

# “Intelligent Systems” – Project

Mariano Basile, Angelo Buono

24th June 2016



# Contents

<b>1</b>	<b>Introduzione</b>	<b>4</b>
1.1	Descrizione del problema . . . . .	4
1.2	Descrizione dei dati . . . . .	4
1.3	Obiettivo . . . . .	5
<b>2</b>	<b>Organizzazione dei dati</b>	<b>5</b>
2.1	Caricamento dei dati . . . . .	5
2.2	Associazione tra dati e posizione/attività . . . . .	6
<b>3</b>	<b>Neural Network</b>	<b>8</b>
3.1	Generazione della Rete Neurale . . . . .	8
3.2	Riconoscimento delle Rete Neurale . . . . .	9
<b>4</b>	<b>Mamdani Fuzzy Inference System</b>	<b>10</b>
4.1	Operazioni Preliminari . . . . .	10
4.2	Model fitting . . . . .	11
4.3	Costruzione del sistema fuzzy (Mamdani-type) . . . . .	12
4.4	Definizione delle regole . . . . .	16
4.5	Riconoscimento del sistema Fuzzy Mamdani . . . . .	17
<b>5</b>	<b>Sugeno_Takagi_Kang Fuzzy Inference System</b>	<b>18</b>
5.1	Approccio al problema . . . . .	18
5.2	Generazione del sistema ANFIS . . . . .	18
5.3	Osservazioni sulle scelte . . . . .	19
5.4	Error plot (Checking - Training) . . . . .	20
5.5	Riconoscimento del sistema Fuzzy Sugeno . . . . .	21
<b>6</b>	<b>Codice</b>	<b>22</b>
6.1	main.m . . . . .	22
6.2	inputs_loading.m . . . . .	24
6.3	importfile.m . . . . .	25
6.4	features_min_mean_max.m . . . . .	26

6.5	f_args_splitting.m . . . . .	27
6.6	compute_errors.m . . . . .	28
6.7	compute_network.m . . . . .	29
6.8	features_analysis.m . . . . .	30
6.9	compute_mf_intervals.m . . . . .	31
6.10	mamdani_recognition.m . . . . .	32
6.11	sugeno_recognition.m . . . . .	33

# 1 Introduzione

## 1.1 Descrizione del problema

Il progetto richiede l'analisi di un insieme di dati medici. Il contesto di lavoro è la dermatologia e, in particolare, la terapia a compressione per mezzo di bendaggi per il trattamento di ulcere venose alle gambe. In dermatologia, le ulcere venose sono lesioni piuttosto frequenti della pelle delle gambe a causa di una cattiva e compromessa circolazione del sangue. Per riparare tali ulcere, la circolazione del sangue deve essere potenziata sfruttando il muscolo del polpaccio, il quale, permette al sangue di circolare indietro verso il cuore. La via più comune per ristabilire la circolazione del sangue è la terapia a compressione. In quest'ultima il medico utilizza nei confronti del paziente un bendaggio che viene applicato fornendo una certa pressione sulla zona pelle in cui è presente l'ulcera. Tale pressione permette di riparare l'ulcera nel giro di pochi mesi. La pressione applicata sull'ulcera, quindi l'efficienza della terapia, dipende da diverse fattori:

- il tipo di bendaggio
- la corretta applicazione del bendaggio: un'accurata pressione dovrebbe essere applicata in parti differenti della gamba a seconda della posizione dell'ulcera. Inoltre, la pressione dovrebbe rimanere il più possibile costante tra le differenti applicazioni del bendaggio
- il tipo di attività svolta dal paziente durante la terapia

## 1.2 Descrizione dei dati

I dati si riferiscono a 10 volontari. Un bendaggio a compressione è stato applicato al loro polpaccio. Tre sensori sono stati applicati in tre differenti posizioni al fine di misurare la pressione sotto-bendaggio. Ogni volontario indossa il bendaggio per 12 minuti. Durante questo tempo, il volontario svolge differenti attività o mantiene differenti posizioni. Ogni attività/posizione è svolta/mantenuta per circa 3 minuti. I sensori misurano la pressione campionando ogni 82ms. Le posizioni/attività prese in considerazione sono le seguenti:

- posizione supina
- dorsiflessione
- camminata
- salire le scale

I dati sono organizzati come di seguito:

È fornita per ogni volontario una cartella. In ogni cartella sono presenti 4 file, uno per ogni posizione/attività svolta. Ogni file contiene i campionamenti effettuati dai 3 sensori (nelle

prime 3 colonne), e il corrispondente tempo di campionamento (nell'ultima colonna). I campionamenti presenti si riferiscono a valori di resistenze elettriche e sono espressi in Ohms.

### 1.3 Obiettivo

L'obiettivo è doppio: da un lato, individuare il tipo di posizione/attività del volontario analizzando i valori di pressione del bendaggio, quindi distinguere tra posizione supina, dorsiflessione, camminata e salire le scale; dall'altro lato è necessario individuare l'intervallo di tempo più piccolo, necessario e sufficiente, al fine di indentificare la posizione/attività.

## 2 Organizzazione dei dati

### 2.1 Caricamento dei dati

I dati relativi ai campionamenti dei 3 sensori associati ad ognuna delle 4 posizioni di ciascuno dei 10 volontari sono stati caricati in un **cell array**, **DATA**, attraverso la funzione

**inputs\_loading()** [vedi 6.2]

definita ad-hoc per reperire tali dati dai file excel relativi ad ogni volontario. Tale cell array è costituito da 10x16 celle in cui per i dieci volontari abbiamo i valori dei 3 sensori e dell'istante di campionamento per le 4 posizioni/attività:

POSITION 1: "Dorsiflex"				POSITION 2: "Stairs"			
Sensor 1	Sensor 2	Sensor 3	Time	Sensor 1	Sensor 2	Sensor 3	Time
1	2120x1 double	2120x1 double	2120x1 double	2094x1 double	2094x1 double	2094x1 double	2094x1 double
2	2093x1 double	2093x1 double	2093x1 double	2046x1 double	2046x1 double	2046x1 double	2046x1 double
3	2107x1 double	2107x1 double	2107x1 double	2108x1 double	2108x1 double	2108x1 double	2108x1 double
4	2170x1 double	2170x1 double	2170x1 double	2101x1 double	2101x1 double	2101x1 double	2101x1 double
5	2159x1 double	2159x1 double	2159x1 double	2096x1 double	2096x1 double	2096x1 double	2096x1 double
6	2109x1 double	2109x1 double	2109x1 double	2118x1 double	2118x1 double	2118x1 double	2118x1 double
7	2126x1 double	2126x1 double	2126x1 double	2149x1 double	2149x1 double	2149x1 double	2149x1 double
8	2121x1 double	2121x1 double	2121x1 double	2113x1 double	2113x1 double	2113x1 double	2113x1 double
9	2122x1 double	2122x1 double	2122x1 double	2096x1 double	2096x1 double	2096x1 double	2096x1 double
10	2037x1 double	2037x1 double	2037x1 double	2046x1 double	2046x1 double	2046x1 double	2046x1 double

Figure 1: (Parte della) Struttura del cell-array DATA

L'associazione della posizione/attività da ora in avanti sarà implicitamente la seguente:

- Position 1 = Dorsiflessione
- Position 2 = Salire le scale
- Position 3 = Posizione supina
- Position 4 = Camminare

## 2.2 Associazione tra dati e posizione/attività

In seguito al caricamento dei dati ci troviamo nella situazione in cui una posizione/attività è identificata dai valori dei 3 sensori campionati in 180s ad un rate di 84ms: ciò significa avere  $3 \cdot (180/84) \cdot 1000$ , circa 6000, feature per la rappresentazione di una posizione. Inoltre il sample a disposizione è effimero, infatti il numero di campioni è 40: i dati di 10 volontari per le 4 diverse posizioni/attività. Risulta quindi necessario organizzare in modo opportuno tali dati.

Per aumentare il numero di samples a disposizione si è pensato di ridurre l'intervallo di tempo necessario all'identificazione di una posizione: l'idea è suddividere l'intervallo dei 180s, intervallo di tempo originario in cui il segnale viene campionato, e considerare ogni sottointervallo ottenuto come un nuovo campione.

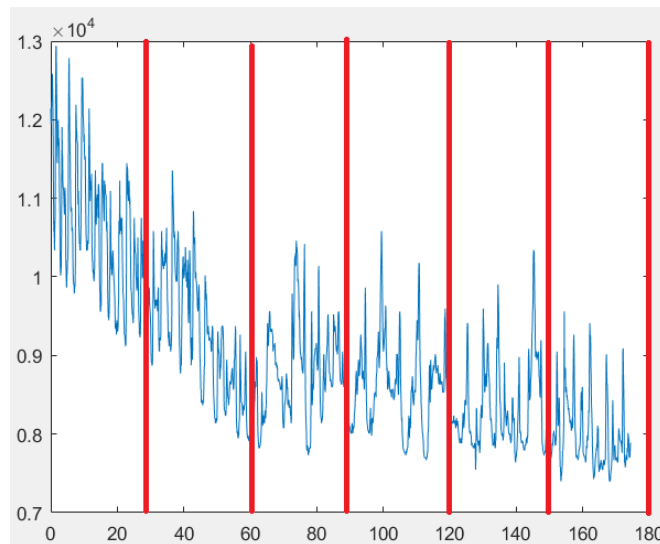


Figure 2: Sensore 1 di Dorsiflessione del Volontario 1 suddiviso in 6 intervalli

Sperimentalmente abbiamo notato che suddividere in 6 intervalli permette di ottenere buoni risultati. Innanzitutto abbiamo ridotto l'intervallo di identificazione, quindi di riconoscimento, di una posizione/attività da 180s a 30s, inoltre abbiamo incrementato

il numero di sample da 40 a 240 e ridotto il numero di feature che rappresentano una posizione/attività. Tutto è stato scalato di un fattore 6.

Malgrado la diminuzione del numero di feature, il loro ammontare è tuttavia ancora troppo elevato per poter essere processate. A tal proposito si è pensato di estrapolare le features più rilevanti per ognuno dei sottointervalli considerati: il primo passo è stato quindi “sintetizzare” tali features in modo da ottenere delle features riassuntive.

Per sintetizzare le features si usa un’approccio simile al precedente: l’idea è suddividere ciascun intervallo individuato in sotto-intervalli e riassumere le feature presenti in tali sottointervalli mediante degli indici, il minimo, il massimo ed il valor medio. Analogamente alla suddivisione precedente quindi, si è suddiviso l’intervallo di identificazione, di 30s, in 6 sotto-intervalli, di circa 5s, ed in essi sono stati calcolati i 3 indici. In questo modo abbiamo sintetizzato le circa 1000, (cioè  $3 \cdot (30/84) \cdot 1000$ ), features in sole 54 (3 indici \* 6 sotto-intervalli \* 3 sensori ) features.

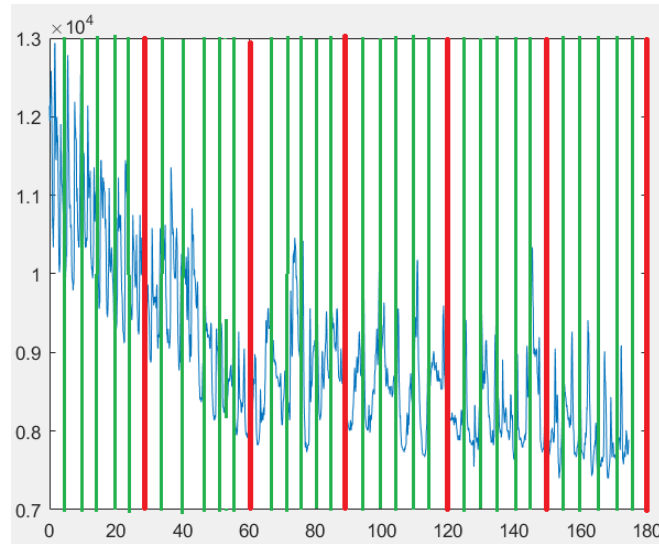


Figure 3: Sensore 1 di Dorsiflessione del Volontario 1 suddiviso in 6 intervalli di 6 sotto-intervalli ciascuno

Questo procedimento di manipolazione dei dati è eseguito dalla

**features\_min\_mean\_max(DATA, 6, 6)** [vedi 6.4],

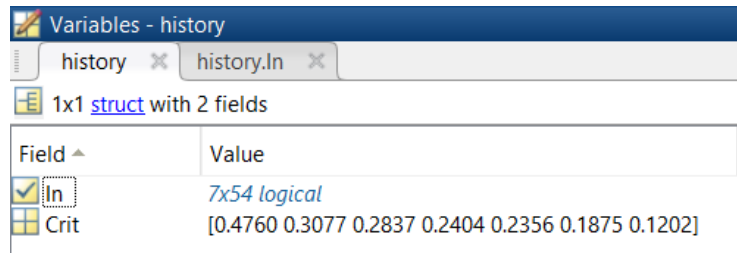
che richiede come input il cell-array DATA, il numero di intervalli in cui splittare (timesplit) ed il numero di sottointervalli per la sintesi delle feature (accuracy), mentre ritorna due matrici: **fs\_inputs**, di dimensione 240x54 e contenenti gli input, e **fs\_targets**, di dimensione 240x4 e contenente i target relativi ai campioni.

Prima di eseguire la selezione delle feature più rilevanti, preleviamo un set di campioni per il test, quello che sarà il nostro testing set per la rete neurale e per i sistemi fuzzy. Tale operazione è eseguita dalla funzione

`f_args_splitting( [fs_inputs, fs_targets], test_size)` [vedi 6.5],

che preleva un numero pari a `test_size` di righe dalla matrice `fs_inputs` e ritorna quattro matrici: `fs_trainX`, `fs_trainT`, `fs_testX` e `fs_testT` rispettivamente input e target dei campioni selezionati per training e testing.

L'operazione di *'feature selection'* è effettuata dalla funzione predefinita `sequentialfs()`. Per effettuare la selezione bisogna definire una handle-function, **fun** [vedi 6.6], che esegue pattern recognition ritornando l'errore di riconoscimento: si tratta di una funzione parametrica che implementa tale rete neurale; gli altri input sono l'insieme di input, **fs\_trainX**, l'insieme di target, **fs\_trainT**, e opzionalmente un set di opzioni, **opts**. Al termine dell'esecuzione, che può richiedere diverse ore, si ottiene un vettore binario, **fs**, con le feature selezionate ed una struttura, **history**, con una matrice, **history.In**, contenente i vettori binari delle feature selezionate ad ogni iterazione ed un vettore, **history.Crit**, con i valori degli errori percentuali di riconoscimento.



Field	Value
In	7x54 logical
Crit	[0.4760 0.3077 0.2837 0.2404 0.2356 0.1875 0.1202]

Figure 4: Struttura history ed i relativi campi

Le feature selezionate sono 7 ed a questo punto possiamo definire il nostro set di input per il training ed il testing: **final\_trainX**, **final\_trainT**, **final\_testX** e **final\_testT**.

## 3 Neural Network

### 3.1 Generazione della Rete Neurale

Una volta ottenute le features necessarie alla rappresentazione delle posizioni/attività da riconoscere possiamo generare la nostra rete neurale. Per avere una accettabile risultato in uscita dalla rete abbiamo definito la funzione

`compute_network(final_trainX, final_trainT, final_testX, final_testT)`,

che esegue il training sul 90% degli input di **final\_trainX** ed usa il restante 10% per la validation, mentre il test usa gli input di **final\_testX** (prelevati precedentemente). La rete ha 20 hidden-layer, e viene allenata e testata 10 volte; alla fine la funzione ritorna la rete con il riconoscimento migliore. [vedi 6.7]



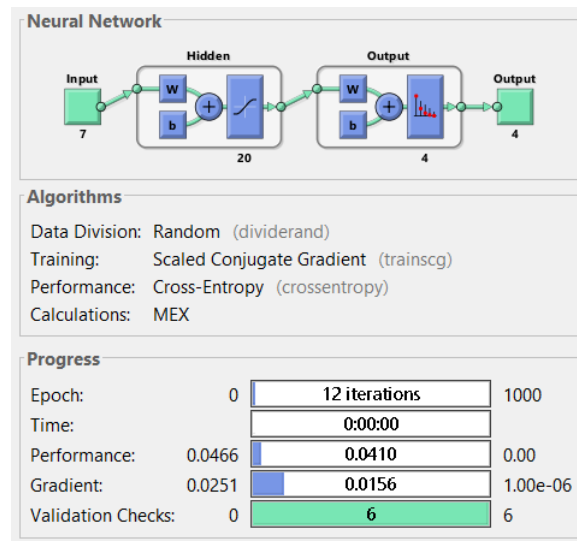


Figure 5: Struttura e training della Rete Neurale

### 3.2 Riconoscimento delle Rete Neurale

La Rete Neurale ottenuta ha una **percentuale di riconoscimento** che varia a seconda del testing set selezionato precedentemente alla fase di feature selection e di training: si va da occasionali 80% a **frequenti 90%**, con rari 100%. La dimensione del testing set è di 36 campioni, ovvero il 15% del sample a disposizione.

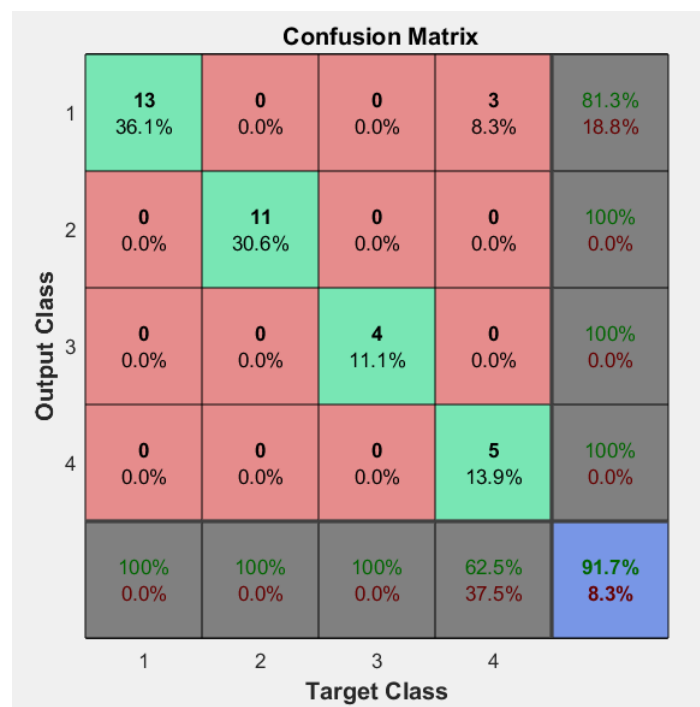


Figure 6: Confusion Matrix della Rete Neurale

## 4 Mamdani Fuzzy Inference System

### 4.1 Operazioni Preliminari

Al fine di procedere con lo sviluppo del sistema Fuzzy, per mezzo del *Fuzzy Logic Toolbox*, è stato necessario, in prima istanza, compiere una serie di operazioni preliminari. In breve:

- **Selezione di un sottoinsieme di features (4) rispetto a un set di partenza più numeroso (7):** da un set iniziale di 7 features ottenute applicando una *'feature selection'* per mezzo della *sequentialfs()*, come espresso nei paragrafi precedenti, si è giunti a un set comprendente 4 features. Tale procedimento è risultato necessario al fine di non complicare eccessivamente la scrittura delle regole per il sistema Fuzzy.
- **Model fitting:** per ciascuna delle 4 features selezionate è stato poi necessario visualizzare la distribuzione dei valori. Questo ha dato la possibilità, da una parte, di definire il *range* di valori su cui ogni feature risulta essere definita e, dall'altra, individuare le forme più adeguate da assegnare alle varie *membership functions* di ciascuna feature. Per ciascuna delle 4 features è stato necessario definire il numero di membership functions ed individuare gli estremi di definizione di ciascuna di esse: si è scelto un approccio visivo a tal scopo, basato cioè sull'osservazione dell'andamento dei valori relativi ad ogni posizione/attività di ciascuna feature.

Volendo eseguire un'analisi più dettagliata è possibile osservare che:

- Lo script principale realizza le operazioni sopra citate invocando la funzione

**features\_analysis(fs\_trainX, fs\_trainT, history, 4).** [vedi 6.8]

Tale funzione prende in ingresso:

- **fs\_trainX, fs\_trainT:** due matrici, rispettivamente di 207x54 e 207x4, sottomatrici delle matrici di partenza rispettivamente di 240x54 e 240x4 (in quanto decurtate degli elementi necessari ad effettuare poi il test della rete neurale).
- **history** l'indice del sottoinsieme di features selezionate (7). Per i motivi sopra citati non è conveniente lavorare sul sistema fuzzy con un numero così elevato di features in ingresso, pertanto è risultato necessario ridurre ulteriormente tale set. Attraverso il campo *In* in tale struttura è possibile selezionare l'indice relativo alle prime 4 features selezionate dalla *sequentialfs()*.
- **n\_feature\_desiderate:** valore che specifica quante features selezionare. Nel nostro caso sono sufficienti le prime 4 features selezionate, le più rilevanti.

## 4.2 Model fitting

Per la parte relativa al model fitting riteniamo più significativo mostrare quanto ottenuto graficamente piuttosto che procedere, come fatto fin'ora, attraverso una spiegazione del codice. In particolare:

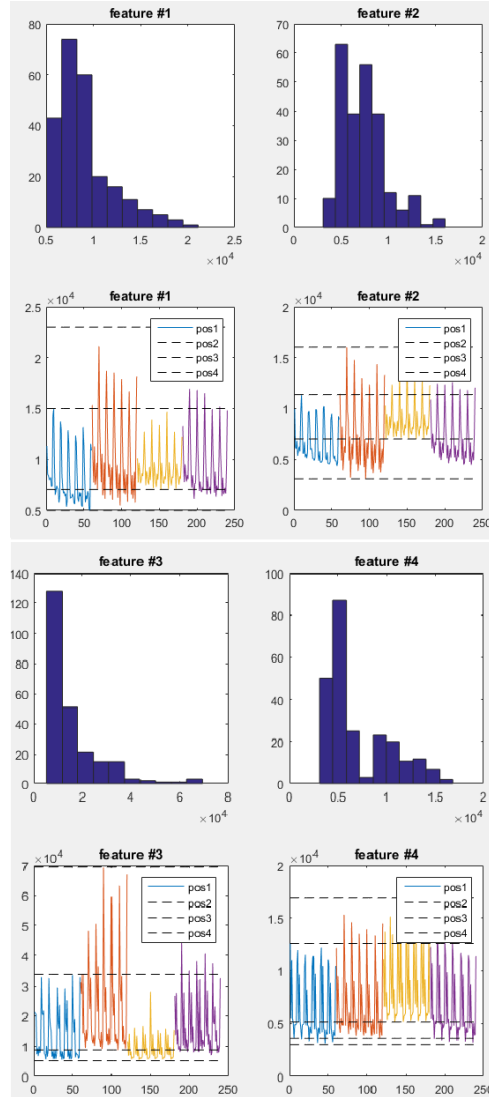


Figure 7: Distribuzione delle posizioni rispetto alle features

Prima di procedere ad esaminare il sistema fuzzy nel dettaglio è necessaria una piccola osservazione sul procedimento utilizzato per stabilire il numero adeguato di mfs e dei relativi intervalli di definizione.

L'approccio utilizzato è un **approccio visivo**: osservando la distribuzione dei valori relativi alle varie posizioni per ciascuna feature si sono individuati intervalli significativi alla distinzione delle posizioni stesse. Una volta individuati tali intervalli e quindi il

numero di mfs da utilizzare, la loro forma è stabilita cercando di matchare l'andamento della distribuzione dei valori relativi alla feature in esame come espresso dal relativo istogramma. Quanto appena descritto per una singola feature è ripetuto per tutte le rimanenti. In questo modo è possibile procedere con la scrittura delle regole.

### 4.3 Costruzione del sistema fuzzy (Mamdani-type)

La configurazione adottata è esattamente la seguente:

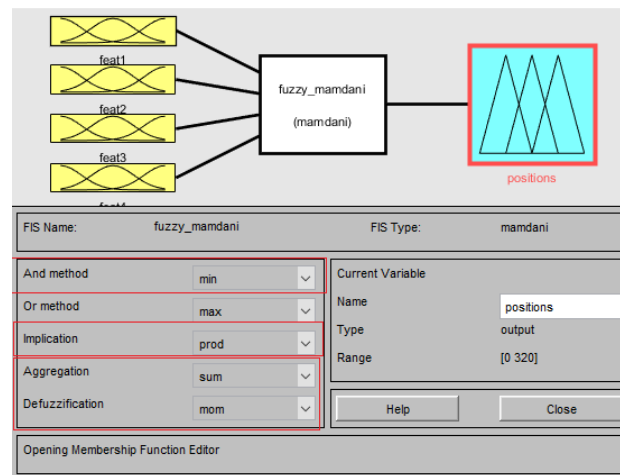


Figure 8: Impostazioni di configurazione generali sistema fuzzy (Mamdani type)

Nella figura di cui sopra è opportuno sottolineare la scelta dell'operatore **“somma”** per realizzare l' **aggregazione** dei fuzzy set prodotti per mezzo delle regole e del metodo **“mean of maxima”** come metodo per la **defuzzificazione**.

L' aggregatore **“somma”** risulta essere il più adatto al fine di ottenere un corretto riconoscimento delle posizioni: tale affermazione è legata al modo con cui sono state costruite le regole nel caso di ambiguità, ovvero stessi antecedenti e diversi conseguenti. L'approccio in questo caso è quello di contare le occorrenze, di ciascuna posizione, per ciascuna feature in ognuna delle mfs individuate; dopo di ch , con tali occorrenze, si calcola il peso da assegnare a ciascuna delle regole ambigue.

La defuzzificazione attraverso il metodo **“mean of maxima”**   strettamente necessaria poich  le posizioni risultano essere logicamente indipendenti tra loro.   doveroso quindi considerare come output del sistema il valore avente grado di appartenenza  $\mu_a$  a maggiore. In caso di ambiguit  tra pi  posizioni, se si fosse scelto il metodo **“center of area”** si sarebbe finiti col ottenere, come risposta dal sistema, un valore numerico centrato tra le ambiguit  stesse producendo, di conseguenza, un risultato errato.

Osserviamo adesso le configurazioni utilizzate per ciascuna delle 4 features in ingresso al sistema e per la variabile d'uscita avente 4 fuzzy values rispettivamente pos1, pos2, pos3, pos4.

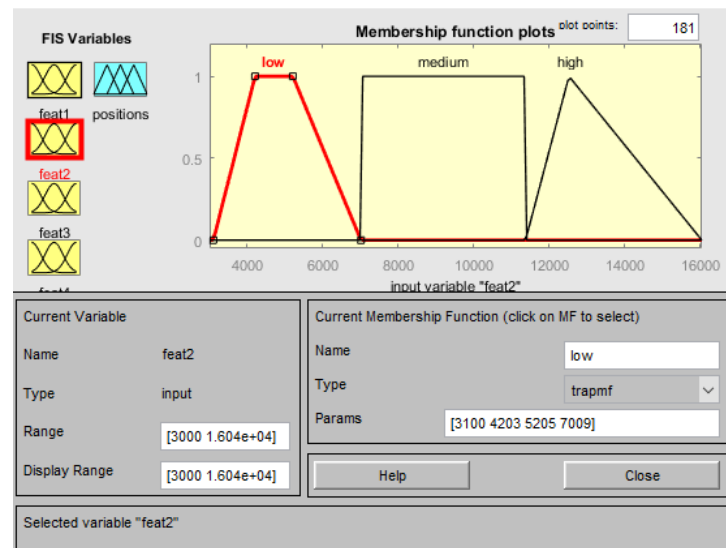
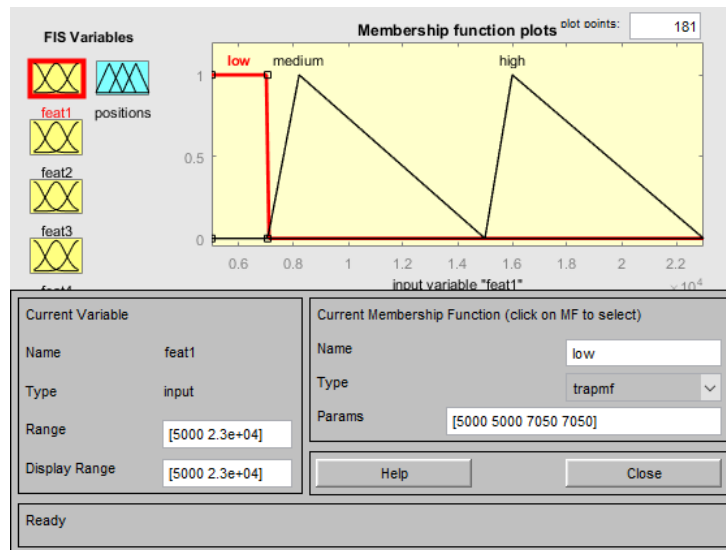


Figure 9: Fuzzy system: Definizione Mfs per feature1 e feature2.

Riportiamo di seguito, al fine di una più rapida consultazione, i range di definizione per le features di cui sopra, così come gli estremi di definizione per ciascuna della Mfs.

Range di definizione:

Feature	Estremo inferiore	Estremo superiore
1	5000	23000
2	3000	16040

Dominio Mfs:

Feature	MF1 (Low)	MF2 (Medium)	MF3 (High)
1	5000; 7050	7050; 15000	15000; 23000
2	3100; 7009	7009; 11370	11370; 16040

Infine per feature3 e feature4:

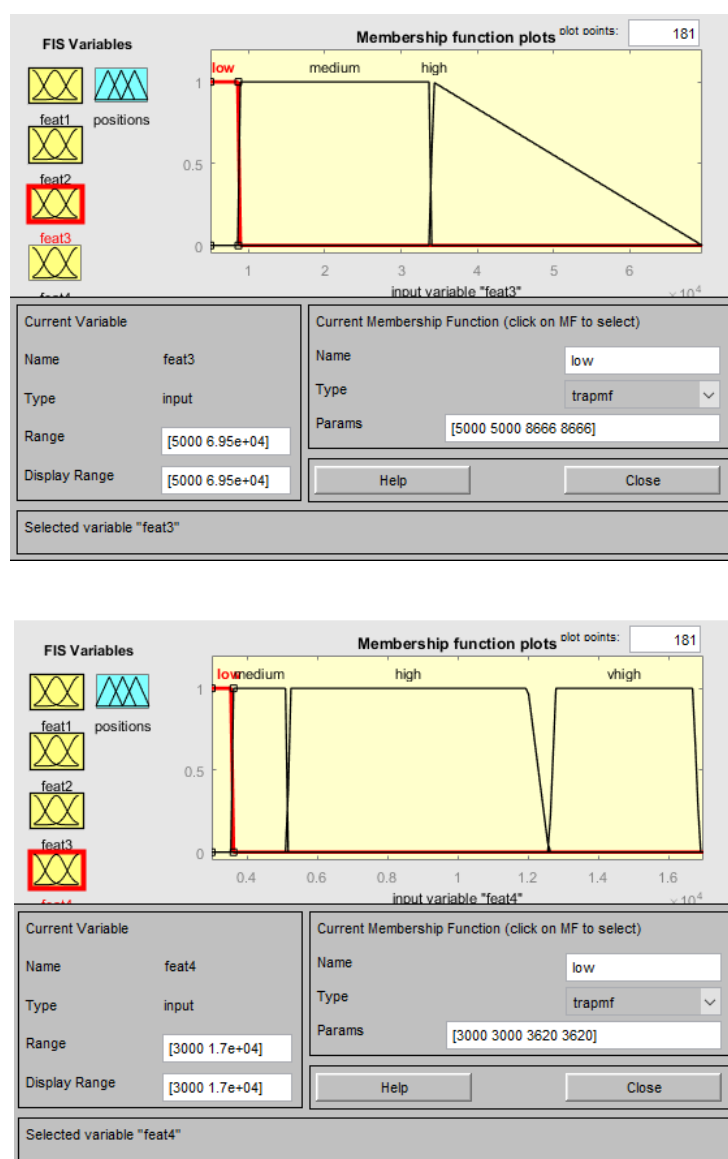


Figure 10: Fuzzy system: Definizione Mfs per feature3 e feature4.

Riportiamo anche in questo caso i range di definizione per le features di cui sopra, così come gli estremi di definizione per ciascuna della Mfs.

Range di definizione:

Feature	Estremo inferiore	Estremo superiore
3	5000	69500
4	3000	17000

Dominio Mfs:

Feature	MF1 (Low)	MF2 (Medium)	MF3 (High)	MF4 (Vhigh)
3	5000; 8666	8666; 33800	33800; 69500	
4	3000; 3620	3620; 5140	5140; 12600	12600; 16900

Infine per la variabile d'uscita abbiamo:

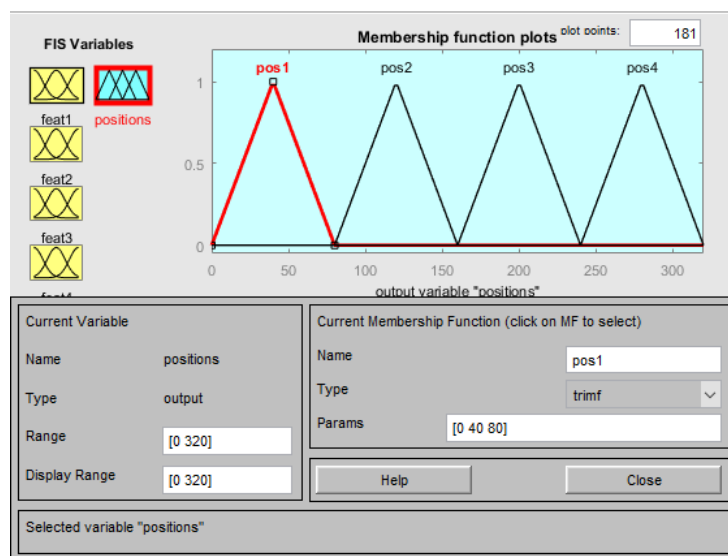


Figure 11: Fuzzy system: Definizione fuzzy values per l'output.

Riportiamo anche in questo caso i range di definizione per le features di cui sopra, così come gli estremi di definizione per ciascuna della mfs.

Range di definizione:

Estremo inferiore	Estremo superiore
0	320

Dominio Fuzzy values:

Fuzzy values	pos1	pos2	pos3	pos4
	0; 80	80; 160	160; 240	240; 320

#### 4.4 Definizione delle regole

L'idea alla base della determinazione delle regole è basata, come anticipato precedentemente, su un approccio visivo: osservando le singole posizioni separatamente per ognuna delle 4 features al fine di identificare le Mfs coinvolte per il riconoscimento. Bisogna, ancora una volta, sottolineare che non tutte le posizioni possono essere identificate per mezzo di regole univoche: ci sono casi in cui gli stessi antecedenti identificano posizioni (conseguenti) differenti.

Le regole identificate sono:

Feature1	Feature2	Feature3	Feature4		Pos1	Pos2	Pos3	Pos4
Low	Low	Low	Medium		*			
Medium	Low	Low	High		*			
Low	Low	Low	High		*			
Medium	Medium	Medium	Medium			*		
Medium	Medium	Low	High				*	
Medium	Low	Medium	Medium		*	*		*
Medium	Low	Medium	High			*		*
Medium	Medium	Medium	High				*	*

A questo punto per cercare di risolvere l'ambiguità si è proceduto come descritto in 4.3. Riassumiamo per mezzo di tabelle i risultati ottenuti:

Feature1					Feature2			
	Low	Medium	High			Low	Medium	High
Posizione					Posizione			
1	28	25	0		1	38	15	0
2	9	36	5		2	29	16	5
3	1	53	0		3	1	47	6
4	11	36	4		4	33	12	6
Totale	49	150	9		Totale	101	90	17

Feature3					Feature4			
	Low	Medium	High			Low	Medium	High
Posizione					Posizione			
1	28	25	0		1	32	17	0
2	0	42	6		2	24	22	4
3	36	18	0		3	1	43	9
4	6	40	5		4	20	23	3
Totale	70	125	11		Totale	77	105	16

Figure 12: Fuzzy system: Individuazione dei pesi per le regole.



Il fine è stato quindi quello di associare dei pesi alle regole e allo stesso tempo dare una certa significatività (peso) ai pesi stessi. Riportiamo quindi il listato delle regole definite:

```

1. If (feat1 is low) and (feat2 is low) and (feat3 is low) and (feat4 is medium) then (positions is pos1) (1)
2. If (feat1 is medium) and (feat2 is low) and (feat3 is low) and (feat4 is medium) then (positions is pos1) (1)
3. If (feat1 is low) and (feat2 is low) and (feat3 is low) and (feat4 is high) then (positions is pos1) (1)
4. If (feat1 is medium) and (feat2 is medium) and (feat3 is medium) and (feat4 is medium) then (positions is pos2) (1)
5. If (feat1 is medium) and (feat2 is medium) and (feat3 is low) and (feat4 is high) then (positions is pos3) (1)
6. If (feat1 is medium) then (positions is pos1) (0.17)
7. If (feat1 is medium) then (positions is pos2) (0.24)
8. If (feat1 is medium) then (positions is pos3) (0.36)
9. If (feat1 is medium) then (positions is pos4) (0.23)
10. If (feat2 is low) then (positions is pos1) (0.38)
11. If (feat2 is low) then (positions is pos2) (0.29)
12. If (feat2 is low) then (positions is pos4) (0.33)
13. If (feat2 is medium) then (positions is pos1) (0.17)
14. If (feat2 is medium) then (positions is pos2) (0.18)
15. If (feat2 is medium) then (positions is pos3) (0.52)
16. If (feat2 is medium) then (positions is pos4) (0.13)
17. If (feat3 is medium) then (positions is pos1) (0.2)
18. If (feat3 is medium) then (positions is pos2) (0.34)
19. If (feat3 is medium) then (positions is pos3) (0.14)
20. If (feat3 is medium) then (positions is pos4) (0.32)
21. If (feat4 is medium) then (positions is pos1) (0.42)
22. If (feat4 is medium) then (positions is pos2) (0.31)
23. If (feat4 is medium) then (positions is pos4) (0.26)
24. If (feat4 is high) then (positions is pos1) (0.16)
25. If (feat4 is high) then (positions is pos2) (0.21)
26. If (feat4 is high) then (positions is pos3) (0.41)
27. If (feat4 is high) then (positions is pos4) (0.22)

```

Figure 13: Fuzzy system: Set di regole individuate.

## 4.5 Riconoscimento del sistema Fuzzy Mamdani

Per verificare il numero di posizioni correttamente riconosciute dal sistema fuzzy è implementata una procedura, la `mamdani_recognition()` [vedi 6.10], che sostanzialmente fa utilizzo della funzione `evalfis()`, invocata utilizzando come parametri attuali il set di valori (segnali) su cui calcolare il riconoscimento e il sistema fuzzy sviluppato salvato in una variabile denominata “fuzzy\_mamdani”. Il valore di ritorno di tale `evalfis()` rappresenta il valore numerico ottenuto per mezzo di un’operazione di defuzzificazione che identifica una delle posizioni. Poichè il sistema Mamdani non impara attraverso training, la **percentuale di riconoscimento** è calcolata **sull’intero set di dati** a disposizione (training + test) ed è **del 52%**.

## 5 Sugeno \_ Takagi \_ Kang Fuzzy Inference System

### 5.1 Approccio al problema

L'approccio *iniziale* per lo sviluppo del Sugeno-type fuzzy inference system è basato sull'utilizzo del ANFIS-GUI-tool disponibile nell'ambiente. Invocato quindi il comando *anfisedit* di Matlab abbiamo d'apprima provveduto a caricare rispettivamente training set, testing set, checking set. A tal proposito abbiamo predisposto, preventivamente, i suddetti set in una forma come quella indicata dalle specifiche, ovvero una matrice in cui nell'ultima colonna sono presenti i rispettivi valori di posizione attesi. Ricordando che una posizione era fin'ora rappresentata da un vettore di 4 elementi è stato necessario modificare la matrice (contenente le posizioni attese) in un vettore e poi effettuare il concatenamento di tale vettore alla matrice rappresentante il training set. Quest'ultimo, così come il testing set, risultano essere gli stessi utilizzati nella rete neurale: ciò ha permesso quindi di rendere confrontabile i risultati, in termini di riconoscimento, ottenuti dai vari sistemi.

### 5.2 Generazione del sistema ANFIS

Nella generazione del sistema si è scelta come tecnica di partizionamento il *Grid-partitioning*, ovvero l'impostazione di default, in quanto l'alternativa, *Subtractive Clustering*, porta a risultati peggiori (in termini di mse). Si è poi specificato il numero di Mfs per ciascun input, il tipo di Mfs ovvero la forma, il tipo di Mfs per l'uscita, il tipo di metodo utilizzato dalla rete per l'apprendimento, il numero di epoche e infine la threshold sull'errore per definire un criterio di stop per l'allenamento. I parametri sono scelti *sperimentalmente*:

Numero di Mfs (per input)	Tipo di Mfs	Tipo di Mf (output)
6	Gaussiana	Costante

Optim.method	Epoche	Error Tolerance
Ibrido	10	0

La struttura ottenuta risulta quindi essere la seguente:

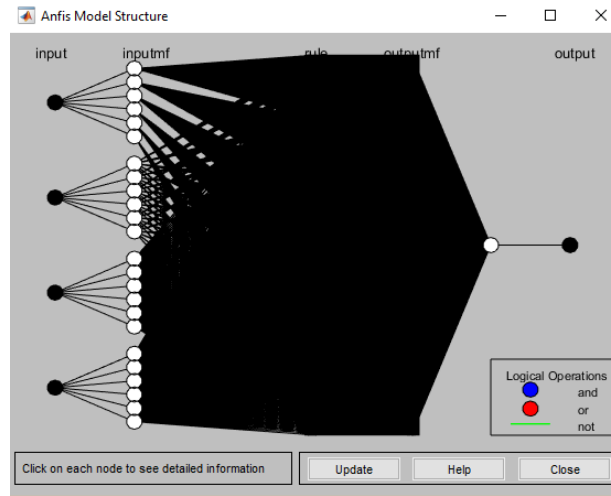


Figure 14: Struttura del sistema Anfis

### 5.3 Osservazioni sulle scelte

- **Numero di Mfs - Apprendimento della rete**

Una prima osservazione è relativa al numero di Mfs definite per ogni input. Il valore 6, presentato poco sopra, risulta essere un buon trade-off tra la complessità ottenuta, intesa come numero di regole generate, e l'errore relativo al training. Un numero più basso di Mfs diminuisce la complessità, essendo quest'ultima espressa da una relazione del tipo  $\prod_i^{\#inputs} \#Mfs_i$ , ma al contempo aumenta l'errore. Con 6 Mfs per ognuno dei 4 input si ottiene un numero di regole pari a  $6^4 = 1296$  e un valore dell'errore (mse) durante la fase di training pari al 53.8%.

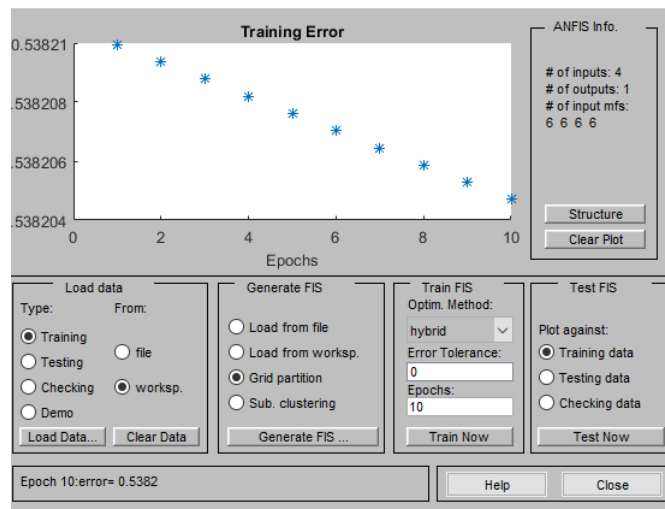


Figure 15: Apprendimento della rete: *Anfis training error* dopo 10 epoche.

- **Tipo di Mfs (input) - Tipo di Mf (output)**

Molteplici prove hanno permesso di identificare come tipo di Mf, per gli input, la Gaussiana. La singola uscita della rete risulta essere una funzione di grado 0 (costante): la motivazione sta nel fatto che una combinazione lineare degli ingressi per l'uscita comporta un valore dell'errore sul training non accettabile.

- **Optmiz.method - Epoche - Error Tolerance**

L'allenamento basato sul metodo ibrido, una combinazione cioè del metodo dei minimi quadrati e del backpropagation gradient descent, garantisce un'errore sul training decisamente più basso rispetto all'approccio basato solo su backpropagation. Il numero (massimo) di epoche necessarie per l'apprendimento è stato impostato a 10 (default 3) in quanto l'errore sul training decresce [vedi 15] anche se di un valore molto basso ( $10^{-6}$ /epoca). L'error tolerance definisce l'altro criterio di stop durante la fase di training: non appena il valore dell'errore risulta essere minore o uguale alla threshold specificata l'apprendimento termina. Un valore pari a 0 cerca di minimizzare l'errore.

## 5.4 Error plot (Checking - Training)

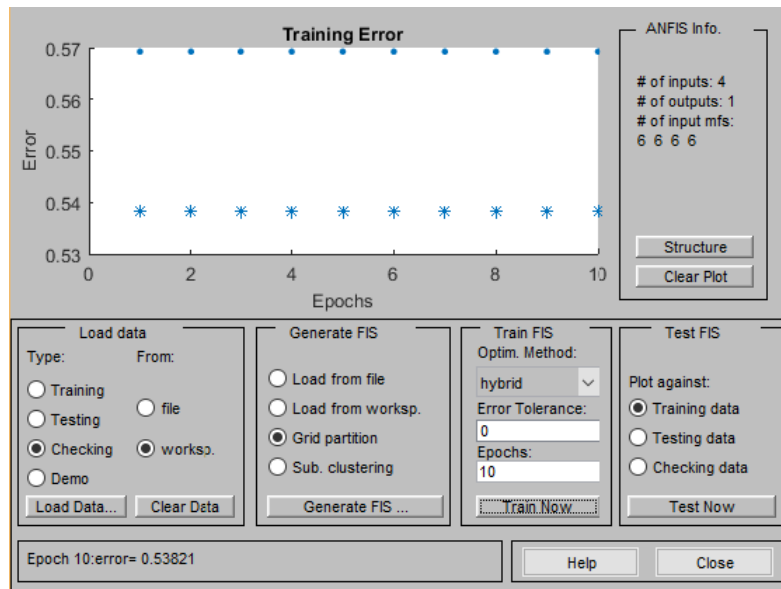


Figure 16: Checking - Training error plot.

Il plot mostra l'errore sul checking (punti) nella parte superiore della figura e l'errore sul training (asterischi) nella parte inferiore. Come si nota dall'immagine l'errore sul checking rimane pressochè costante al variare del numero di epoche: ciò rappresenta l'impossibilità da parte del sistema di fare overfitting.

Di conseguenza l'utilizzo di un set di dati, rappresentanti un checking-set, al fine di evitare tale situazione è trascurabile. L'affermazione è provata osservando che l'errore sul training rimane costante per tutte le epoche considerate. Nel caso di overfitting avremmo assistito al decrescere dell'errore sul training, a partire dall'epoca in cui l'overfitting si sarebbe manifestato (cioè a partire dall'epoca in cui l'errore sul checking sarebbe iniziato a crescere).

## 5.5 Riconoscimento del sistema Fuzzy Sugeno

Quanto presentato fin'ora, come sottolineato all'inizio di tale sezione, è reso possibile dall'Anfis-Gui tool. Al fine di automatizzare il processo di creazione della Anfis, così come il relativo addestramento, si è deciso di ricorrere all'utilizzo delle funzioni disponibili nell'ambiente. In particolare si è ricorso all'utilizzo delle funzioni `genfis1()`, per la creazione della rete, e della `anfis()` per l'allenamento.

Anche qui, per verificare il numero di posizioni correttamente riconosciute dal sistema è implementata una procedura, la `sugeno_recognition()` [vedi 6.11], che sostanzialmente fa utilizzo della funzione `evalfis()`, invocata utilizzando come parametri attuali il set di valori (segnali) su cui calcolare il riconoscimento e il sistema fuzzy sviluppato e salvato in una variabile denominata "fuzzy\_sugeno". Il valore di ritorno di tale `evalfis()` rappresenta il valore numerico ottenuto per mezzo di un operazione di defuzzificazione.

Per calcolare la percentuale di riconoscimento abbiamo collezionato i riconoscimenti del sistema allenato con differenti training set. In particolare abbiamo collezionato un set di 10 valori, la cui percentuale media di riconoscimento è **80% sul training set e 38% sul testing set**.

Volendo confrontare i due sistemi ottenuti abbiamo che: il Mamdani ha riconoscimento medio del 52%, indipendentemente dal training e testing set (in quanto non impara con training), mentre il Sugeno ha un riconoscimento migliore sul training set, 80%, mentre inferiore sul testing set, 38%.

In sintesi:

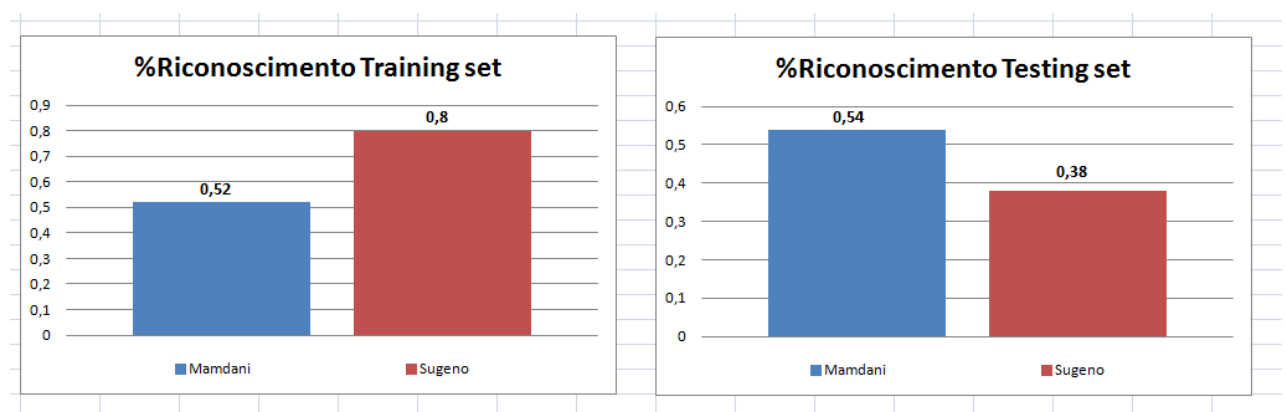


Figure 17: Confronto tra le percentuali di riconoscimento dei sistemi fuzzy.

## 6 Codice

### 6.1 main.m

Lo script **main.m** è il file principale che, chiamando funzioni secondarie, provvede alla creazione della Rete Neurale per la classificazione delle posizioni/attività e ai sistemi Fuzzy Mamdani e Takagi-Sugeno-Kang.

---

```
%DATA IMPORTING (returning a cell arrays with data for each volunteer)
if(not(exist('DATA','var')))
    DATA = inputs_loading();
end

%DATA MANIPULATION
%
%representing the positions according to min, mean and max values related
%to the sensors in different time intervals (accuracy = # of intervals)
% timeplit = A means splitting the singles sampling in A intervals
%this implies reducing the recognizing times!
% accuracy = B means splitting in B intervals
if(not(exist('fs_inputs','var')&&exist('fs_targets','var'))))
    [fs_inputs, fs_targets] = features_min_mean_max(DATA, 6,6);
end

%DATA SPLITTING (arguments: DATA VALUES and TEST SET SIZE)
%
%    %trainingX & trainingT = INPUTS and TARGET for the TRAINING SET (to reduce)
%    %testingX & testingT = INPUTS and TARGET for the TESTING SET
%test size ~15%
test_size = fix(numel(fs_inputs(:,1))*0.15);
[fs_trainX, fs_trainT, fs_testX, fs_testT] = f_args_splitting( [fs_inputs, fs_targets
    ], test_size);

%FEATURES SELECTION
%
%fs contains indeces of the selected features
if(not(exist('fs','var')&&exist('history','var'))))
    opts = statset('display','iter');
    fun = @(trainingX,trainingT,testingX,testingT)compute_errors(trainingX',trainingT',
        testingX',testingT');

    [fs, history] = sequentialfs(fun, fs_trainX, fs_trainT,'options', opts );
end

%NEURAL NETWORK
%
%computing the NEURAL NETWORK for CLASSIFICATION
%training set according fs
final_trainX = fs_trainX(:,fs);
final_trainT = fs_trainT;
%testing set according fs
final_testX = fs_testX(:,fs);
final_testT = fs_testT;

% Choose a Training Function
if(not(exist('net','var'))))
    net = compute_network(final_trainX, final_trainT, final_testX, final_testT);
end
```

```

%TESTING NN RECOGNITION
%
%trying the network classification accuracy on different test sets
y = net(final_testX');
plotconfusion(final_testT',y);

#####
%FUZZY SYSTEM

%Feature reduction (from 7 to 4) and Model Fitting

%return values:
% feature definition range
% ( 4x2 matrix
% ROW: feature
% COLUMN: min max )

% ling_val: min and max for each position for each feature
% (2x1 cell array.Each cell is a 4x4 matrix
% ROW: feature
% COLUMN: position (min, max) )
% )

[ranges, ling_val] = features_analysis(fs_inputs, fs_targets, history, 4);

%Compute Fuzzy (Mamdani type) recognition percentage.
%(fuzzy_mamdani stores the fuzzy system)

mamdani_recognition_perc = mamdani_recognition(fs_inputs,fs_targets,fs_redux,
    fuzzy_mamdani);

#####
%ANFIS SYSTEM

%Training set
sugeno_trainX = fs_trainX(:,fs_redux);
sugeno_trainT = vec2ind(fs_trainT(:, :)');
sugeno_train = [sugeno_trainX sugeno_trainT'];

%SUGENO FIS:
%
%Generating the Sugeno FIS with Grid Partitioning
fuzzy_sugeno = genfis1(sugeno_train, 6, 'gaussmf', 'constant');

%training Sugeno fis (without the checking set)
fuzzy_sugeno = anfis(sugeno_train, fuzzy_sugeno);

%testing Sugeno recognition over all the data
sugeno_recognition_perc = sugeno_recognition(fs_inputs,fs_targets,fs_redux,fuzzy_sugeno);

%Sugeno: Percetage Recognition on training and testing set

sugeno_test_recognition = sugeno_recognition(fs_testX, fs_testT, fs_redux, fuzzy_sugeno)
sugeno_train_recognition = sugeno_recognition(fs_trainX, fs_trainT, fs_redux,
    fuzzy_sugeno)

```

## 6.2 inputs\_loading.m

La funzione **inputs\_loading()** è scritta ad-hoc per caricare i file relativi ai volontari. Accede ai fogli excel ed organizza i dati in un cell array. Si serve della funzione **importfile()**. [vedi 2.1]

---

```
function V = inputs_loading()
    %importing data

    V = cell(10,16);
    %V is a cell array where:
    %10 volunteers
    %16 features: 1-4: dorsiflexion sensor 1,2,3, TIME
    %               5-8: stairs sensor 1,2,3, TIME
    %               9-12: supine sensor 1,2,3, TIME
    %               13-16: walkin sensor 1,2,3, TIME

    for i=1:10
        %creating paths strings
        f = strcat('Measurements_10_volunteers/Volunteer_', num2str(i));
        f = strcat(f, '/V');
        f = strcat(f, num2str(i));
        s = strcat('S', num2str(i));

        f1 = strcat(f, '_dorsiflexion.xlsx');
        s1 = strcat(s, '_Dorsiflessione');
        [V{i,1},V{i,2},V{i,3},V{i,4}] = importfile(f1, s1, 1, 2200);

        f1 = strcat(f, '_stairs.xlsx');
        s1 = strcat(s, '_Scale');
        [V{i,5},V{i,6},V{i,7},V{i,8}] = importfile(f1, s1, 1, 2200);

        f1 = strcat(f, '_supine.xlsx');
        s1 = strcat(s, '_Sdraiato');
        [V{i,9},V{i,10},V{i,11},V{i,12}] = importfile(f1, s1, 1, 2200);

        f1 = strcat(f, '_walking.xlsx');
        s1 = strcat(s, '_Camminata');
        [V{i,13},V{i,14},V{i,15},V{i,16}] = importfile(f1, s1, 1, 2200);
    end
end
```



## 6.3 importfile.m

La funzione **import()** è stata generata automaticamente in matlab e si interfaccia con i file excel per caricare i dati di un foglio in un file .xlsx.

---

```
function [Sensor1,Sensor2,Sensor3,Times] = importfile(workbookFile, sheetName, startRow,
    endRow)
%IMPORTFILE Import data from a spreadsheet
% [Sensor1,Sensor2,Sensor3,Times] = IMPORTFILE(FILE) reads data from the
% first worksheet in the Microsoft Excel spreadsheet file named FILE and
% returns the data as column vectors.
%
% [Sensor1,Sensor2,Sensor3,Times] = IMPORTFILE(FILE,SHEET) reads from the
% specified worksheet.
%
% [Sensor1,Sensor2,Sensor3,Times] =
% IMPORTFILE(FILE,SHEET,STARTROW,ENDROW) reads from the specified
% worksheet for the specified row interval(s). Specify STARTROW and
% ENDROW as a pair of scalars or vectors of matching size for
% dis-contiguous row intervals. To read to the end of the file specify an
% ENDROW of inf.
%
% Non-numeric cells are replaced with: NaN
%
% Auto-generated by MATLAB on 2016/04/27 12:51:47

%% Input handling

% If no sheet is specified, read first sheet
if nargin == 1 || isempty(sheetName)
    sheetName = 1;
end

% If row start and end points are not specified, define defaults
if nargin <= 3
    startRow = 1;
    endRow = 2200;
end

%% Import the data
[~,~,raw] = xlsread(workbookFile, sheetName, sprintf('A%d:D%d',startRow(1),endRow(1)));
for block=2:length(startRow)
    [~,~,tmpRawBlock] = xlsread(workbookFile, sheetName, sprintf('A%d:D%d',startRow(
        block),endRow(block)));
    raw = [raw;tmpRawBlock]; %#ok<AGROW>
end
raw(cellfun(@(x) ~isempty(x) && isnumeric(x) && isnan(x),raw)) = {' '};

%% Exclude rows with non-numeric cells
I = ~all(cellfun(@(x) (isnumeric(x) || islogical(x)) && ~isnan(x),raw),2); % Find rows
    with non-numeric cells
raw(I,:) = [];

%% Create output variable
I = cellfun(@(x) ischar(x), raw);
raw(I) = {NaN};
data = reshape([raw{:}],size(raw));

%% Allocate imported array to column variable names
Sensor1 = data(:,1);
Sensor2 = data(:,2);
Sensor3 = data(:,3);
Times = data(:,4);
```

## 6.4 features\_min\_mean\_max.m

La funzione `features_min_mean_max()` svolge tutte le operazioni di organizzazione e manipolazione dati, necessarie alla rappresentazione di una posizione/attività. [vedi 2.2]

---

```
function [FS_X, FS_T] = features_min_mean_max( DATA, timesplit, accuracy)
%DATA is a cell array containing the inputs and targets for each volunteer
%accuracy is the number of times to split single volunteer features to
%timesplit is the number of times to split the recognizing interval
%compute min, mean and max indeces

n_volunteers = 10;
n_positions = 4;
n_sensors = 3;

%The features selection will be a matrix FS:
%   columns: S1_Int1_min | S1_Int1_mean | S1_Int1_max | S1_Int2_min | ... |
%           S3_Intacc_mean | S3_Intacc_max
%   rows: P1
%         P2
%         P3
%         P4

FS_X = zeros(40*timesplit,accuracy*3*3);
FS_T = zeros(40*timesplit,4);

for l=0:timesplit-1
    %for each volunteer
    for i=0:n_volunteers-1
        %for each position
        for j=0:n_positions-1
            %target matrix:
            %(P1 = 1 0 0 0, P2= 0 1 0 0, P3= 0 0 1 0, P4=0 0 0 1)
            FS_T(40*l+i*4+j+1,j+1) = 1;

            %input matrix:
            %for each sensor
            for k=0:n_sensors-1
                %starting from the sampling intervals # 1
                start = fix( numel(DATA{i+1,4*j+k+1})/timesplit)*1 +1;
                pass = fix( numel(DATA{i+1,4*j+k+1})/timesplit /accuracy);
                %according to accuracy (accuracy = # intervals)
                for h=0:accuracy-1
                    %Sk_Inth_min
                    FS_X(40*l+i*4+j+1, accuracy*3*k+3*h+1) = min(DATA{i+1,4*j+k+1}(
                        start:start+pass-1));
                    %Sk_Inth_mean
                    FS_X(40*l+i*4+j+1, accuracy*3*k+3*h+2) = mean(DATA{i+1,4*j+k+1}(
                        start:start+pass-1));
                    %Sk_Inth_max
                    FS_X(40*l+i*4+j+1, accuracy*3*k+3*h+3) = max(DATA{i+1,4*j+k+1}(
                        start:start+pass-1));
                    start = start+pass;
                end
            end
        end
    end
end
end
end
end
```

## 6.5 f\_args\_splitting.m

La funzione `f_args_splitting()` si occupa del prelievo del testing set.

---

```
function [Xtrain, Ttrain, Xtest, Ttest] = f_args_splitting( XT, test_size)

%splitting TRAIN and TESTING SETS
testrows = fix( (rand(test_size,1))*numel( XT(:,1)) +1 );
num_cols = numel(XT(1,:));
XTtest = zeros(test_size, num_cols);
XTtest = XT(testrows,:);
XT(testrows,:) = [];

%returning training sets for INPUT SET and TESTING SET
Xtrain = XT(:,1:num_cols-4);
Ttrain = XT(:,num_cols-3:num_cols);

%returning testing set for INPUT SET and TESTING SET
Xtest = XTtest(:,1:num_cols-4);
Ttest = XTtest(:,num_cols-3:num_cols);
end
```

## 6.6 compute\_errors.m

La funzione `compute_errors()` definisce, parametricamente, il comportamento della function handle, `fun`, necessaria per la feature selection mediante la funzione `sequentialfs()`.

---

```
function error = compute_errors(trainingX,trainingT,testingX,testingT)
% Choose a Training Function
% For a list of all training functions type: help nntrain
% 'trainlm' is usually fastest.
% 'trainbr' takes longer but may be better for challenging problems.
% 'trainscg' uses less memory. Suitable in low memory situations.
trainFcn = 'trainscg'; % Scaled conjugate gradient backpropagation.

% Create a Pattern Recognition Network
hiddenLayerSize = 10;
net = patternnet(hiddenLayerSize);

% Setup Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 80/100;
net.divideParam.valRatio = 20/100;
net.divideParam.testRatio = 0/100;

% Train the Network
[net,tr] = train(net,trainingX,trainingT);

% Test the Network
y = net(testingX);
tind = vec2ind(testingT);
yind = vec2ind(y);
error = sum(tind ~= yind);
end
```

## 6.7 compute\_network.m

La funzione `compute_network()` si occupa della definizione e del training della Rete Neurale. [vedi 3]

---

```
function best_net = compute_network(trX, trT, tstX, tstT)

    % Choose a Training Function
    trainFcn = 'trainscg'; % Scaled conjugate gradient backpropagation.

    % Create a Pattern Recognition Network
    hiddenLayerSize = 20;
    net = patternnet(hiddenLayerSize);

    perc_error = 1;
    %training 10 times the network to find the best
    for i=1:10
        % Setup Division of Data for Training, Validation, Testing
        net.divideParam.trainRatio = 90/100;
        net.divideParam.valRatio = 10/100;
        net.divideParam.testRatio = 0/100;

        % Train the Network
        [net,tr] = train(net,trX',trT');

        % Test the Network
        y = net(tstX');
        tind = vec2ind(tstT');
        yind = vec2ind(y);
        percentError = sum(tind ~= yind)/numel(tind);

        if(percentError < perc_error)
            best_net = net;
            perc_error = percentError;
        end
    end

    %plotting best network classification over the test set
    y = best_net(tstX');

    plotconfusion(tstT',y)
end
```

## 6.8 features\_analysis.m

La funzione `features_analysis()` esegue l'analisi delle feature relative alle varie posizioni attraverso la visualizzazione di grafici ed il calcolo di indici statistici. [vedi ]

---

```
function [r, l_vals]=features_analysis(input, target, history, n_features)
%Selecting only n_features to give as input to the fuzzy system
%Find the indexes of the first n_features selected by seqfs()
fs_redux = find(history.In(n_features,:) == 1);

%Group (all) feature values according to the target position = {1,2,3,..}
pY = cell(4,1);
pX = cell(4,1);

for k=1:4
    %Values for each position for all the features
    pY{k,1} = input(find( vec2ind(target') == k),:);
    %Number of values for each position for each feature
    pX{k,1} = 1:numel(pY{k,1}(:,1));
end

[features_intervals, x_values] = compute_mf_intervals(input);

for i=1:n_features
    %Plot values distribution (histogram) for each feature
    subplot(2,n_features,i), hist(input(:,fs_redux(i))' )
    title(strcat('feature_#', num2str(i)))

    start=0;
    for j=1:4
        app = pX{j,1} + start;
        subplot(2,n_features,4+i),
        %Plot values distribution for each position for each feature
        plot( app , pY{j,1}(:,fs_redux(i))' )
        hold on
        start = start + numel(pX{j,1});
    end

    title(strcat('feature_#', num2str(i)));
    legend('pos1','pos2','pos3','pos4')

    for j=1:4
        %Plot membership functions intervals
        plot(x_values,features_intervals{i,1},'k--')
    end
end

%Computing ranges of definition for each feature
r = zeros(n_features, 2);
for i=1:n_features
    r(i,1) = min(input(:, fs_redux(i)));
    r(i,2) = max(input(:, fs_redux(i)));
end

%Computing min,max for each position for each feature
l_vals = cell(2,1);

for i=1:n_features
    for j=1:4
        l_vals{1,1}(i,j) = min( pY{j,1}(:,fs_redux(i)) );
        l_vals{2,1}(i,j) = max( pY{j,1}(:,fs_redux(i)) );
    end
end
```

## 6.9 compute\_mf\_intervals.m

La funzione `compute_mf_intervals()` è un utility di supporto alla creazione dei grafici rappresentati gli intervalli di definizione delle membership function relativi alla singola feature.

---

```
function [features_intervals, x_values] = compute_mf_intervals(input)
num_elem = numel(input(:,1));
x_values = 1:num_elem;
features_intervals = cell(4,1);
feature = zeros(17,num_elem);
    for i=1:4
        for j=1:4
            if i==1
                if j==1
                    feature(1,:) = 5000;
                elseif j==2
                    feature(2,:) = 7050;
                elseif j==3
                    feature(3,:) = 15000;
                elseif j==4
                    feature(4,:) = 23000;
                end
                features_intervals{1,1} = feature(1:4,:);
            elseif i==2
                if j==1
                    feature(5,:) = 3100;
                elseif j==2
                    feature(6,:) = 7009;
                elseif j==3
                    feature(7,:) = 11370;
                elseif j==4
                    feature(8,:) = 16040;
                end
                features_intervals{2,1} = feature(5:8,:);
            elseif i==3
                if j==1
                    feature(9,:) = 5000;
                elseif j==2
                    feature(10,:) = 8666;
                elseif j==3
                    feature(11,:) = 33800;
                elseif j==4
                    feature(12,:) = 69500;
                end
                features_intervals{3,1} = feature(9:12,:);
            else
                if j==1
                    feature(13,:) = 3000;
                elseif j==2
                    feature(14,:) = 3620;
                elseif j==3
                    feature(15,:) = 5140;
                elseif j==4
                    feature(16,:) = 12600;
                end
                feature(17,:) = 16900;
                features_intervals{4,1} = feature(13:17,:);
            end
        end
    end
end
```

## 6.10 mamdani\_recognition.m

La funzione **mamdani\_recognition()** testa il Mamdani Fuzzy System ritornando la percentuale delle posizioni/attività correttamente riconosciuta. [vedi 4.5]

---

```
function recog_percentage = mamdani_recognition(fs_inputs,fs_targets,fs_redux,
    fuzzy_mamdani)
match=0;
TRAIN_ELEMS = numel(fs_inputs(:,1));
OUTPUT_INTERVAL_LENGTH = 80;

    for i=1:TRAIN_ELEMS
        %fuzzy output
        y = evalfis(fs_inputs(i,fs_redux), fuzzy_mamdani);
        %target
        t = vec2ind(fs_targets(i,:))';
        %Converting fuzzy outputs to position (1,2,3,4)
        y = fix(1 + (y-1)/OUTPUT_INTERVAL_LENGTH);
        %counting matches
        if y==t
            match=match+1;
        end
    end
    recog_percentage = (match/TRAIN_ELEMS)*100;
end
```



## 6.11 sugeno\_recognition.m

La funzione **sugeno\_recognition()** testa il Sugeno Fuzzy System ritornando la percentuale delle posizioni/attività correttamente riconosciuta. [vedi ]

---

```
function recog_percentage = sugeno_recognition(fs_inputs,fs_targets,fs_redux,
    fuzzy_sugeno)
match=0;
TRAIN_ELEMS = numel(fs_inputs(:,1));

    for i=1:TRAIN_ELEMS
        %fuzzy output
        y = evalfis(fs_inputs(i,fs_redux), fuzzy_sugeno);
        %target
        t = vec2ind(fs_targets(i,:));
        %Rounding fuzzy outputs to position (1,2,3,4)
        y = round(y);
        %counting matches
        if y==t
            match=match+1;
        end
    end
    recog_percentage = (match/TRAIN_ELEMS)*100;
end
```