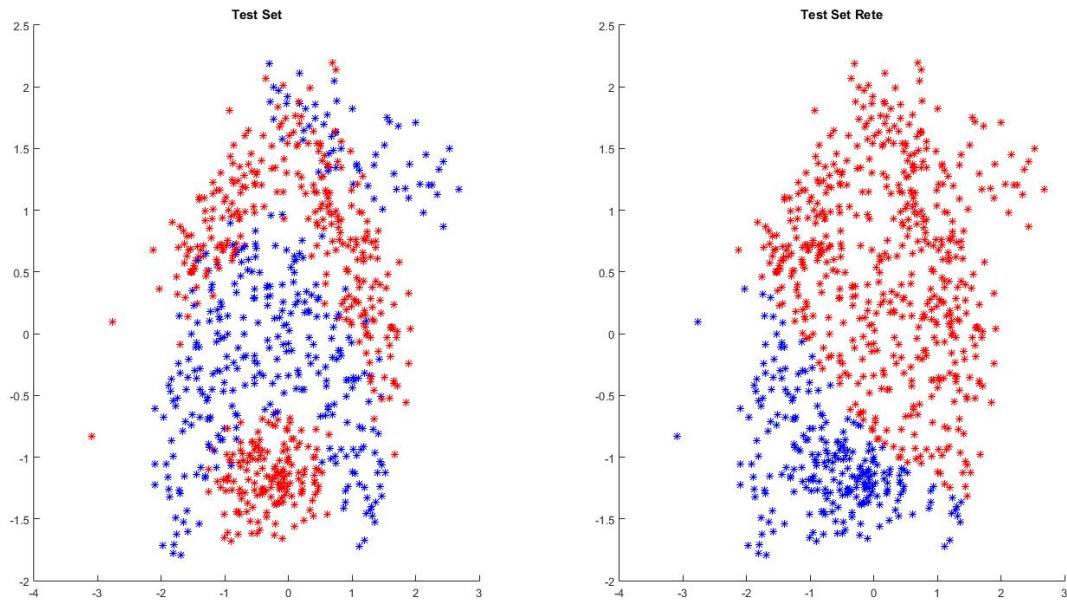


Figura 26: test1 - Confronto elementi classificati

Di seguito dei dati statistici per un'analisi più approfondita.

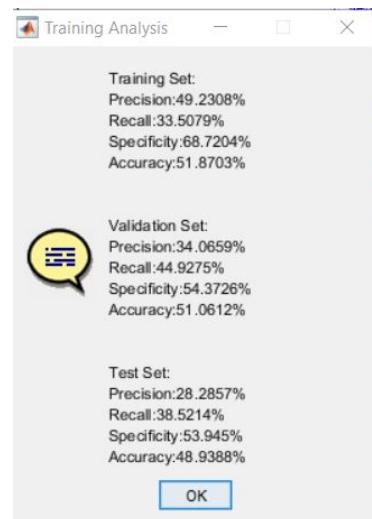


Figura 27: test1 - Dati statistici

5.3.2 Online

Di seguito la schermata relativa all'apprendimento della rete, con l'andamento dell'errore rispetto alle epoche.

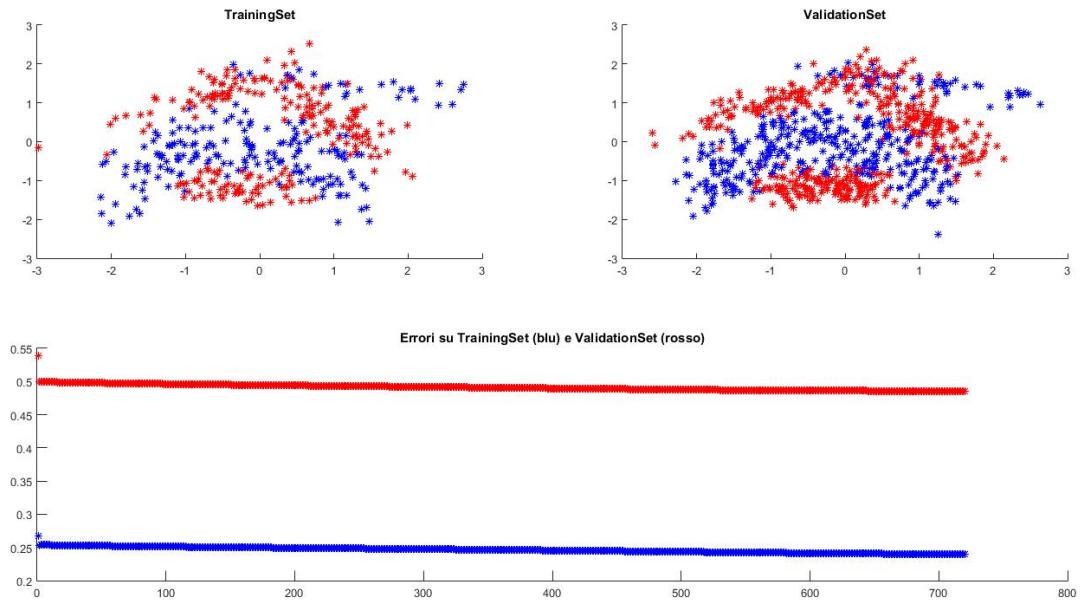


Figura 28: test1 - Apprendimento batch

Di seguito il confronto tra i dati del training set e quelli classificati dalla rete.

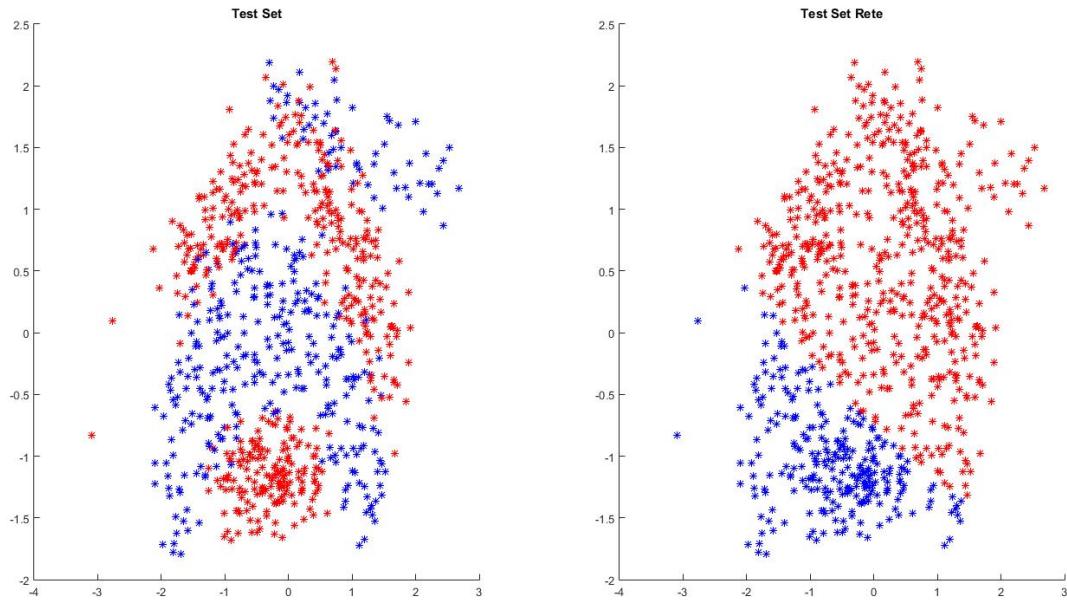


Figura 29: test1 - Confronto elementi classificati

Di seguito dei dati statistici per un'analisi più approfondita.

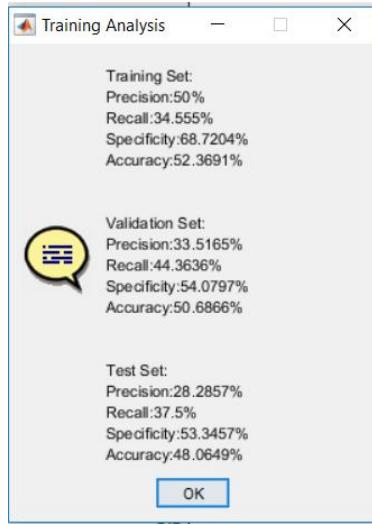


Figura 30: test1 - Dati statistici

5.4 Test4

Struttura della rete utilizzata per il test: 2,200,300,1 .

Come nel test precedente, avere a disposizione una gran quantitá di nodi all'interno della rete neurale consente alla stessa, di essere molto efficace in prima battuta, abbassando molto rapidamente l'errore sulla classificazione. Al contrario del test precedente però, il minor numero di strati (e di archi) consente alla rete di trovare un valido equilibrio tra i vari parametri, ciò risulta in una classificazione ottima, in cui i parametri di Accuracy, Specificity, Precision e Recall rilevati sul test set si attestano intorno al 90

5.4.1 Batch

Di seguito la schermata relativa all'apprendimento della rete, con l'andamento dell'errore rispetto alle epoche.

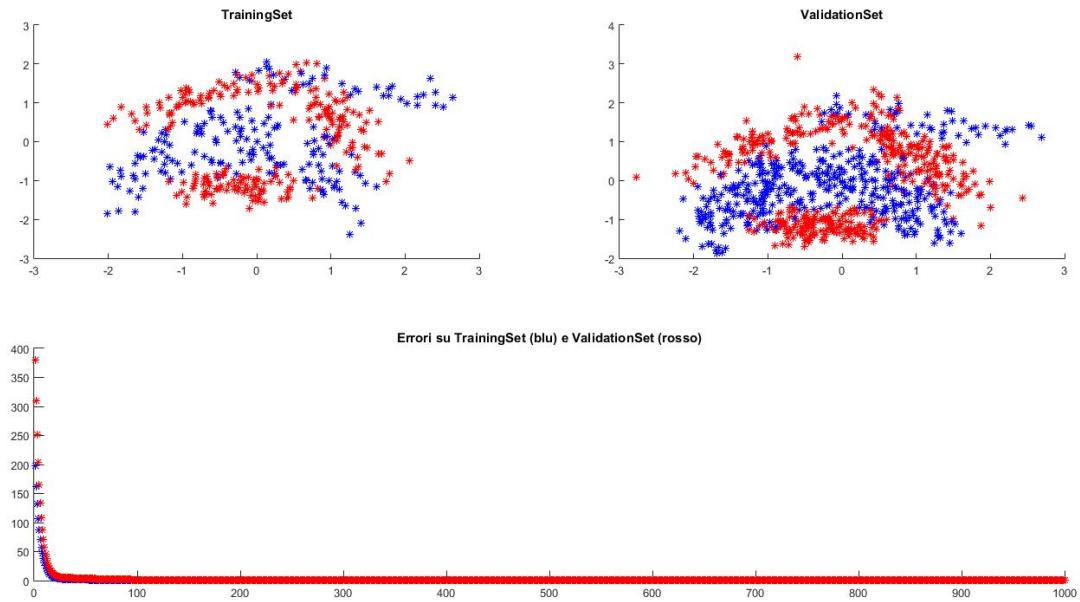


Figura 31: test1 - Apprendimento batch

Di seguito il confronto tra i dati del training set e quelli classificati dalla rete.

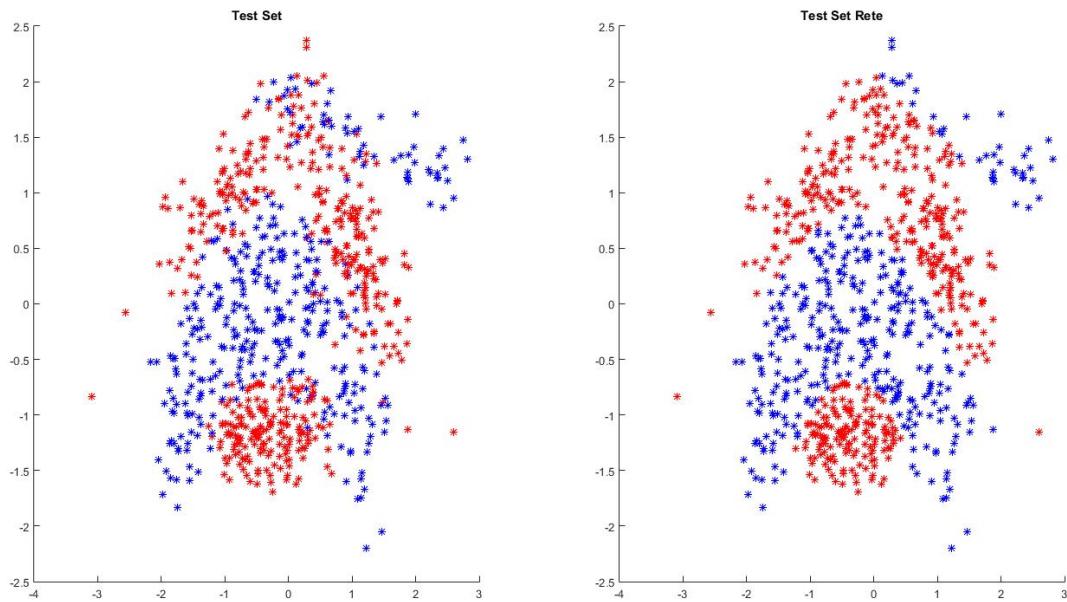


Figura 32: test1 - Confronto elementi classificati

Di seguito dei dati statistici per un'analisi più approfondita.

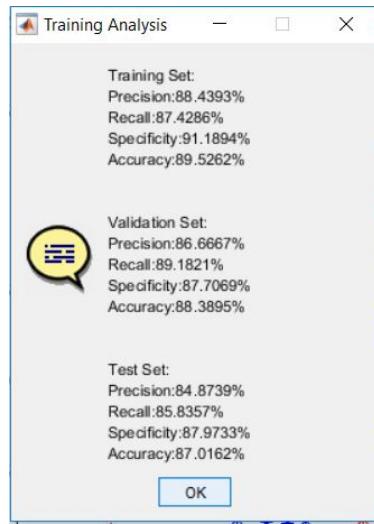


Figura 33: test1 - Dati statistici

5.4.2 Online

Di seguito la schermata relativa all'apprendimento della rete, con l'andamento dell'errore rispetto alle epoche.

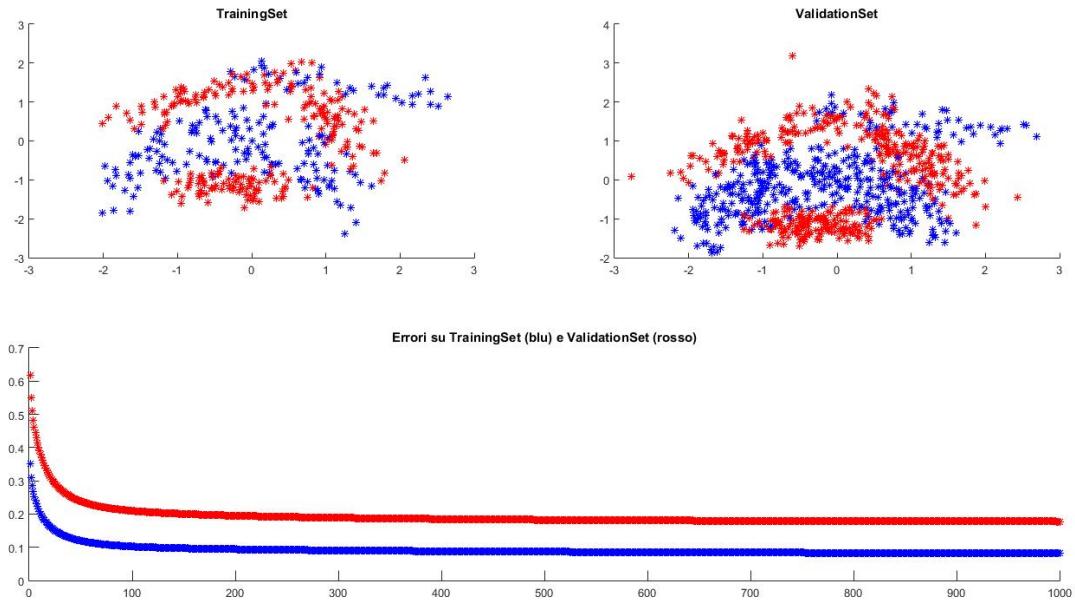


Figura 34: test1 - Apprendimento batch

Di seguito il confronto tra i dati del training set e quelli classificati dalla rete.

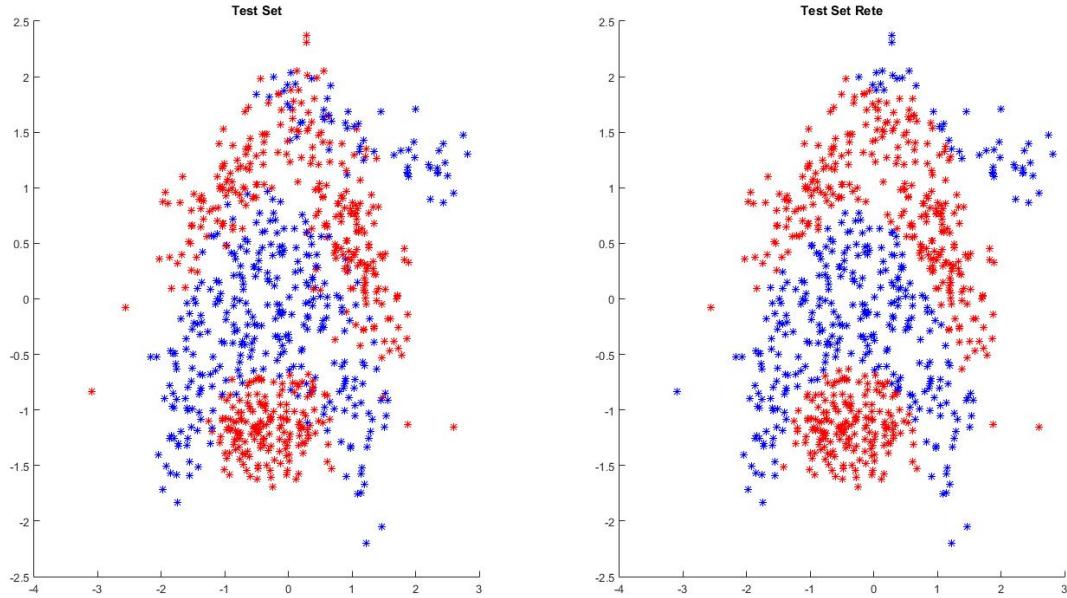


Figura 35: test1 - Confronto elementi classificati

Di seguito dei dati statistici per un'analisi più approfondita.

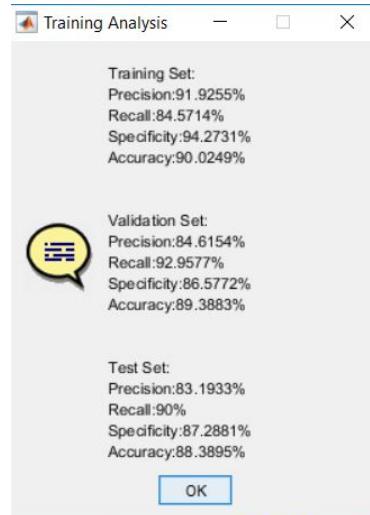


Figura 36: test1 - Dati statistici

5.5 Test5

Struttura della rete utilizzata per il test: 2,200,300,200,300,200,300,1 .

Aumentando il numero di strati rispetto al test precedente si può notare come i risultati siano sostanzialmente invariati. Una grande quantità di neuroni per strato, consente alla rete di conservare un'elevata precisione di classificazione, bisogna però tenere in conto che aumentare le dimensioni della rete in questo modo comporta un

evidente calo nella rapidità d'apprendimento causato dall'enorme mole di calcoli da fare durante la simulazione della rete (forward propagation) e l'aggiustamento dei pesi.

5.5.1 Batch

Di seguito la schermata relativa all'apprendimento della rete, con l'andamento dell'errore rispetto alle epoche.

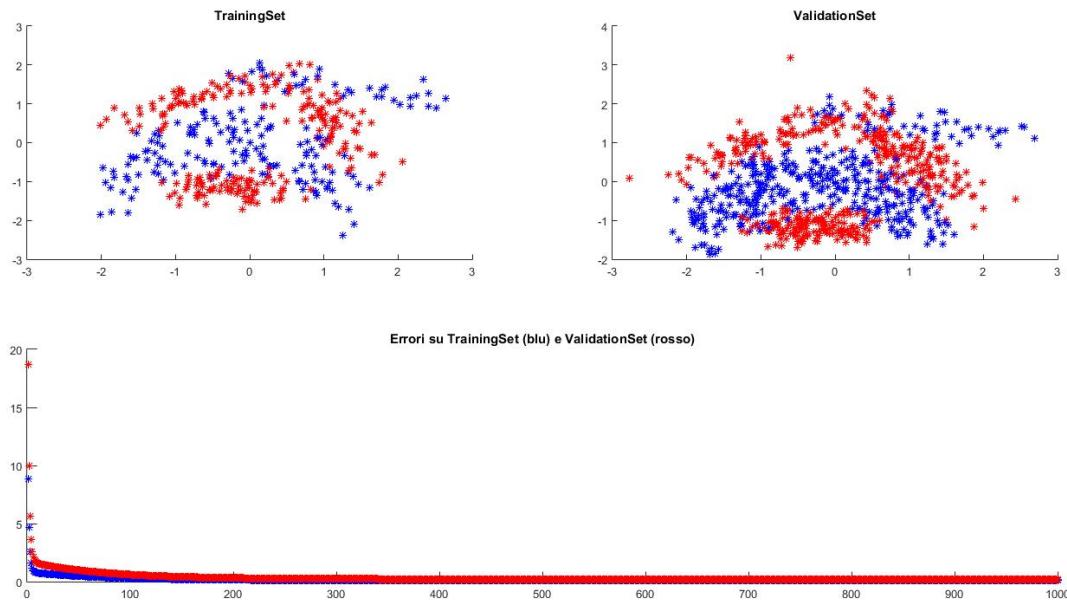


Figura 37: test1 - Apprendimento batch

Di seguito il confronto tra i dati del training set e quelli classificati dalla rete.

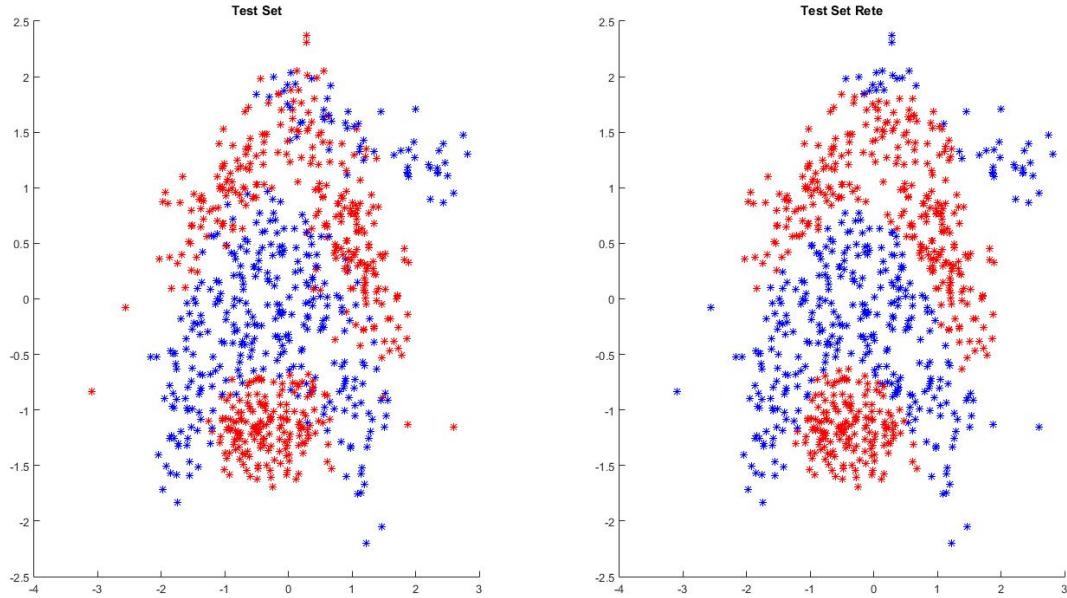


Figura 38: test1 - Confronto elementi classificati

Di seguito dei dati statistici per un'analisi più approfondita.

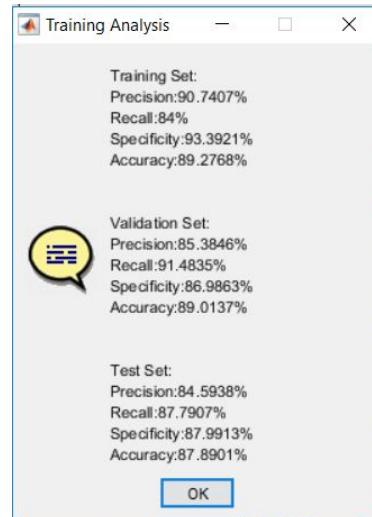


Figura 39: test1 - Dati statistici

5.5.2 Online

Di seguito la schermata relativa all'apprendimento della rete, con l'andamento dell'errore rispetto alle epoche.

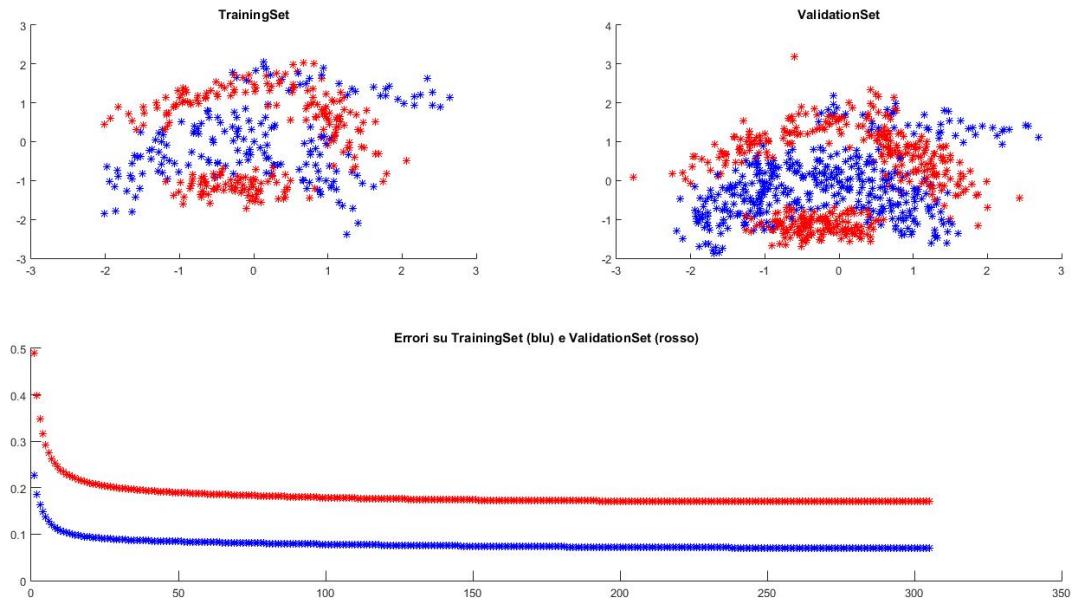


Figura 40: test1 - Apprendimento batch

Di seguito il confronto tra i dati del training set e quelli classificati dalla rete.

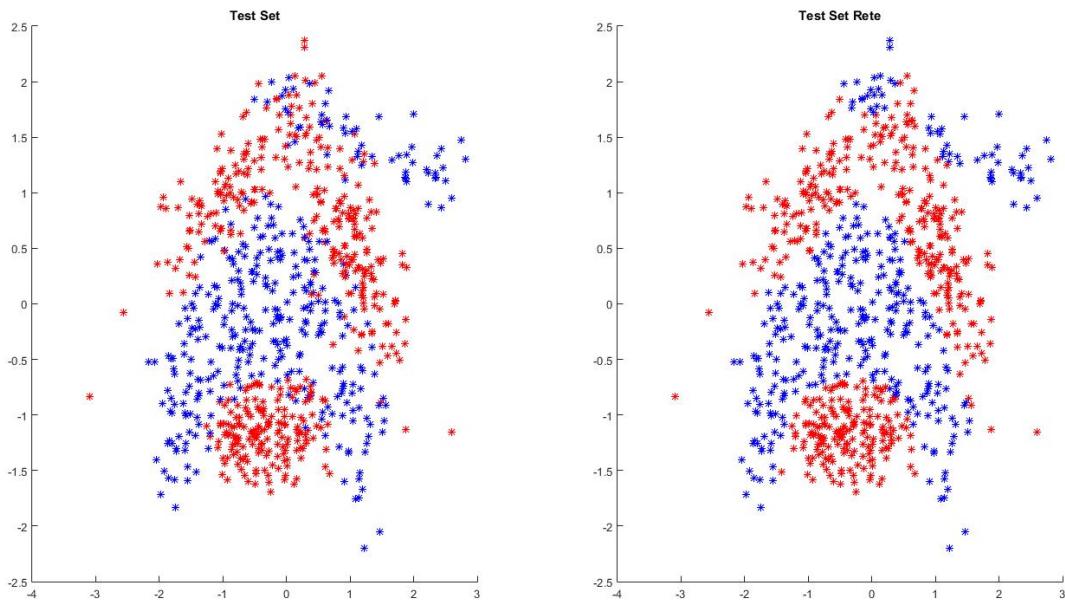


Figura 41: test1 - Confronto elementi classificati

Di seguito dei dati statistici per un'analisi più approfondita.

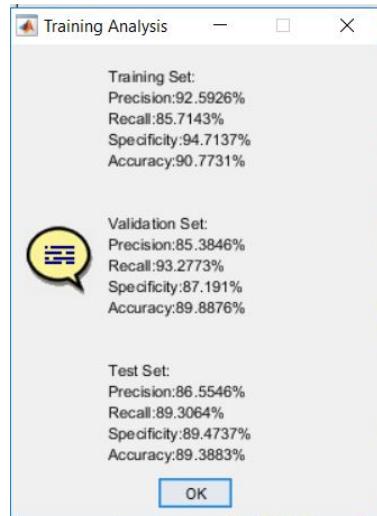


Figura 42: test1 - Dati statistici

6 Source code

Di seguito sono presentate le componenti principali del software sviluppato, per quanto riguarda le altre funzioni fare riferimento al pacchetto zip allegato.

6.1 NewNet

```
1 function net = newNet( numFeatures , infoStrati , funOutput , funErr )
2 %NEWNET - funzione per la creazione di una nuova rete
3 % La funzione prende in input:
4 %     - numFeatures = numero di features del dataset o nodi di input della
5 %         rete
6 %     - infoStrati = vettore contenente il numero di strati (i.e.
7 %         size(infoStrati)) e il numero di nodi per ogni strato
8 %     - funOutput = vettore contenente le funzioni di output dei vari
9 %         strati della rete
10 %     - funErr = variabile contenente la funzione di errore che la rete
11 %         deve minimizzare
12 % La funzione restituisce in output una rete inizializzata con valori
13 % random per i pesi.
14 gradient.RHO=1.1;
15 gradient.MU =0.1;
16 gradient.SIGMA = 0.5;
17 gradient.ETA=0.1;
18
19 %MLB - Creiamo delle variabili simboliche per le derivate delle
20 %funzioni di output e di errore
21 syms x;
22 syms y;
23 syms z;
24
25 %MLB - Se il numero di funzioni di output diverso dal numero di
26 %strati, allora ci sono esattamente due funzioni. La prima viene
27 %replicata per tutti i nodi degli strati interni, la seconda assegnata a
28 %tutti i
29 %nodi dell'ultimo strato della rete
30 if (size(funOutput ,2)~=size(infoStrati ,2))
31     funzinterna=funOutput{1};
32     funzoutput=funOutput{2};
33
34     for i=1:size(infoStrati ,2)-1
```

```

33         funOutput{i}=funzinterna;
34
35     end
36
37 end
38
39 %MLB - Per ogni strato assegnamo valori random a pesi e bias di ogni
40 %strato, assegnamo la funzione corrispondente e la derivata della
41 %funzione corrispondente ad ogni strato
42 for i=1:size(infoStrati,2)
43
44     if i == 1
45
46         gradient.errorVariationW{i}=zeros(infoStrati(i),numFeatures);
47
48         net.W{i} = 2*(rand(infoStrati(i),numFeatures))-1;
49
50     else
51
52         gradient.errorVariationW{i}=zeros(infoStrati(i),infoStrati(i-1));
53
54         net.W{i} = 2*(rand(infoStrati(i),infoStrati(i-1)))-1;
55
56     end
57
58     gradient.errorVariationB{i}=zeros(1,infoStrati(i));
59
60
61     net.B{i} = (2*rand(1, infoStrati(i)))-1;
62
63     net.F{i} = funOutput{i};
64
65     f= net.F{i}(x);
66
67     net.dF{i} = matlabFunction(diff(f,x));
68
69 end
70
71 %MLB - assegna alla rete la funzione d'errore e la sua derivata
72
73 net.E = funErr;
74
75 e = net.E(y,z);
76
77 net.dE = matlabFunction(diff(e));
78
79 assignin('base','gradient',gradient);
80
81 return;
82
83 end

```

6.2 ForwardPropagation

```

1 function [ y,a ] = forwardPropagation( net,x )
2 %FORWARDPROPAGATION - funzione per la forwardPropagation dell'elemento x
3 %(oppure dell'insieme x) nella rete net.
4 %
5 % La funzione prende in input una rete net e un elemento\insieme x e simula
6 % la
7 %
8 % rete net su x.
9 %
10 % La funzione restituisce in output due vettori a e y che rappresentano

```

```

7 % rispettivamente i valori di input ed output per ogni nodo di ogni livello
8 [N d]= size(x);
9
10 a = cell(size(net.W,1),size(net.W,2));
11 y = cell(size(net.W,1),size(net.W,2));
12
13 %MLB - Prendiamo x e lo diamo in pasto alla rete, salvando i
14 %valori di input ed output per ogni nodo di ogni livello
15 a{1} = (net.W{1} * x')' + repmat(net.B{1},N,1);
16 y{1} = net.F{1,1}(a{1});
17
18 for k=2: size(net.W,2)
19 a{k} = (net.W{k}*y{k-1}')' + repmat(net.B{k},N,1);
20 y{k}= net.F{k}(a{k});
21 end
22
23 return;
24 end

```

6.3 BackPropagation

```

1 function [ dw, db ] = backPropagation( x_t,net )
2 %BACKPROPAGATION - Funzione di backPropagation per l'addestramento della
3 %rete
4 % La funzione applica il processo di addestramento tramite
5 % backPropagation alla rete in ingresso.
6 % Prende in input un elemento del training set (x_t) e una rete (net).
7 % La funzione restituisce in output i vettori dw e db che rappresentano
8 % rispettivamente le derivate dei pesi e dei bias ottenuti con il
9 % processo di backpropagation che punta alla minimizzazione della
10 % funzione di errore.
11
12 %MLB - Prende solo le features dall'elemento escludendo il tag di
13 % classificazione e li salva in x
14 x = x_t(:,1:end-size(net.W{1},end),1);
15 targ = x_t(:,size(x,2)+1:end);
16
17 %MLB - eseguiamo la forwardPropagation sull'elemento e salva input ed
18 %output dei nodi (i.e. a,y)
19 [y,a] = forwardPropagation(net,x);

```

```

20
21 %MLB - dim = numero di strati
22 dim = size(net.W,2);
23
24 %MLB - inizializzazione per velocizzare l'esecuzione del programma
25
26 dw = cell(1,dim);
27 db = cell(1,dim);
28
29
30 %MLB - Calcoliamo il delta_k, ovvero la derivata della funzione di
31 %errore rispetto agli input dei nodi di output
32 derr= net.dE(y{dim},targ);
33
34 if nargin(net.dF{1,end}) == 0
35     delta_k = derr .* net.dF{1,end}();
36 else
37     delta_k = derr .* net.dF{1,end}(a{dim});
38 end
39 delta=delta_k;
40 %MLB - Calcoliamo le derivate dei bias e dei pesi per l'ultimo strato
41 %di nodi. Questi saranno poi usati nella discesa del gradiente in modo
42 %da aggiornare correttamente i pesi e minimizzare l'errore di
43 %classificazione
44 db{dim}=sum(delta_k);
45 dw{dim}= delta_k' * y{dim-1};
46
47 %MLB - Ripetiamo il procedimento appena descritto per tutti gli strati
48 %interni
49 for k=dim-1:-1:1
50
51     delta_j =net.dF{k}(a{k}) .* (delta * net.W{1,k+1} );
52     db{k} = sum(delta_j);
53
54     if k > 1
55         dw{k}=(delta_j' * y{k-1});
56     else
57         dw{k}=(delta_j' * x);
58     end
59     delta=delta_j;

```

```
58     end
```

```
59
```

```
60     return;
```

```
61 end
```

6.4 GradientDescent

```
1 function [ net, gradient ] = gradientDescent( dw,db,net,gradient )
2 %GRADIENTDESCENT
3 % MLB - Funzione per la discesa del gradiente con momento.
4 % Input :
5 % - dw = derivate dell'errore rispetto ai pesi;
6 % - db = derivate dell'errore rispetto ai bias;
7 % - net = la rete;
8 % - gradient = la struttura gradient;
9 % Output :
10 % - net = la rete aggiornata;
11 % - gradient = la struttura gradient;
12
13 errorVariationW = cell(1,size(net.W,2));
14 errorVariationB = cell(1,size(net.W,2));
15
16 for k=1:size(net.W,2)
17     errorVariationW{k} =-(gradient.ETA .* dw{k}) + (gradient.MU .* gradient.
18         errorVariationW{k});
19
20     net.W{k} = net.W{k}+errorVariationW{k};
21     gradient.errorVariationW{k} = errorVariationW{k};
22
23     errorVariationB{k} =-(gradient.ETA .* db{k}) + (gradient.MU .* gradient.
24         errorVariationB{k});
25     net.B{k} = net.B{k}+errorVariationB{k};
26     gradient.errorVariationB{k} = errorVariationB{k};
27
28 end
29
30 return;
31 end
```

6.5 Batch

```

1 function [net,gradient] = batch( numeroEpoche,subsetTR, subsetVS,subsetTS,net,
2 gradient )
3 %BATCH
4 % MLB - La funzione effettua un apprendimento batch della rete e la testa
5 % su Validation Set e Test Set.
6 % Input :
7 % - numeroEpoche : numero delle epoch;
8 % - subsetTR : il training set;
9 % - subsetVS : il validation set;
10 % - subsetTS : il test set;
11 % - net : la rete;
12 % - gradient : la struttura gradient;
13 %
14 % Output :
15 % - net : la rete aggiornata;
16 % - gradient : la struttura gradient aggiornata;
17
18
19 %MLB - eseguiamo l'addestramento della rete su ogni nodo del trainingset
20
21 %MLB - inizializzazione delle variabili necessarie
22 h=waitbar(0, sprintf('%s',[ num2str(0) '/ num2str(numeroEpoche) ]), 'Name',
23 , 'Apprendimento in corso... ');
24 erroreTotaleTR = zeros(1,numeroEpoche);
25 erroreTotaleVS = zeros(1,numeroEpoche);
26
27
28 gradient.ETA = (1/size(subsetTR,1))*gradient.ETA;
29 gradient.SIGMA = 0.5;
30 gradient.RHO = 1.01;
31 consecutiveOverfitting=0;
32 %MLB - inizio dell'apprendimento
33 for k=1:size(gradient.errorVariationW,2)
34     gradient.errorVariationW{k} = zeros(size(gradient.errorVariationW{k},1)
35         ,size(gradient.errorVariationW{k},2));
36     gradient.errorVariationB{k} = zeros(size(gradient.errorVariationB{k},1)
37         ,size(gradient.errorVariationB{k},2));
38 end

```

```

37 numFeatures = size(subsetTR(:,1:end-size(net.W{1},end),1)),2);
38 numTargets = numFeatures+1;
39 for contaEpoca=1:numeroEpoch
40
41
42 waitbar(contaEpoca/numeroEpoch,h, sprintf('%s',[ num2str(contaEpoca) ,
43 /' num2str(numeroEpoch) ' Epoch ]) );
44
45 [dw, db]=backPropagation(subsetTR,net);
46 [net, gradient] = gradientDescent(dw,db,net,gradient);
47
48 %MLB - Calcolo dell'errore sul tr
49
50 erroreSommatoTR=0;
51 y = forwardPropagation(net,subsetTR(:,1:numFeatures));
52 erroreSommatoTR = sum(net.E(y{1,size(net.W,2)}),subsetTR(:,numTargets:
53 end));
54
55 erroreTotaleTR(contaEpoca) = sum(erroreSommatoTR);
56 assignin('base','erroreTotaleTR', erroreTotaleTR);
57
58 if contaEpoca>1 && (erroreTotaleTR(contaEpoca)-erroreTotaleTR(
59 contaEpoca-1) > 0)
60     gradient.ETA= gradient.ETA * gradient.SIGMA;
61 else
62     gradient.ETA= gradient.ETA * gradient.RHO;
63 end
64
65 %MLB - Calcolo dell'errore sul vs
66
67 erroreSommatoVS=0;
68 y = forwardPropagation(net,subsetVS(:,1:numFeatures));
69 erroreSommatoVS = sum(net.E(y{1,size(net.W,2)}),subsetVS(:,numTargets:
70 end));
71 erroreTotaleVS(contaEpoca) = sum(erroreSommatoVS);
72
73 if contaEpoca > 20 && erroreTotaleTR(contaEpoca)/size(subsetTR,1) <
74     erroreTotaleVS(contaEpoca)/size(subsetVS,1) && erroreTotaleVS(

```

```

    contaEpoca) - erroreTotaleVS(contaEpoca-1) > 0 && erroreTotaleTR(
    contaEpoca) - erroreTotaleTR(contaEpoca-1) < 0
        consecutiveOverfitting = consecutiveOverfitting+1;
    else
        consecutiveOverfitting = 0;
    end
    if consecutiveOverfitting > 15
        uiwait(helpdlg('Apprendimento terminato causa overfitting.', 'modal'
            ));
        break;
    end
end
delete(h);
%MLB - Stampa dei risultati
plots(subsetTR, subsetVS,subsetTS, erroreTotaleTR, erroreTotaleVS,
contaEpoca, 'batch', net);

end

```

6.6 Online

```

1 function [net,gradient] = online( numeroEpoche,subsetTR, subsetVS,subsetTS,net,
gradient )
2 %ONLINE
3 % MLB - La funzione effettua un apprendimento online della rete e la testa
4 % su Validation Set e Test Set.
5 % Input :
6 % - numeroEpoche : numero delle epoche;
7 % - subsetTR : il training set;
8 % - subsetVS : il validation set;
9 % - subsetTS : il test set;
10 % - net : la rete;
11 % - gradient : la struttura gradient;
12 %
13 % Output :
14 % - net : la rete aggiornata;
15 % - gradient : la struttura gradient aggiornata;

16
17
18
19

```

```

20 %MLB - eseguiamo l'addestramento della rete su ogni nodo del trainingset
21
22 %MLB - inizializzazione delle variabili necessarie
23 h=waitbar(0, sprintf('%s',[ num2str(0) '/' num2str(numeroEpoche) ]), 'Name',
24 , 'Apprendimento in corso... );
25 erroreTotaleTR = zeros(1,numeroEpoche);
26 erroreTotaleVS = zeros(1,numeroEpoche);
27 consecutiveOverfitting = 0;
28 numFeatures = size(subsetTR(:,1:end-size(net.W{1},end),1)),2);
29 numTargets = numFeatures+1;
30 %MLB - inizio dell'apprendimento
31 for contaEpoca=1:numeroEpoche
32
33
34
35 gradient.ETA = 0.1;
36 %MLB - apprendimento sulla singola epoca
37 for k=1:size(subsetTR,1)
38 [dw, db]=backPropagation(subsetTR(k,1:end),net);
39 [net, gradient] = gradientDescent(dw,db,net,gradient);
40 end
41
42 %MLB - Calcolo dell'errore sul tr
43 erroreSommatoTR = 0;
44
45 y = forwardPropagation(net,subsetTR(:,1:numFeatures));
46 erroreSommatoTR = sum(net.E(y{size(net.W,2)}),subsetTR(:,numTargets:
47 end));
48
49 erroreTotaleTR(contaEpoca) = sum(erroreSommatoTR);
50
51 if contaEpoca>1 && (erroreTotaleTR(contaEpoca)-erroreTotaleTR(
52 contaEpoca-1) > 0)
53 gradient.ETA= gradient.ETA * gradient.SIGMA;
54 else
55 gradient.ETA= gradient.ETA * gradient.RHO;
56 end

```

```

56         %MLB - Calcolo dell' errore sul vs
57         erroreSommatoVS = 0;
58         y = forwardPropagation(net,subsetVS(:,1:numFeatures));
59         erroreSommatoVS = sum( net.E(y{size(net.W,2)}},subsetVS(:,numTargets:end
60             )));
61
62         erroreTotaleVS(contaEpoca) = sum(erroreSommatoVS);
63
64         if contaEpoca > 20 && erroreTotaleTR(contaEpoca)/size(subsetTR,1) <
65             erroreTotaleVS(contaEpoca)/size(subsetVS,1) && erroreTotaleVS(
66             contaEpoca) - erroreTotaleVS(contaEpoca-1) > 0 && erroreTotaleTR(
67             contaEpoca) - erroreTotaleTR(contaEpoca-1) < 0
68             consecutiveOverfitting = consecutiveOverfitting+1;
69         else
70             consecutiveOverfitting = 0;
71         end
72         if consecutiveOverfitting > 15
73             uiwait(helpdlg('Apprendimento terminato causa overfitting.', 'modal'
74                 ));
75             break;
76         end
77     end
78
79     delete(h);
80
81     %MLB - Stampa dei risultati
82     plots(subsetTR, subsetVS,subsetTS, erroreTotaleTR, erroreTotaleVS,
83         contaEpoca,'online',net);
84
85
86 end

```

6.7 VerifyNet

```

1 function [ results ] = verifyNet( net, subsetTR, subsetVS, subsetTS )
2 %VERIFYNET
3 %    MLB - La funzione a partire dallo stato della rete dopo l'apprendimento
4 %    e dai subset del validation set e del training set restituisce una
5 %    serie di statistiche utili a valutare l'apprendimento effettuato.
6 %    Riceve in input :
7 %        - la rete;
8 %        - il subset del Training Set
9 %        - il subset del Validation Set
10 %    Restituisce in output :

```

```

11 % - la struttura 'result' contenente tutte le statistiche sulla rete
12 N = size(net.W,2);
13 %MLB - Inizializza la struttura result
14 results.tpTR=0;
15 results.tnTR=0;
16 results.fpTR=0;
17 results.fnTR=0;
18
19 results.tpVS=0;
20 results.tnVS=0;
21 results.fpVS=0;
22 results.fnVS=0;
23
24 results.tpTS=0;
25 results.tnTS=0;
26 results.fpTS=0;
27 results.fnTS=0;
28 %MLB - Ottiene le informazioni dalla rete e dal target per poter
29 %effettuare dei confronti nodo per nodo
30 numFeatures = size(subsetTR(:,1:end-size(net.W{1,end},1)),2);
31 numTargets = numFeatures+1;
32 [y, a] = forwardPropagation(net,subsetTR(:,1:numFeatures));
33 TargetTR = subsetTR(:,numTargets:end);
34 yTR=y{N};
35
36 %MLB - Valuta nodo per nodo confrontando con il training set :
37 % - true positive;
38 % - true negative;
39 % - false positive;
40 % - false negative;
41
42
43 for k=1:size(subsetTR,1)
44     if and((yTR(k) >= 0.5), (TargetTR(k) == 1))
45         results.tpTR = results.tpTR+1;
46     elseif and((yTR(k) < 0.5), (TargetTR(k) == 0))
47         results.tnTR = results.tnTR+1;
48     elseif and((yTR(k) < 0.5), (TargetTR(k) == 1))
49         results.fnTR = results.fnTR+1;
50     elseif and((yTR(k) >= 0.5), (TargetTR(k) == 0))

```

```

51         results.fpTR=results.fpTR+1;
52
53     end
54
55 %MLB - Ottiene le informazioni dalla rete e dal target per poter
56 %effettuare dei confronti nodo per nodo
57 [y, a] = forwardPropagation(net,subsetVS(:,1:numFeatures));
58 TargetVS=subsetVS(:,numTargets:end);
59 yVS=y{N};
60 %MLB - Valuta nodo per nodo confrontando con il validation set :
61 % - true positive;
62 % - true negative;
63 % - false positive;
64 % - false negative;
65 for k=1:size(subsetVS,1)
66     if and((yVS(k) >= 0.5), (TargetVS(k) == 1))
67         results.tpVS = results.tpVS+1;
68     elseif and((yVS(k) < 0.5), (TargetVS(k) == 0))
69         results.tnVS = results.tnVS+1;
70     elseif and((yVS(k) < 0.5), (TargetVS(k) == 1))
71         results.fpVS = results.fpVS+1;
72     elseif and((yVS(k) >= 0.5), (TargetVS(k) == 0))
73         results.fnVS=results.fnVS+1;
74     end
75 end
76
77
78
79 TargetTS=subsetTS(:,numTargets:end);
80 yTS=evalin('base','yTS');
81 %MLB - Valuta nodo per nodo confrontando con il test set :
82 % - true positive;
83 % - true negative;
84 % - false positive;
85 % - false negative;
86
87 for k=1:size(subsetTS,1)
88     if and((yTS(k) >= 0.5), (TargetTS(k) == 1))
89         results.tpTS = results.tpTS+1;
90     elseif and((yTS(k) < 0.5), (TargetTS(k) == 0))

```

```

91         results.tnTS = results.tnTS+1;
92     elseif and((yTS(k) < 0.5), (TargetTS(k) == 1))
93         results.fpTS = results.fpTS+1;
94     elseif and((yTS(k) >= 0.5), (TargetTS(k) == 0))
95         results.fnTS=results.fnTS+1;
96     end
97 end
98
99 %MLB - A partire dai risultati ottenuti calcola sia per il validation
100 %set che per il training set :
101 % - precision;
102 % - recall;
103 % - specificity;
104 % - accuracy;
105
106 results.precisionTR = (results.tpTR+1) / (results.tpTR + results.fpTR+1);
107 results.recallTR = (results.tpTR+1) / (results.tpTR + results.fnTR+1);
108 results.specifityTR = (results.tnTR+1) / (results.tnTR + results.fpTR+1);
109 results.accuracyTR = (results.tpTR + results.tnTR+1) / (results.tpTR +
110                         results.fpTR + results.tnTR + results.fnTR+1);
111
112
113 results.precisionVS = (results.tpVS+1) / (results.tpVS + results.fpVS+1);
114 results.recallVS = (results.tpVS+1)/ (results.tpVS + results.fnVS+1);
115 results.specifityVS = (results.tnVS+1) / (results.tnVS + results.fpVS+1);
116 results.accuracyVS = (results.tpVS + results.tnVS+1) / (results.tpVS +
117                         results.fpVS + results.tnVS + results.fnVS+1);
118
119 results.precisionTS = (results.tpTS+1) / (results.tpTS + results.fpTS+1);
120 results.recallTS = (results.tpTS+1)/ (results.tpTS + results.fnTS+1);
121 results.specifityTS = (results.tnTS+1) / (results.tnTS + results.fpTS+1);
122 results.accuracyTS = (results.tpTS + results.tnTS+1) / (results.tpTS +
123                         results.fpTS + results.tnTS + results.fnTS+1);
124
125 return;
126
127 end

```