UNIVERSITÀ DI PISA

Dipiartimento di Informatica
Corso di Laurea Magistrale in Informatica

Selezione del Modello e delle Feature per l'Apprendimento in Ecologie Robotiche

Candidato:

Filippo Benedetti

Relatori:

Alessio Micheli

Davide Bacciu

Claudio Gallicchio

Anno Accademico 2013/14

Ringraziamenti

Ringraziamenti

Abstract

Questa tesi è stata svolta nell'ambito del progetto europeo RUBICON (Robotic UBIquitous COgnitive Network). RUBICON ha l'obiettivo di sviluppare ecologie robotiche adattive. A tal fine, l'ecologia robotica RUBICON sfrutta un sistema di apprendimento distribuito su reti di dispositivi eterogenei, inclusi sensori wireless. Tale sistema necessita della capacità di gestire in maniera autonoma l'acquisizione dinamica di nuovi compiti di apprendimento. La necessità di lavorare con dati sequenziali, come quelli raccolti dai sensori wireless, ha reso opportuno l'utilizzo di modelli neurali di tipo ricorrente, nello specifico le Echo State Networks (ESN). La tesi è incentrata sull'implementazione di una procedura di selezione automatica di modelli tramite la tecnica di cross-fold validation e sulla realizzazione di un algoritmo innovativo di feature selection specifico per ESN e serie temporali. I risultati ottenuti consentono ad un'ecologia robotica che adotti le soluzioni sviluppate di *auto-configurarsi* e gestire autonomamente il processo di apprendimento adattandosi ai cambiamenti nelle abitudini degli utenti. Inoltre, l'algoritmo di feature selection sviluppato nel lavoro di tesi, permette di identificare automaticamente i segnali di ingresso maggiormente rilevanti per un compito di apprendimento acquisito dinamicamente. Ciò consente di ottimizzare i costi di comunicazione e, potenzialmente, migliorare le prestazioni predittive del sistema di apprendimento.

Indice

Capitolo 1

Introduzione

Negli ultimi anni la società sta sempre maggiormente portando al centro dell'attenzione lo studio di quelle tecnologie basate sulla robotica e sull'intelligenza artificiale, atte a migliorare la qualità della vita dell'uomo nel proprio ambiente. In tali ambiti trovano spazio le ecologie robotiche del progetto europeo RUBICON [1] (Robotic UBIquitous COgnitive Network), intorno alle quali è stato svolto questo lavoro di tesi.
Le ecologie robotiche sono sistemi complessi composti da molteplici dispositivi elettronici eterogenei che cooperano per assistere gli utenti nelle loro attività quotidiane.
Nel contesto del progetto RUBICON le ecologie robotiche si applicano all'*Ambient Assisted Living* (AAL) [2], un settore che mira al miglioramento della qualità della vita per gli anziani e disabili. In tale ambito le ecologie robotiche sono utilizzate per assistere le persone nella loro vita quotidiana con lo scopo di preservarne l'autonomia. Gli ambienti dove possono essere installate le ecologie sono quindi abitazioni private, uffici ed ospedali. Mentre i principali dispositivi che costituiscono l'ecologia sono: una rete di sensori per monitorare gli utenti, e una serie di piattaforme robotiche con capacità attuative differenti a supporto delle diverse attività quotidiane nelle quali gli utenti sono coinvolti.
Nella situazione descritta le ecologie devono essere adattive, cioè devono adattarsi ai cambiamenti nell'ambiente e alle abitudini degli utenti. Sarà quindi necessario prevedere l'utilizzo di un sistema software che permetta l'adattamento dell'ecologia in modo autonomo alle diverse situazioni possibili. La soluzione a questo tipo di esigenze viene fornita dai sistemi ad apprendimento

automatico. Ovvero dei sistemi in grado di adattare i propri comportamenti in base all'esperienza (fornita al sistema come un insieme di dati). In questo caso un'esperienza di base viene fornita all'ecologia, da insiemi di dati storici raccolti durante le fasi sperimentali del sistema. Questo nella prima fase della sua esistenza, ovvero durante la sua costruzione nell'azienda produttrice. In seguito, è l'ecologia stessa a ricavare nuove esperienze da aggiungere alle precedenti sia per mezzo dei dati raccolti dai sensori (e quindi conservati), che tramite i feedback ricevuti dagli utenti. L'ecologia successivamente può *studiare* l'esperienza raccolta per adattare il proprio comportamento.
In un'ecologia robotica di questo tipo si possono distinguere due fasi concorrenti: una detta *predittiva* nella quale viene utilizzata attivamente dagli utenti, ed un'altra detta di *raffinamento* nella quale l'ecologia adatta il proprio comportamento sulla base delle nuove esperienze raccolte. Nell'ambito del progetto le informazioni che l'ecologia utilizzerà sia in fase *predittiva* che di *raffinamento* sono sequenze ordinate di dati.
Nel progetto RUBICON i dati sequenziali necessari sono raccolti tramite reti di sensori wireless [3]. Tutti questi sensori, ed anche gli altri dispositivi facenti parte dell'ecologia, sono dotati di capacità computazionali. Queste proprietà consentono l'installazione di applicazioni, ma molti sistemi di apprendimento automatico richiedono risorse hardware ben superiori a quelle dei dispositivi che RUBICON utilizza.
I sistemi ad apprendimento automatico utilizzati nel progetto RUBICON sono realizzati con modelli matematici detti reti neurali artificiali. Una rete neurale è un modello computazionale di apprendimento automatico capace di approssimare relazioni complesse (ovvero non-lineari) tra i dati in ingresso alla rete e un determinato insieme di dati obiettivo. In generale il comportamento complesso delle reti neurali emerge dall'interazione di un insieme di più semplici unità computazionali, dette neuroni (da cui l'ispirazione biologica, per maggiori dettagli vedere [4, 5]). Reti neurali particolarmente adatte all'apprendimento su dati sequenziali sono le reti neurali ricorrenti, le quali sono dotate di una memoria che gli permette di tenere una traccia storica dei dati in ingresso. Nell'ambito di RUBICON si utilizza una classe di reti neurali ricorrenti dette Echo State Network (ESN) [6, 7]. Queste reti sono particolarmente vantaggiose perché rispetto alle consuete coinvolgono algoritmi meno costosi dal punto di vista computazionale.
Nella situazione sinora illustrata c'è quindi l'esigenza di sviluppare un sistema

per l'apprendimento automatico e autonomo con le reti neurali artificiali di tipo ESN. In questo contesto di fatto è necessaria una procedura che valuti automaticamente l'efficienza di molte reti ESN che differiscono tra loro per differenti motivi, ad esempio il numero di neuroni di cui è composta. Per sviluppare un procedimento che selezioni in modo autonomo una tra le differenti ESN è importante avere una buona misura per valutare l'efficacia di queste reti neurali. Infatti un processo che si basa sui dei valori di valutazione dei modelli di ESN non può selezionare in modo autonomo e con efficacia quello più adatto per lo scopo, se i valori delle valutazioni per primi non sono sufficientemente accurati.

Al fine di avere una buona misura di valutazione si utilizza una tecnica dell'apprendimento automatico conosciuta come cross-fold validation. Questa tecnica, rispetto ad altre tecniche più semplici, permette sia di allenare più efficacemente i modelli sia di calcolare una valutazione per gli stessi più accurata.

Nell'ottica della cross-fold validation è possibile sviluppare una serie di procedure, che sfruttando i dati a disposizione, riescano in modo autonomo sia ad addestrare tutti i modelli di ESN (definiti dai parametri scelti) che a selezionare automaticamente il modello di rete ESN più accurato in termini di performance predittive.

Il progetto RUBICON si prefigge anche l'obiettivo di riuscire a installare le ecologie robotiche a costi contenuti. Un'ecologia robotica adattiva ha dei costi maggiormente sostenibili rispetto a quelle non adattive. Di fatti adattando il loro comportamento in modo autonomo alle nuove esigenze degli utenti necessitano di un minor numero di interventi da parte di tecnici specializzati. L'utilizzo per le comunicazioni tra i dispositivi di tecnologie a basso costo come il wireless, è un altro vantaggio.

Si può osservare che ogni sensore fornisce alle reti neurali dell'ecologia una sequenza di dati. Queste singole sequenze di dati sono anche chiamate caratteristiche del segnale di input o feature. Perciò l'intera sequenza in ingresso ad una rete neurale ESN è un segnale multidimensionale, dove la dimensionalità del segnale è pari al numero di sensori dai quali la rete neurale riceve l'informazione.

Quindi un minor numero di caratteristiche nel segnale di input (feature) ad una rete neurale ESN, si traduce in un numero minore di sensori (che si devono usare per supportare la raccolta delle sequenze di dati) necessari all'assistenza

degli utenti. Il vantaggio principale che offre la possibilità di utilizzare un numero ridotto di sensori è quello di ottenere modelli di ESN che usano meno informazione, con risparmi in risorse di calcolo e di comunicazione in fase *predittiva*. Per cui per fare in modo che l'ecologia sia maggiormente adattiva e che non usi l'informazione non rilevante durante le fasi *predittive*, si deve prevedere l'esistenza di un particolare processo che possa diminuire il numero di caratteristiche utilizzate nel segnale di input. Ovvero che permetta la selezione, anche questa volta in modo automatico, di un sottoinsieme feature interessanti allo scopo. Alla luce delle considerazioni fatte il lavoro della tesi è quindi incentrato sullo sviluppo di un'applicazione costituita da due componenti.
La componente primaria è un sistema in grado di selezionare autonomamente un modello di reti ESN, tra i molteplici modelli possibili definiti da una serie di attributi, tramite l'uso della tecnica di cross-fold validation. L'altra componente consiste di un algoritmo innovativo (chiamato ESNIGMA) per la selezione delle feature e specificatamente pensato per le reti ESN. Questa parte è integrata nel sistema di scelta automatica del modello, e per ognuno dei modelli definiti da una serie di attributi, la procedura seleziona un sottoinsieme di feature. Dopo di che il sistema sceglierà in modo autonomo tra tutti i modelli così modificati quello che riterrà più adatto al particolare scopo dell'ecologia robotica.
Il software oggetto della tesi fa parte dell'applicazione del progetto RUBICON ed è implementato all'interno del modulo detto Learning Layer (la componente software principale dell'intero sistema RUBICON adibita all'apprendimento automatico).
Le componenti implementate sono validate sperimentalmente su due insiemi di dati reali, nell'ambito del progetto RUBICON, che trattano la localizzazione indoor di dispositivi robotici a forma di carrello.

1.1 Stuttura della Tesi

Nei capitoli successivi si mostra il percorso della tesi e si verifica il raggiungimento degli obiettivi prefissati.
Oltre a questo capitolo questa tesi è strutturata in altri 5 capitoli.
I primi 2 capitoli introducono lo stato dell'arte e le principali tecnologie uti-

lizzate. I capitoli successivi invece illustrano le componenti sviluppate e i risultati ottenuti.
In sintesi i capitoli seguenti sono così organizzati.

- **Capitolo 2**
  Questo capitolo inizialmente introduce le ecologie robotiche e la loro applicazione nell'ambito dell'Ambient Assisted Living e del progetto RUBICON.
  Successivamente si concentra sul descrivere che cosa sono e come funzionano le reti neurali ESN. In particolar modo il capitolo è incentrato sulla descrizione del *Reservoir Computing* (RC), un paradigma nella classe delle reti neurali ricorrenti, questa è di fatto la caratteristica principale del ESN e ne definisce la struttura. Mentre la principale proprietà, anch'essa qui descritta è la Echo State Property che ne definisce invece il comportamento.
  Il capitolo si chiude poi illustrando in modo più dettagliato l'architettura di un'ecologia robotica all'interno del progetto RUBICON, e in particolar modo del modulo Lerning Layer oggetto di sviluppo della tesi.

- **Capitolo 3**
  In questo capitolo viene presentato lo stato dell'arte del problema della selezione di Feature.
  Il capitolo si apre con la discussione delle problematiche inerenti e le diverse soluzioni adottate. Segue poi una presentazione dei principali approcci utilizzati: filter e wrapper. Quindi nel contesto dei metodi wrapper (lo stesso del nuovo algoritmo oggetto della tesi) sono illustrati le diversi soluzioni.
  In conclusione il capitolo termina da prima presentando una soluzione del problema della selezione delle feature nell'ambito delle reti neurali, ed infine con un'analisi dello stesso problema nel contesto delle sequenze di dati.

- **Capitolo 4**
  Il capitolo riporta una descrizione delle principali componenti sviluppate all'interno del sistema ed il loro funzionamento. Ogni sezione del capitolo dettaglia ognuna delle diverse componenti.

Le due principali componenti sono l'oggetto della tesi, una procedura di selezione del modello di ESN e un algoritmo innovativo (ESNIGMA) per la selezione di feature nelle serie temporali, durante lo sviluppo di queste si è pensato di sviluppare altre 3 componenti. Le altre 3 componenti sviluppate sono: un procedimento per generare nuovi dati di addestramento da quelli esistenti, un sistema per controllare e modificare la correttezza delle reti ESN generate (poiché devono rispettare la Echo State Property descritta nel capitolo 2), ed una tecnica detta *comitato* che migliora i risultati in fase *predittiva* delle reti ESN.

- **Capitolo 5**
  In questo capitolo si riportano e commentano i risultati ottenuti con le componenti del sistema sviluppato su due diversi insiemi di dati.
  Il capitolo inizialmente riporta una presentazione degli scenari in cui le sequenze di dati sono state raccolte. Dopo avere quindi illustrato gli scopi degli insiemi di dati e descritto le impostazioni del sistema, il capitolo prosegue con il riportare i risultati sperimentali veri e propri.
  Lo scopo degli esperimenti e delle sezioni finali del capitolo è di verificare e dimostrare il funzionamento delle principali componenti del sistema oggetto della tesi.

- **Capitolo 6**
  In questo capitolo si riassume ed analizzano criticamente i principali contributi della tesi e quindi si introducono possibili sviluppi futuri al lavoro svolto.

Oltre ai Capitoli principali la tesi include due appendici. La prima appendice riporta il manuale d'uso dell'applicazione sviluppata, descrivendo i requisiti per il corretto funzionamento e spiegando l'output prodotto dalla stessa. La seconda appendice è invece un dettaglio dei risultati sperimentali ottenuti durante il lavoro svolto (ma che non riguarda significativamente le principali componenti sviluppate), in questa appendice è riportato anche uno dei caratteristici *report*, auto-generati a partire dall'output dell'applicazione, utilizzati in fase di analisi e di sviluppo delle componenti software oggetto della tesi.

Capitolo 2

Reservoir Computing nelle Ecologie Robotiche

Questo capitolo è incentrato sul *Reservoir Computing* (RC), un paradigma nella classe delle *Recurrent Neural Networks*(RNN), che in diversi ambiti di codifica di segnali si caratterizza, sia dal punto di vista della bassa complessità computazionale che per le prestazioni raggiunte nell'accuratezza, o anche per un buon compromesso tra i due.
In particolare, nell'ambito del paradigma RC presentiamo il modello Echo state Network (ESN), le motivazioni del progetto RUBICON [1] e le ecologie robotiche nella Sezione 2.1, nelle sezioni successive andremo ad approfondire il RC e le ESN (Sezione 2.2), il loro funzionamento ed i diversi tipi e l'architettura di un Ecologia Robotica all'interno del progetto RUBICON (Sezione 2.3.1).

2.1 Le ESN, Ambient Assisted Living, RUBICON e le Ecologie Robotiche

Le reti neurali artificiali (ANN) sono modelli matematici che rappresentano l'interconnessione tra elementi definiti come neuroni (o detti anche unità), ossia costrutti matematici che in qualche misura imitano le proprietà dei neuroni viventi [4, 5].
Le RNN sono reti neurali artificiali contraddistinte da connessioni tra neuroni dello strato precedente, dello stesso strato e sul neurone stesso, realizzando in questo modo un collegamento di retroazione, cioè ciclico [5].

Le reti neurali in generale richiedono comunque un lungo periodo di apprendimento, dovuto al fatto di dover addestrare tutte le diverse connessioni dei molteplici livelli su cui sono divise le unità.
Il *Reservoir Computing*(RC) [8] è un paradigma che consiste nella separazione concettuale in due partizioni dell'insieme dei neuroni di una RNN. Una partizione composta da sole unità non lineari e ricorrenti (detta *reservoir*), e l'altra è un insieme di unità feed-forward che calcolano l'uscita della rete (detto *readout*). L'apprendimento è ristretto al solo readout. Il particolare vantaggio nell'uso del RC sta quindi proprio nel fatto che soltanto una piccola parte della rete deve essere addestrata, apportando un significativo risparmio di tempo computazionale.
Il RC si è sviluppato e specializzato in modi e campi diversi nel corso degli anni, assumendo alcune differenziazioni e diversi nomi, come *Liquid State Machine*(LSM) [9], o Echo State Network(ESN) [6, 7].
Le ESN sono un modello di reti neurali ricorrenti (RNN). Le ESN ed in particolare una sua variante, le *Leaky Integrator Echo State Network*(LI-ESN), si sono contraddistinte con successo per la computazione e l'analisi di sequenze di segnali come il *fault detection* [10] e l'*anomaly detection* [11]. Sia per i suoi requisiti ridotti in termini di risorse hardware rispetto alle altre RNN, che per le prestazioni raggiunte, questo modello è stato scelto come buon candidato nello sviluppo di nuove tecnologie nel campo dell'*Ambient Assisted Living*(AAL) [2]. L'AAL è un settore che mira al miglioramento della qualità della vita per gli anziani o le persone disabili, assistendole nella loro vita quotidiana per preservarne l'autonomia e farli sentire sicuri nel luogo dove vivono o lavorano (tipicamente la loro casa, il loro ufficio, l'ospedale ed altri luoghi dove passano la maggior parte del loro tempo). Ovviamente, di primaria importanza in un campo simile è l'analisi di molti tipi diversi di sequenze di dati e segnali, che possono andare dal battito cardiaco, alla misura della temperatura corporea, al riconoscimento dei movimenti (tramite accelerometri), o della posizione nell'ambiente, sulla quale ci focalizzeremo. Il tracciamento e la localizzazione sia degli utenti che degli oggetti nel loro ambiente, può essere realizzato tramite un gran numero di differenti tecnologie, ma soltanto poche possono essere applicate nell'AAL. È necessaria, infatti, una soluzione che abbia dei costi ragionevoli, non invasiva per l'utente, ed accettabile dal punto di vista visivo ed emotivo di quest'ultimo [12](ad esempio una telecamera potrebbe essere mal accettata perché l'utente si sentirebbe costantemente osservato

e di conseguenza non a suo agio). Una tecnologia promettente, considerati i vincoli suddetti, è basata sull'uso dei *Wireless Sensor Networks*(WSN) [3], una determinata tipologia di rete, costituita da una architettura distribuita di dispositivi elettronici autonomi in grado di rilevare dati dall'ambiente circostante e di comunicare tra loro. Tale tecnologia è stata scelta in particolar modo per le efficienti proprietà in termini di tempo e costo di implementazione.

Ad esempio, nei problemi come la localizzazione, è stato usato un trasmettitore che invia segnali *wirless*, e uno o più sensori installati nell'ambiente che misurano la forza (ampiezza) del segnale ricevuto. La soluzione specifica può essere basta sullo scopo di calcolare il *Received Signal Strength*(RSS), in modo da valutare la posizione del trasmettitore portato dall'utente in tempo reale.

Di contro, nell'utilizzo di questa tecnologia si ha che ogni singolo dispositivo offre capacità computazionali limitate. Le ESN sono una soluzione interessante a tale problema in quanto riescono a mantenere buone prestazioni allo scalare della complessità; offrono cioè un buon trade off tra requisiti computazionali e abilità di apprendimento sulle sequenze temporali. Le reti neurali di tipo ESN sono adeguate ad essere implementate in modo distribuito, risultando adatte ad essere utilizzate contemporaneamente ad una tecnologia come le WSN [13, 14].

Il progetto RUBICON (Robotic UBIquitous COgnitive Network) [15, 16] ha l'obiettivo di integrare un apprendimento auto-sostenuto all'Ecologia Robotica in modo da fornire una soluzione economica, adattiva ed efficiente, allo scopo di applicarla nel campo dell'AAL.

La ubiquitous robotics [17] si colloca nel contesto delle reti di sensori-attuatori, con scopi di: connessione in rete di tutti i dispositivi, usabilità delle interfacce, e possibilità di servire servizi consoni alla situazione corrente.

Le Ecologie Robotiche sfruttano il concetto di ubiquitous robotics e ridefiniscono la nozione di robot, i quali diventano parte dell'ambiente stesso. In questa visione distribuita molti dispositivi specializzati (come trasmettitori, sensori, carrelli, bracci meccanici, tavoli con le ruote o persino mobili che cambiano forma e funzione) cooperano allo scopo di migliorare le condizioni degli utenti nel loro ambiente. Usualmente con ecologia si intende infatti lo studio delle relazioni tra organismi viventi e il loro ambiente, nel caso delle Ecologie Robotiche al posto degli organismi viventi abbiamo i dispositivi (sopracitati) e gli utenti.

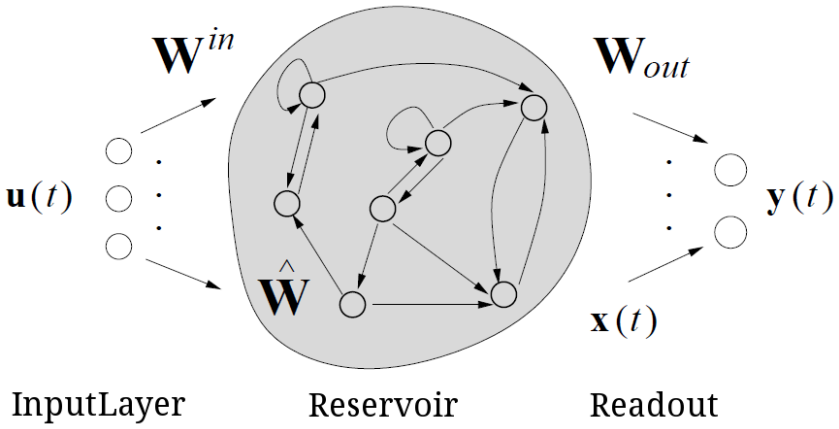
2.2 Il Reservoir Computing e le Echo State Networks

In questa sezione viene descritto il paradigma di Reservoir Computing, focalizzando l'attenzione sul modello ESN e su una sua variante, Leaky Integration-ESN (LI-ESN), particolarmente adatta a trattare sequenze di dati temporali, come i segnali wirless.

2.2.1 Architettura dell' Echo State Network

La ESN è un modello delle RNN che prevede 3 *layer*, strati distinti: un *input layer*, un *reservoir* e un *readout* con rispettivamente N_U, N_R e N_Y unità (Figura 2.1).

Figura 2.1: Architettura dell'ESN



L'input layer consiste in una serie di unità, una per ogni dimensione dell'input; nelle ESN la particolarità sta nel fatto che su questo strato non viene fatto alcun addestramento.

Il reservoir è lo strato *hidden*, nelle ESN è caratterizzato da un gran numero di unità non lineari sparsamente connesse tra loro in modo ricorrente, e con la particolarità di essere anche queste senza addestramento. Il reservoir, detto anche *dynamic reservoir*, implementa un processo di codifica di una sequenza di input in uno spazio degli stati. Questo strato

proprio tramite le connessioni ricorrenti implementa anche la memoria del sistema.

Il readout calcola l'output delle ESN (ha quindi una dimensione per ognuna delle dimensioni dell'output). È generalmente un livello feed-forward, cioè aciclico nelle connessioni, ed è l'unico componente addestrato dell'intera architettura.

L'approccio delle ESN [6, 7] è basato sull'osservazione che se un reservoir rispetta una determinata proprietà, detta *Echo State Property* (ESP) [6] il training limitato ad un readout di sole unità lineari, è sufficiente ad ottenere buone prestazioni in molteplici ambiti riguardanti le sequenze di segnali.
Formalmente, data una sequenza di input $\mathbf{s} = [\mathbf{u}(1), \mathbf{u}(2), ..., \mathbf{u}(n)]$ nello spazio di input $\mathbb{R}^{N_U}$ ad ogni passo $t = 1, ..., n$ il reservoir calcola la seguente funzione:

$$\mathbf{x}(t) = f(\mathbf{W}^{in}\mathbf{u}(t) + \hat{\mathbf{W}}\mathbf{x}(t-1)) \tag{2.1}$$

dove

- $\mathbf{x}(t) \in \mathbb{R}^{N_R}$ è lo stato del reservoir, ovvero l'output del hidden layer al tempo t,
- $\mathbf{W}^{in} \in \mathbb{R}^{N_R \times N_U}$ è la matrice dei pesi di input al reservoir,
- $\hat{\mathbf{W}} \in \mathbb{R}^{N_R \times N_R}$ è la matrice dei pesi ricorrenti del reservoir,
- f è la funzione di attivazione dei neuroni del reservoir, tipicamente si utilizza $f = tanh$.

Per $t = 0$ si utilizza uno stato nullo $\mathbf{x}(0) = 0 \in \mathbb{R}^{N_R}$. Come funzione dell'output del readout, che si applica soltanto alla fine del processo di codifica calcolato dal reservoir, generalmente si ha l'equazione:

$$\mathbf{y}(\mathbf{s}) = f_{out}(\mathbf{W}_{out}[\mathbf{x}(n)|\mathbf{u}(n)]) \tag{2.2}$$

dove

- $\mathbf{W}_{out} \in \mathbb{R}^{N_Y \times (N_R + N_U)}$ è la matrice dei pesi del readout e opzionalmente dall'input all'output,

- f_{out} è la funzione di output che dipende dallo scopo; ovvero dal tipo di task computazionale *regressione* (funzione lineare) o *classificazione* (funzione segno).
- $\mathbf{x}(n)$ è lo stato interno della rete dopo la codifica dell'intera sequenza s, dove se l'input è collegato direttamente all'output l'n-esimo input $\mathbf{u}(n)$ è concatenato nella formula allo stato interno $\mathbf{x}(n)$ (nei casi successivi l'input non sarà collegato direttamente all'output).
- $\mathbf{y}(\mathbf{s})$ è l'output di classificazione calcolato per la sequenza di input $\mathbf{s}$.

L'output si può, come finora illustrato, calcolare alla fine di una sequenza di dati oppure per ogni passo della sequenza.
La (2.1) può anche essere modificata in modo da introdurre un feedback dell'output:

$$\mathbf{x}(t) = f(\mathbf{W}^{in}\mathbf{u}(t) + \hat{\mathbf{W}}\mathbf{x}(t-1) + \mathbf{W}_{ofb}\mathbf{y}(t-1)) \tag{2.3}$$

dove rispetto alla (2.1) $\mathbf{W}_{ofb} \in \mathbb{R}^{N_R \times N_Y}$ è la matrice dei pesi del feedback dell'output, e $\mathbf{y}(t-1)$ è il risultato della (2.2) calcolato al passo di tempo precedente. In questo caso la funzione dell'output del readout viene calcolata per ogni istante della sequenza di input e non soltanto alla fine di quest'ultima.
Come descritto il reservoir deve rispettare la ESP, ovvero, asintoticamente lo stato interno della rete deve dipendere soltanto dal segnale di input e non dalle condizioni iniziali. Formalmente, sia $\mathbf{s}_n = [\mathbf{u}(1)...\mathbf{u}(n)] \in (\mathbb{R}^{N_U})^n$ una sequenza di input di lunghezza n, e sia la versione ricorsiva della funzione di transizione dello stato interno $\tau : (\mathbb{R}^{N_U})^* \times \mathbb{R}^{N_R} \to \mathbb{R}^{N_R}$ dove:

$$\begin{aligned}
&\forall \mathbf{s}(\mathbf{u}) \in (\mathbb{R}^{N_U})^*, \forall \mathbf{x} \in \mathbb{R}^{N_R} \\
&\qquad \tau(\mathbf{s}(\mathbf{u}), \mathbf{x}) = \mathbf{x} \qquad se \qquad \mathbf{s}(\mathbf{u}) = [] \\
&\qquad\qquad\qquad e \\
&\qquad \tau(\mathbf{s}(\mathbf{u}), \mathbf{x}) = \tau([\mathbf{u}(n)], \tau([\mathbf{u}(1)...\mathbf{u}(n-1)], \mathbf{x})) \\
&\qquad\qquad\qquad\qquad se \qquad \mathbf{s}(\mathbf{u}) = [\mathbf{u}(1)...\mathbf{u}(n)]
\end{aligned}$$

presi $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^{N_R}$ due diversi stati interni iniziali allora la ESP vale se $\forall \mathbf{s}_n \in (\mathbb{R}^{N_U})^n$ e $\forall \mathbf{x}, \mathbf{x}' \in \mathbb{R}^{R_N}$ vale che:

$$||\tau(\mathbf{s}_n, \mathbf{x}) - \tau(\mathbf{s}_n, \mathbf{x}')|| \to 0 \qquad per \qquad n \to \infty \tag{2.4}$$

È stato dimostrato [6] che per ogni input e per ogni funzione dello stato interno condizione sufficiente affinché valga la ESP, è che $\sigma_{max}(\hat{\mathbf{W}}) < 1$, dove $\sigma_{max}(\hat{\mathbf{W}})$ è il più grande valore singolare di $\hat{\mathbf{W}}$. Quindi, poiché il massimo valore singolare di una matrice quadrata è uguale alla sua norma 2, anche la condizione

$$||\hat{\mathbf{W}}||_2 < 1 \tag{2.5}$$

è condizione sufficiente per la ESP. Inoltre, questa condizione 2.5 implica la proprietà di *contrattività* per la rete [18]:

$$\exists C \in \mathbb{R}, o \leq C < 1, \forall \mathbf{u} \in (\mathbb{R}^{N_U})^1, \forall \mathbf{x}, \mathbf{x}' \in \mathbb{R}^{N_R}$$
$$||\tau(\mathbf{u}, \mathbf{x}) - \tau(\mathbf{u}, \mathbf{x}')|| \leq C||\mathbf{x} - \mathbf{x}'||$$

e se tale proprietà vale allora vale anche la ESP, e la contrattività delle dinamiche di stato del reservoir garantisce anche la sua *markovianità*. La markovianità è una proprietà secondo la quale a sequenze che condividono lo stesso suffisso corrispondono output simili, proporzionalmente alla lunghezza del suffisso in comune. Nell'ESN sequenze di input che condividono un suffisso comune, sono mappate nello spazio degli stati quindi tanto più vicine quanto più è lungo tale suffisso.
Sia $\rho(\hat{\mathbf{W}})$ il raggio spettrale della matrice $\hat{\mathbf{W}}$, soddisfare le condizioni sufficienti, ha come risultato $\rho(\hat{\mathbf{W}})$ molto inferiori ad 1, ovvero pesi di $\hat{\mathbf{W}}$ molto piccoli, che portano a dare troppa poca importanza agli istanti di tempo precedenti, cioè a non avere abbastanza memoria del passato. Per questo motivo, la condizione sufficiente è troppo restrittiva, per cui non si usa e si preferisce soddisfare la condizione necessaria: $\rho(\hat{\mathbf{W}}) < 1$. Nella pratica si utilizzano valori di ρ molto vicini ad uno perché spesso le ESN così configurate restituiscono i migliori risultati predittivi (e la ESP vale comunque) [19]. Tipicamente, un ESN viene inizializzata nel modo seguente:

1. Una procedura genera dei valori casuali da una distribuzione uniforme per le matrici $\mathbf{W}^{in}$ e $\hat{\mathbf{W}}$.

2. Sia $\rho_{desired}$ il valore che si vuole che ρ abbia, i valori di $\hat{\mathbf{W}}$ sono scalati in modo da soddisfare le condizioni necessarie per la ESP $\hat{\mathbf{W}}_{new} = \hat{\mathbf{W}} \frac{\rho_{desired}}{\rho(\hat{\mathbf{W}})}$ in tal modo $\rho(\hat{\mathbf{W}}_{new}) = \rho_{desired}$.

3. Per ogni sequenza, si esegue una fase iniziale detta *transiente*. Il transiente consiste, in generale, nello scartare il risultato dalla codifica di un numero sufficiente di passi della sequenza di input, in modo da rendere il reservoir indipendente dallo stato iniziale.

4. Si addestra il readout risolvendo il problema lineare dei minimi quadrati

$$min||\mathbf{W}_{out}\mathbf{X} - \mathbf{Z}_{target}||_2^2 \tag{2.6}$$

dove, siano N le collezioni delle sequenze di training, $\mathbf{X} = [\mathbf{x}(1)...\mathbf{x}(N)]$ e $\mathbf{Z}_{target} = [\mathbf{z}(1)...\mathbf{z}(N)]$, con $\mathbf{z}(t)$ il valore di addestramento di riferimento per l'output calcolato all'istante di tempo t.

Un metodo spesso utilizzato per risolvere questo problema è fare uso della pseudo inversa di Moore-Penrose $\mathbf{X}^+$ di $\mathbf{X}$:

$$\mathbf{W}_{out} = \mathbf{Z}_{target}\mathbf{X}^+ \tag{2.7}$$

ma poiché $\mathbf{X} \in \mathbb{R}^{N_R \times N}$, la soluzione dipende direttamente dal numero N di esempi di training per cui si preferisce utilizzare il metodo delle equazioni normali:

$$\mathbf{W}_{out}\mathbf{X}\mathbf{X}^T = \mathbf{Z}_{target}\mathbf{X}^T \Rightarrow \mathbf{W}_{out} = \mathbf{Z}_{target}\mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1} \tag{2.8}$$

con $\mathbf{Z}_{target}\mathbf{X}^T \in \mathbb{R}^{N_Y \times N_R}$ e $(\mathbf{X}\mathbf{X}^T)^{-1} \in \mathbb{R}^{N_R \times N_R}$, in tal modo la complessità della risoluzione non dipende più dalla quantità N degli esempi di training e valori intermedi di $\mathbf{W}_{out}$ possono essere calcolati durante l'esecuzione. In aggiunta, questo metodo consente l'introduzione della regolarizzazione di Tikhonov:

$$\mathbf{W}_{out} = \mathbf{Z}_{target}\mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda^2 I)^{-1} \tag{2.9}$$

dove $I \in \mathbb{R}^{N_R \times N_R}$ è la matrice identica e λ è un fattore di regolarizzazione. Numericamente questi ultimi metodi sono meno stabili rispetto al primo (2.7), ma questo, può essere mitigato utilizzando anche in questi ultimi la pseudo inversa calcolando rispettivamente $(\mathbf{X}\mathbf{X}^T)^+$ e $(\mathbf{X}\mathbf{X}^T + \lambda^2 I)^+$.
Uno dei principali vantaggi nell'uso dell'ESNs è l'efficienza. Come già spiegato il training è ristretto alla sola parte dell'output lineare, ed inoltre la parte dinamica della rete è fissa e il costo della procedura di codifica scala linearmente con la lunghezza dell'input, sia per il training che per il test.

2.2.2 Leaky Integrator ESN

Oltre alle classiche unità sigmoidali è possibile utilizzare neuroni detti *Leaky Integrator* [6, 20]. In tal caso la ESN prende il nome di Leaky Integrator ESN (LI-ESN)[20]. Questo tipo di neuroni eseguono un integrazione parziale della loro funzione di attivazione con il loro output al passo precedente. Esistono molti metodi per realizzare questo concetto tra i più utilizzati troviamo il seguente.

$$\mathbf{x}(t) = (1 - a\delta t)\mathbf{x}(t-1) + \delta t f(\mathbf{W}^{in}\mathbf{u}(t) + \hat{\mathbf{W}}\mathbf{x}(t-1)) \tag{2.10}$$

con δt il gap temporale tra 2 passi consecutivi di una sequenza diviso per l'unità di tempo utilizzata e a è il *leakage rate* [21], in tal caso $a\delta t \in [0, 1]$. Oppure un'altra formula (spesso preferibile per semplicità) si ottiene settando $a = 1$ e ridefinendo δt come il *leakage rate* a.

$$\mathbf{x}(t) = (1 - a)\mathbf{x}(t-1) + af(\mathbf{W}^{in}\mathbf{u}(t) + \hat{\mathbf{W}}\mathbf{x}(t-1)) \tag{2.11}$$

dove $a \in [0, 1]$. Si noti che se $a = 1$ in (2.11) allora la funzione di transizione di stato 2.11 $\mathbf{x}(t)$ è uguale a quello in (2.1) delle ESNs. Questi parametri addizionali regolano la velocità delle dinamiche del reservoir. Valori più piccoli o più grandi di a e δt fanno in modo che il reservoir agisca rispettivamente più lentamente o più rapidamente all'input. L'effettivo vantaggio nell'utilizzo della LI-ESN, rispetto all'ESN, sta nel poter regolare tramite i parametri su detti, l'effettivo intervallo di frequenze nel quale il reservoir opera, in modo da permettere alla rete di gestire meglio segnali di input che cambiano lentamente rispetto alla frequenza di campionamento [22, 23, 24].
Sia $\tilde{\mathbf{W}} = (1 - a)I + a\hat{\mathbf{W}}$, le condizioni necessarie sono soddisfatte se [19]

$$\rho(\tilde{\mathbf{W}}) < 1 \tag{2.12}$$

dove $\rho(\tilde{\mathbf{W}})$ è il raggio spettrale di $\tilde{\mathbf{W}}$. La matrice $\hat{\mathbf{W}}$ dopo essere stata inizializzata casualmente da una distribuzione uniforme (come $\mathbf{W}^{in}$), viene ri-scalata in modo che valga la (2.12), e quindi:

$$\hat{\mathbf{W}}_{new} = \frac{1}{a}(\tilde{\mathbf{W}}\frac{\rho_{desired}}{\rho(\tilde{\mathbf{W}})} - (1 - a)I) \tag{2.13}$$

2.3 L'Ecologia Robotica in RUBICON

Un Ecologia Robotica come introdotto nella Sezione 2.1 è un concetto che si basa sulla nozione di ubiquitous robotics [17]. L'idea è quella di integrare le conoscenze della robotica e della sensoristica per creare ambienti nei quali i servizi non sono forniti tramite pochi dispositivi estremamente complessi ma attraverso molteplici componenti robotiche che cooperano tra loro. I nodi (i dispositivi integrati nell'ambiente) che fanno parte della rete possono avere capcità computazionali eterogenee. Le interfacce tra i componenti e con gli utenti sono composte da sensori, attuatori e dispositivi mobili.
In RUBICON La particolarità consiste nell'obiettivo di creare un ecologia robotica ad apprendimento *auto-sostenuto*.

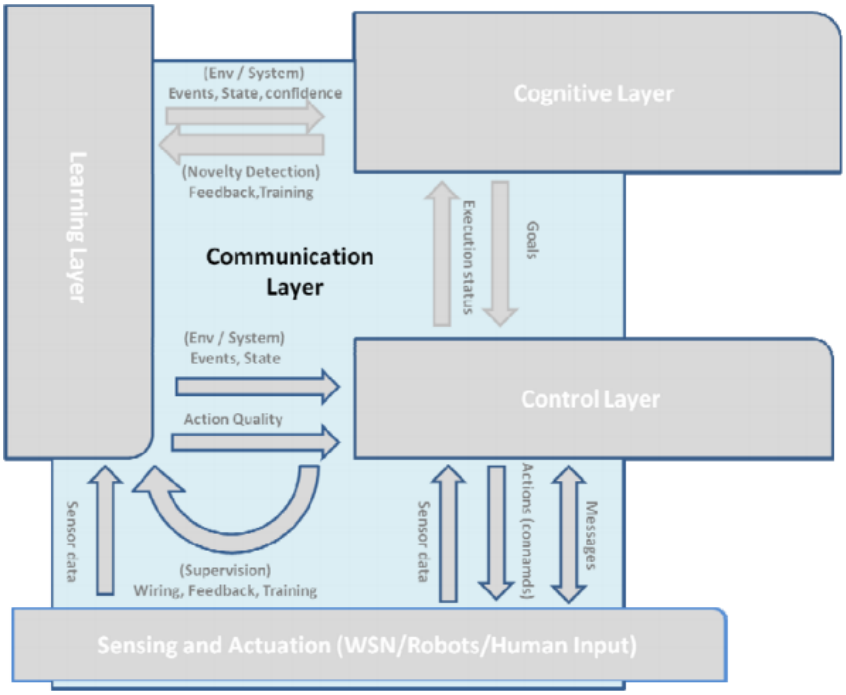
2.3.1 Architettura dell'Ecologia Robotica

Questa sezione descrive l'architettura dell'ecologia robotica nell'ambito del progetto RUBICON [25]. Come detto in precedenza l'obiettivo è realizzare un apprendimento auto-sostenuto. Questo si ottiene soddisfacendo i seguenti principi:

- I componenti dell'ecologia cooperano al fine di migliorare le prestazioni usando l'esperienza passata, definendo così una memoria dell'ecologia condivisa e distribuita.
- Tollerare la rimozione di un componente dall'ecologia.
- Un nuovo componente aggiunto all'ecologia deve poter beneficiare della conoscenza catturata nella memoria dell'ecologia.

Al fine di raggiungere lo scopo si è reso necessario l'incontro di diversi ambiti di ricerca quali il Wireless Sensor Network(WSNs), la *cognitive robotics*, l'*agent control system* e il *machine learning*. La Figura 2.2 riassume l'architettura del sistema RUBICON composta da quattro *Layer*: il *Communication* layer,

Figura 2.2: Panoramica del sistema RUBICON



il *Learning* Layer, il *Cognitive* Layer e il *Control* Layer. Ognuno di questi realizza un aspetto dell'ecologia e si occupa di realizzare specifici scopi:

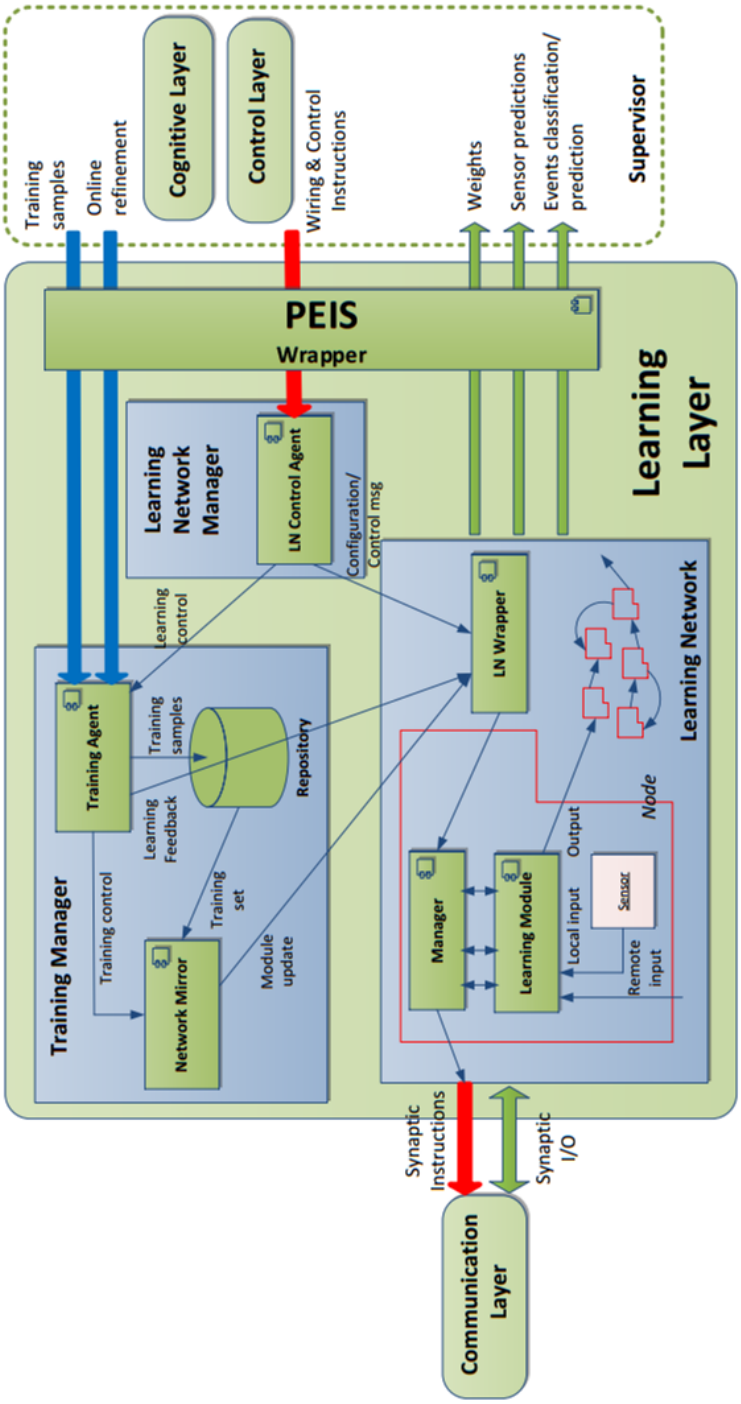
Communication layer: il quale implementa tutti i protocolli di comunicazione tra tutti gli altri layer, come al livello di wireless sensor network (basato su tecnologia PEIS [26, 27]).

Control Layer: un sistema multi-agente che modella ogni nodo dell'ecologia come un agente autonomo pianificando, configurando e controllando l'ecologia e i suoi dispositivi.

Cognitive Layer: fornisce funzionalità di apprendimento per identificare nuove situazioni di alto livello e a lungo termine nell'ambiente. Ad esempio come riconoscere attività insolite da parte degli utenti, in modo da inferire dei nuovi obiettivi per l'ecologia.

Learning Layer: la parte principale del sistema adibita all'apprendimento automatico a breve termine. Lo scopo di questo livello è riconoscere eventi e fornire predizioni basandosi su sequenze temporali di segnali in input.

Figura 2.3: Architettura software del *Learning Layer*. I sottosistemi logici sono rappresentati da rettangoli che contengono i componenti software. I componenti software sono rappresentati da rettangoli più piccoli che contenenti nell'angolo in alto a destra l'icona di componente UML. Le interfacce sono denotate con frecce fini e spesse.



2.3.2 Architettura del Learning Layer

Il livello nel quale sono state sviluppate le funzionalità oggetto di questa tesi è il *Learning Layer*.
Il Learning Layer è un complesso sistema software composto da molteplici componenti implementati in diversi linguaggi in base all'hardware e al SO supportati. L'architettura del Learning Layer è mostrata in figura 2.3. Il Learning Layer è organizzato in 3 distinti sottosistemi logici, rappresentati in figura 2.3 dai rettangoli blu chiaro. Ognuno dei sottositemi è composto da diversi moduli software, distribuiti sui nodi eterogenei che compongono l'architettura, comprendente sia piccoli dispositivi con risorse di calcolo e memoria limitate (come i sensori della Wireless Sensor Network), sia dispositivi con risorse computazionali più potenti.
I sottosistemi del Learning Layer sono i seguenti:

Learning Network (LN) realizza la memoria dell'ecologia per mezzo di ESN distribuite tra i differenti dispositivi eterogenei (rappresentati con poligoni a forma di L nella figura 2.3). Ogni dispositivo contiene un **Learning Module**, che implementa una ESN che è configurata dal componente **Manager**.

Learning Network Manager (LNM) è utilizzato per la configurazione e il controllo del Learning Layer. Questo sottosistema è implementato da un singolo componente software, il **LN Control Agent**, implementato come una agente *Java* su un dispositivo di gateway, in generale un PC.

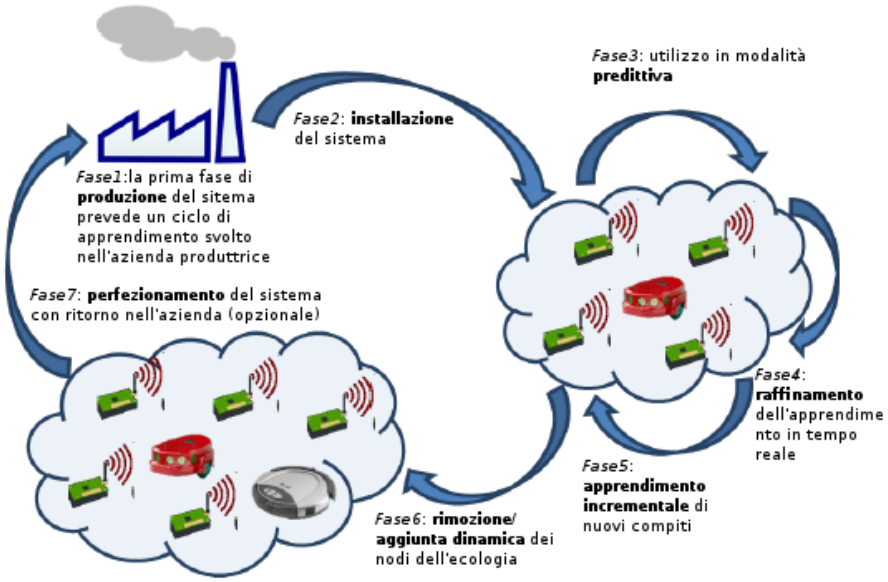
Training Manager (TM) controlla la fase di apprendimento del Learning Layer, riceve i dati dal supervisore e li usa per aggiornare i moduli di learning del sottosistema LN. Le funzionalità del TM sono implementate su due componenti software java, il **Training Agent** e il **Network Mirror**. Il **Repository** è usato per salvare e conservare i dati di training.

2.3.3 Il Ciclo di un Ecologia Robotica Adattiva

Il ciclo per la messa appunto di ecologia robotica adattiva, attraversa diverse fasi riassunte nella Figura 2.4.

La fase 1 è la **produzione**, che verte nella costruzione dei componenti dell'ecologia completata da un'iniziale fase di apprendimento.
Il passo successivo, fase 2, consiste nell'uscita del prodotto dalla fabbrica e nell'installazione dell'ecologia nell'ambiente degli utenti. La fase 3 consiste nella ecologia in fase predittiva, in cui l'ecologia viene usata dagli utenti ed elabora e raccoglie continuamente nuovi dati. Durante le fasi 4 e 5 l'ecologia utilizza i dati raccolti in fase predittiva rispettivamente per: raffinare le predizioni in base allo specifico ambiente e utenti con i quali interagisce, e tentare di apprendere nuovi compiti dove necessario.

Figura 2.4: Fasi di un ecologia robotica adattiva



La fase 6 consiste nel supportare sia l'aggiunta di nuovi dispositivi all'ecologia che la rimozione di altri dalla stessa.
La fase 7, che prevede il ritorno alla fase di produzione, si basa sull'utilizzo di tutti i dati raccolti nelle precedenti fasi con lo scopo di raffinare ulteriormente l'intero sistema correggendone gli eventuali difetti o mancanze ed eventualmente migliorandone le tecniche di apprendimento. L'adattività, ad esclusione della fase 2, è chiaramente parte integrante durante tutto il ciclo dell'ecologia. In particolare nelle fasi 1, 4, 5, e 7, l'adattività è realizzata tramite l'apprendimento confermandone l'assoluta importanza durante l'intero ciclo *vitale* dell'ecologia, e non solo in fase di ideazione o produzione.

Capitolo 3

La Feature Selection e le Sequenze di Dati

In questo capitolo si presenta una panoramica sul problema della Feature Selection ed i principali approcci alla sua risoluzione. Nella Sezione 3.1 sono discusse le problematiche e le diverse soluzioni studiate, si analizza quindi il problema della selezione di feature chiarendo la definizione di feature rilevanti. Nella Sezione 3.2 si descrivono gli approcci filter e wrapper ed i loro vantaggi e svantaggi. Nel contesto dei metodi detti wrapper, in Sezione 3.3, si illustrano le principali tecniche utilizzate confrontando i diversi approcci, le relative euristiche, e discutendone i vantaggi. Nel contesto delle reti neurali, in Sezione 3.3.1, si presenta un'euristica legata al modello di apprendimento detta ANNIGMA. Nella Sezione 3.4 si analizza il problema della feature selection nell'ambito delle sequenze.

3.1 La Feature Selection

Tom M. Mitchell ha fornito una definizione di apprendimento che include qualsiasi programma per computer che migliora le sue prestazioni per un certo compito attraverso l'esperienza.

> Un programma apprende da una certa esperienza E se: nel rispetto di una classe di compiti T, con una misura di prestazione P, la prestazione P misurata nello svolgere il compito T è migliorata dall'esperienza E [4].

Dove l'esperienza E è fornita da un insieme di dati.
Un quesito che emerge nei problemi di apprendimento automatico è comprendere su quali attributi dei dati focalizzare l'attenzione. Tali sistemi di apprendimento, devono infatti evincere dai dati stessi quali feature siano rilevanti per il problema trattato.
Gli insiemi di dati che si utilizzano possono avere un gran numero di attributi, il problema di ridurne la dimensionalità risulta perciò importante e si presta a diverse applicazioni.
Ad un problema di minore dimensionalità, in particolare, corrisponde una complessità minore in termini di tempi di risoluzione. É anche possibile che eliminando dell'informazione non rilevante nell'input si possa ottenere una misura di prestazione migliore nello svolgimento dei compiti del programma.
Uno dei metodi più comuni e utilizzati per ridurre la dimensionalità consiste nel selezionare un sottoinsieme degli attributi, ovvero fare *Feature Selection*.
Ci sono tre approcci standard alla Feature Selection[28]: *embedded*, *filter*, e *wrapper*.

Embedded La selezione è integrata nell'algoritmo di apprendimento. Nello specifico, l'algoritmo stesso decide quali attributi usare e quali ignorare.

Filter La selezione delle feature viene fatta prima dell'esecuzione dell'algoritmo di apprendimento usando dei metodi da esso indipendenti.

Wrapper Per selezionare un sottoinsieme di attributi questi metodi utilizzano la risposta dell'algoritmo di apprendimento e la misura di prestazione, tipicamente senza enumerare tutti i possibili sottoinsiemi.

Concettualmente, la selezione di un sottoinsieme di feature, in particolar modo con l'approccio a filter e a wrapper, consiste in una ricerca tra tutti i possibili sottoinsiemi dell'insieme degli attributi del data-set. La selezione di feature può quindi essere vista come un processo determinato da quattro elementi: una misura per valutare un sottoinsieme, una strategia di ricerca per controllare la generazione dei sottoinsiemi, un criterio di fermata e una procedura di validazione[28].

3.1.1 Il Problema del Subset Feature Selection

Il problema della selezione del sottoinsieme di feature, nell'apprendimento automatico, è quello di individuare un sottoinsieme dall'originario insieme di feature, tale che l'algoritmo di apprendimento eseguito su quel sottoinsieme restituisca una funzione con la massima precisione possibile. In pratica il risultato della misura di prestazione sul risultato dell'algoritmo di apprendimento può essere fatto coincidere con la misura sia per valutare che per validare un sottoinsieme selezionato.

Definizione 1 Dato un algoritmo di apprendimento A, e un data-set D con feature $X_1, X_2, ..., X_n$, da una distribuzione D sullo spazio delle etichette, un *sottoinsieme ottimale di feature*, X_{opt}, è un sottoinsieme tale che l'accuratezza della funzione $F = A(D)$ risultato dell'algoritmo di apprendimento è massimale [29].

Il principale problema di ordine pratico nell'utilizzo di questa definizione è che si deve stimare l'accuratezza della funzione risultato dell'algoritmo di apprendimento dai dati.
Un argomento importante è la relazione tra le feature ottimali e le feature rilevanti. Almuallim and Dietterich[30] definiscono la rilevanza di una feature sotto l'assunzione che tutte le feature e i target siano booleani e che non ci sia rumore.

Definizione 2 Una feature X_i si dice rilevante ad un concetto C se X_i appare in ogni formula booleana che rappresenta C, altrimenti è detta irrilevante.

Gennari [31] considera il rumore e le feature con molteplici valori e definisce le feature rilevanti come quelle i cui "valori variano sistematicamente con la categoria di appartenenza".

Definizione 3 X_i è rilevante se e solo se esistono degli x_i e y per i quali $p(X_i = x_i) > 0$ tale che

$$p(Y = y|X_i = x_i) \neq p(Y = y). \tag{3.1}$$

Secondo questa definizione X_i è rilevante se Y è condizionalmente dipendente da X. Si può notare che questa definizione fallisce nel catturare la rilevanza

nel concetto di parità dove tutte le istanze sono equiprobabili [29]. Sia $S_i = X_1, ..., X_{i-1}, X_{i+1}, ..., X_m$, l'insieme di tutte le feature eccetto X_i. Si denota con s_i l'assegnamento di un valore a tutte le feature in S_i.

Definizione 4 X_i è rilevante se e solo se esistono x_i, y, e s_i per i quali $p(Xi = xi) > 0$ tale che

$$p(Y = y, S_i = s_i | X_i = x_i) \neq p(Y = y, S_i = s_i). \tag{3.2}$$

Seguendo questa definizione, X_i è rilevante se la probabilità di un istanza (date tutte le feature) può cambiare quando si elimina la conoscenza del valore di X_i [29].

Definizione 5(*Rilevanza forte*) X_i è rilevante se e solo se esistono x_i, y, e s_i per i quali $p(Xi = xi, Si = si) > 0$ tale che

$$p(Y = y | X_i = x_i, S_i = s_i) \neq p(Y = y | S_i = s_i). \tag{3.3}$$

La rilevanza dovrebbe essere definita in termini di un classificatore *Bayesiano* ottimale per quel dato problema. Una feature X è *fortemente rilevante* se la sua esclusiva rimozione risulta in un deterioramento delle prestazioni del classificatore Bayesiano. Una feature X è *debolmente rilevante* se non è fortemente rilevante e se esiste un sottoinsieme di feature, S, tale che le prestazioni del classificatore su S sono peggiori delle prestazioni su $S \cup \{X\}$[29].

Definizione 6(*debolmente rilevante*). Una feature X_i è *debolmente rilevante* se e solo se non è fortemente rilevante e se esiste un sottoinsieme di feature S'_i di S_i per il quale esistono x_i, y, e s'_i con $p(Xi = xi, S'_i = s'_i) > 0$ tale che

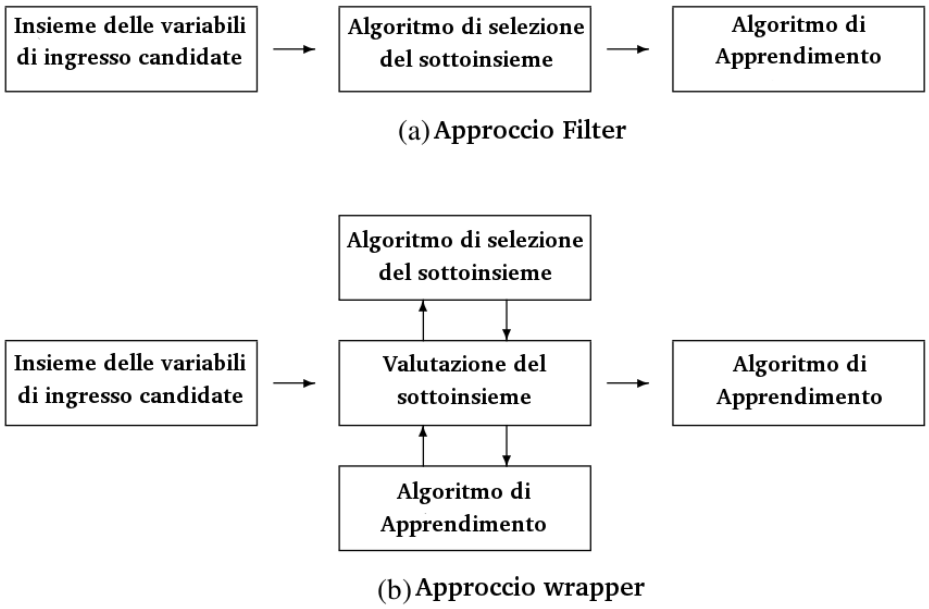
$$p(Y = y | X_i = x_i, S'_i = s'_i) \neq p(Y = y | S'_i = s'_i). \tag{3.4}$$

Una feature è *rilevante*, quindi, se è debolmente o fortemente rilevante, è irrilevante altrimenti[29]. È possibile mostrare sia che la rilevanza non implica l'ottimalità, sia che l'ottimalità non implica la rilevanza. Tuttavia, nella pratica i casi in cui una feature appartenga al sottoinsieme ottimo e contemporaneamente non sia rilevante sono estremamente rari e quindi le feature irrilevanti possono generalmente essere rimosse.

3.2 Approcci Filter e Wrapper

I metodi wrapper implementano una ricerca nello spazio delle feature guidata dall'algoritmo di apprendimento. In generale l'accuratezza del modello risultato dell'algoritmo di apprendimento è calcolata ogni volta che si aggiunge o si elimina una feature dall'insieme. In base all'algoritmo di ricerca utilizzato si prosegue selezionando ad ogni passo una feature da aggiungere o eliminare finché non siano soddisfatte delle condizioni di fermata.D'altro canto i metodi filter selezionano un sottoinsieme di feature in modo del tutto indipendente dall'algoritmo di apprendimento e il biases di quest'ultimo non interagisce con quello del filter.

Figura 3.1: Schemi dei metodi wrapper e filter



Differenti algoritmi di apprendimento però, possono avere prestazioni migliori con differenti sottoinsiemi di feature anche se utilizzano lo stesso training set (come mostrano gli esperimenti sul CorrAL data-set [32]). L'uso di un adeguato algoritmo di apprendimento dunque, emerge come fattore cruciale per determinare il successo dei metodi wrapper.
È da considerare, in particolar modo, che i metodi wrapper sono computazionalmente molto onerosi rispetto ai filter su grandi insiemi di dati. Si può notare, inoltre, che l'accuratezza della cross-validation ha un'alta varianza su

piccoli insiemi di training e quindi su questi insiemi i metodi wrapper tendono ad incorrere in overfitting [33].
Il vantaggio primario nei metodi filter è la complessità computazionale e la buona scalabilità sui grandi insiemi di dati. Inoltre il sottoinsieme di feature selezionate è spesso migliore nei casi in cui i wrapper possono andare in overfitting come su insiemi di dati molto piccoli.
Il dibattito sull'utilizzo dei metodi wrapper o filter è sempre aperto: i metodi wrapper ottengono prestazioni predittive generalmente migliori in particolar modo quando il sottoinsieme ottimale di feature è legato ad uno specifico algoritmo di apprendimento. Ad ogni modo l'eccessivo costo computazionale non sempre ne giustifica l'utilizzo, considerando che l'accuratezza spesso non è significativamente migliore; inoltre, insiemi di feature che hanno buone prestazioni per un algoritmo di apprendimento avranno prestazioni simili per differenti algoritmi[34].
Alcuni studi[34, 35], hanno esplorato anche la possibilità di sviluppare e utilizzare algoritmi ibridi che cercano di sfruttare i punti di forza sia degli approcci a wrapper che a filter. In diversi casi, quest'ultimi metodi, migliorano sensibilmente le prestazioni dell'algoritmo di apprendimento, risultando competitivi con i metodi wrapper, mentre selezionano il sottoinsieme di feature molto più rapidamente.

3.3 Wrapper per la Ricerca nello Spazio delle Feature

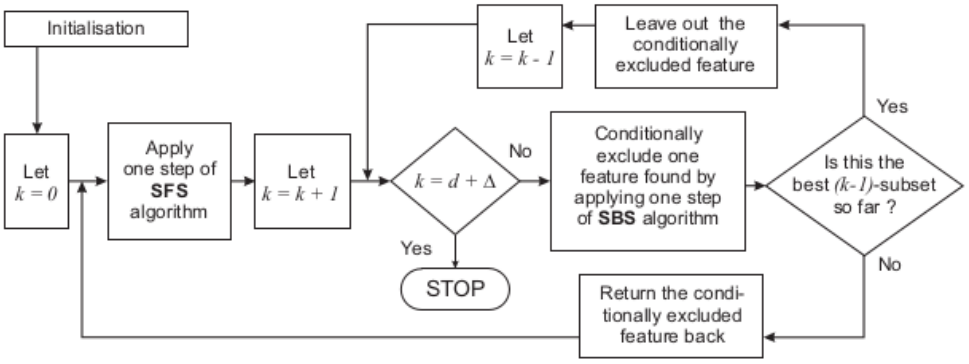
Nei metodi wrapper, storicamente, si possono distinguere 2 principali approcci per la ricerca nello spazio delle feature. La distinzione è fatta in base alla direzione di ricerca: *forward sequential selection* (FSS o SFS) e *backward sequential selection* (BSS o SBS) [36].
I metodi FSS iniziano la ricerca con l'insieme formato da zero feature e valutano tutti i sottoinsiemi con esattamente una feature, selezionando quello con le migliori prestazioni. Si prosegue con l'aggiungere all'attuale sottoinsieme un'altra feature che permetta di ottenere prestazioni migliori secondo il criterio di valutazione utilizzato. Quest'ultima procedura viene quindi ripetuta fin quando non si soddisfano i criteri di arresto.
I metodi BSS invece, iniziano la ricerca considerando l'intero insieme di fea-

ture, rimuovendo una feature ricorsivamente fintanto che la valutazione delle prestazioni non soddisfi i criteri di arresto. I BSS spesso superano in prestazioni gli FSS [37], probabilmente perché i BSS valutano il contributo di una data feature nel contesto di tutte le altre, mentre gli FSS al contrario, possono valutare l'utilità di una feature soltanto in un contesto limitato alle feature già precedentemente selezionate. Sebbene tale problema con gli FSS sia stato riscontrato, è da considerare che esistono diversi casi in cui i BSS non superano gli FSS [38]. Gli FSS sembrano avere soluzioni migliori, in termini di accuratezza del sotto-insieme selezionato, quando l'insieme ottimale di feature ne richiede molte. In caso contrario sono i BSS a dare risposte più soddisfacenti. Questo risultato deriva dal fatto che una tecnica nei casi in cui è favorevole rispetto all'altra, affronta un numero minore di scelte, e di conseguenza incorre anche in un minore rischio di finire in un ottimo locale.
Oltre a questo tipo di approcci sono stati investigati anche altri metodi, allo scopo sia di diminuire i tempi computazionali sia di ottenere delle prestazioni tali da giustificare l'impiego dei metodi wrapper rispetto ai filter.
Un algoritmo di ricerca può alternativamente iniziare con un sottoinsieme casuale di feature [36, 39] anziché l'insieme vuoto o completo. A partire da questo sottoinsieme così selezionato si può procedere sia con una ricerca sequenziale in una direzione (FSS o BSS), oppure si può procedere con una ricerca bidirezionale. La ricerca, in quest'ultimo caso, procede con il valutare sia l'aggiunta che la rimozione di una feature dal sottoinsieme attuale e sceglie tra tutti i sottoinsiemi valutati il migliore secondo il criterio di valutazione.
In ogni caso la ricerca sequenziale presenta un problema, cioè che se una determinata feature viene tolta (o aggiunta) dal sottoinsieme, quella feature non può più essere reinserita (o rimossa), con il rischio non irrilevante di rimuovere feature rilevanti o aggiungere feature irrilevanti. Per ovviare a questo problema si utilizzano delle ricerche che prevedono una forma di back-tracking. Tra le più rilevanti vi sono le ricerche dette *Floating*[40] oppure le più complesse *Oscillating*[41].
La forma più semplice e classica di back-tracking si applica ad un algoritmo del tipo hill-climbing nel quale la ricerca sequenziale avviene in una sola direzione e nel caso in cui si raggiunga un così detto *plateaux*, ovvero un punto nella ricerca dove tutti i sottoinsiemi valutati non apportano alcun miglioramento alle prestazioni, si fa un passo indietro. Ovvero, si sceglie un sottoinsieme delle stesse dimensioni di quello attuale e non ancora scelto tra tutti quelli

che apportavano comunque un miglioramento. Se tra questi sottoinsiemi non ne esiste uno valido allora si fa un ulteriore passo indietro. In questo caso si valutano tutti i sottoinsiemi con cardinalità nel numero delle feature maggiore di uno rispetto a quello attuale. Il grado di back-tracking determina il numero di passi all'indietro consecutivi che siamo disposti a fare.
Come illustrato in figura 3.2, il metodo *Floating* è maggiormente articolato. La variabile k denota le attuali dimensioni del sottoinsieme, d è una costante che fissa le dimensioni target del sottoinsieme e Δ è un parametro che permette all'algoritmo di continuare nell'esecuzione anche dopo aver raggiunto d in modo da provare a migliorare ulteriormente le prestazioni. La figura in particolare mostra l'algoritmo con la direzione dominante di ricerca in avanti, ma è ovviamente anche possibile l'inverso.

Diversamente dagli altri metodi, gli *Oscillating*, vedere Figura 3.3, modifi-

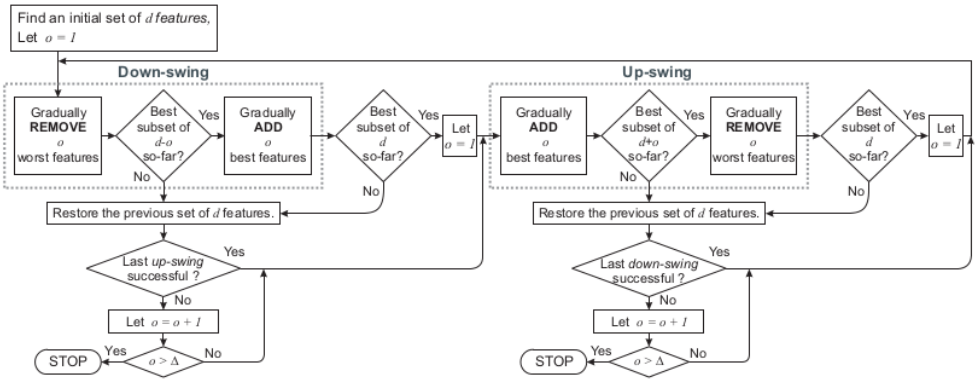
Figura 3.2: Schema semplificato dell'algoritmo di feature selection *Floating Search*



cano ripetutamente il sottoinsieme di feature considerato, ma il sottoinsieme di d feature dell'insieme originario deve avere la dimensionalità che si desidera ottenere. Per raggiungere lo scopo si alternano 2 fasi dette *swings*. La *down-swing* rimuove o feature dal corrente sottoinsieme per ottenere un nuovo sottoinsieme di $(d - o)$ feature, per poi aggiungere o feature e ottenere nuovamente un sottoinsieme di d feature. L'*up-swing* è la controparte del *down-swing*. Le due fasi eseguite alternativamente compongono un ciclo detto di oscillazione. Il metodo *Oscillating* consiste nella ripetizione del ciclo. Il valore o denota l'ampiezza dell'oscillazione, che viene aumentata ogni qual volta il ciclo non trova una sottoinsieme che migliora le prestazioni e viene reset-

tato ad 1 altrimenti. L'algoritmo si ferma quando o supera una certo valore Δ. Sia l'aggiunta che la rimozione possono essere ottenute in diversi modi, generalmente tramite metodi BSS per la rimozione e FSS per l'aggiunta.

Figura 3.3: Schema semplificato dell'algoritmo di feature selection *Oscillating Search*



Un altro metodo, detto *ibrido* [34], utilizzato nei task di classificazione, tenta di unire i vantaggi dei metodi a filter con i vantaggi dei metodi wrapper. Questo metodo sfrutta i dati acquisiti dagli algoritmi di apprendimento. Utilizza AdaBoost [42] per velocizzare il processo, e tramite l'*information gain criterion* [43] cercare di individuare le feature più promettenti in termini di miglioramento delle prestazioni, in tal modo indirizza la ricerca verso l'utilizzo di queste feature a discapito delle altre. Oltre alle prestazioni in termini di accuratezza, questi modelli [34] si pongono l'obiettivo di diminuire i tempi di calcolo. Quindi tendono ad adottare tutte le strategie che migliorano i costi computazionali rispetto agli altri wrapper, come l'adottare ricerche del tipo FSS.

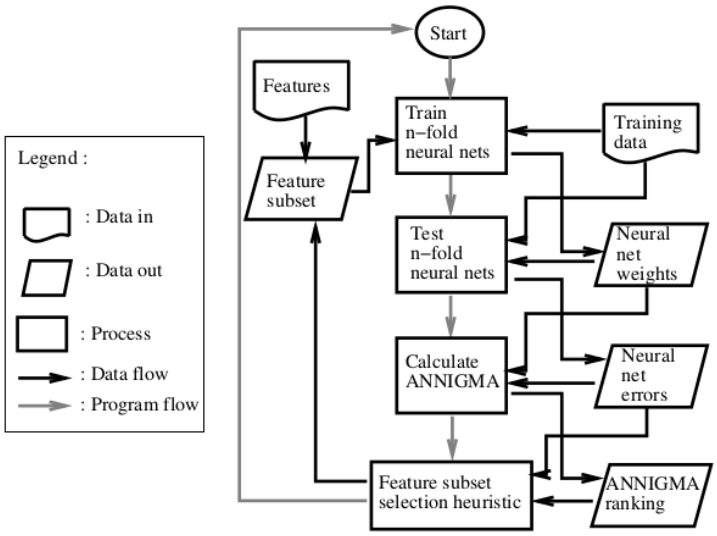
Tutti questi metodi hanno apportato delle novità nel settore della selezione di feature. In particolar modo, nei metodi *Oscillating* di tipo ibrido sono stati registrati degli ottimi risultati in termini di prestazioni [41].

Nel problema della riduzione della dimensionalità quindi si preferisce spesso utilizzare i wrapper. Infatti la ricerca della soluzione dovrebbe avvenire come parte dell'obiettivo dell'apprendimento [41].

3.3.1 Wrapper per Reti Neurali: l'indice ANNIGMA

La tecnica di feature selection a wrapper ha dimostrato in svariati contesti [33, 34, 44] di essere più efficiente in termini di accuratezza rispetto alla tecnica a filter. Ad ogni modo, risulta computazionalmente estremamente onerosa quando applicata in contesti reali caratterizzati da un largo volume di dati e insiemi di feature numerosi.
Una soluzione a questo problema [45] suggerisce un approccio che include un'euristica basata sull'analisi dei pesi di una rete neurale, al fine di guidare la ricerca del wrapper.

Figura 3.4: Diagramma di flusso dell'algoritmo ANNIGMA-Wrapper



Gli esperimenti con questa euristica chiamata ANNIGMA(Artificial Neural Net Input Gain Measurement Approximation) infatti, dimostrano un sensibile miglioramento nei tempi computazionali, pur mantenendo o persino migliorando l'accuratezza.
L'euristica ordina l'insieme delle feature per importanza, basandosi sui pesi ad esse associati. Il ragionamento che ha portato a quest'euristica è che i pesi della rete neurale possono essere un'immagine di quanto un dato segnale di input contribuisca al risultato in output. I segnali di input che sono rumorosi o irrilevanti per l'output portano un alto tasso di errore se sono associati con pesi alti in modulo. L'algoritmo di training quindi, dovrebbe ridurre il valore del modulo di questi pesi in modo che gli input associati non contribuiscano

significativamente all'output. Similmente, i pesi associati ad input significativi o senza rumore saranno invece incrementati. In una rete neurale a due livelli, uno di hidden e uno di output, l'indice ANNIGMA è definito come:

$$ANNIGMA_{ik} = \frac{LG_{ik}}{max(LG_k)} * 100$$

dove LG_{ik} è il guadagno locale per l'input i-esimo e l'output k-esimo, definito come:

$$LG_{ik} = \sum_j |W_{ij} * W_{jk}|$$

Dove W_{ij} è il peso della rete neurale associato all'input i in ingresso all'unità j dell'hidden layer, e W_{jk} è il peso associato all'uscita dell'unità j dell'hidden layer e in ingresso all'unità k dell'output layer.
La selezione di feature fatta esclusivamente sulla base di una metrica basata sui pesi può dare risultati inaffidabili. L'algoritmo ANNIGMA-wrapper integra l'euristica ANNIGMA nel wrapper al fine di raggiungere risultati attendibili.

Listing 3.1: Algoritmo di Backward Stepwise Elimination (BSE).

```
Run a training cycle with all feature, calculate ANNIGMA ranking A(0)
Let error(0) = averaged error rate and the feature subset f(0) = all features
Let H = the set of discarded features, initially empty
Let cycle counter t = 1
WHILE NOT termination condition
IF t < 4 THEN LET f(t) = top p1 % of the best features in A(t - 1)
ELSE IF error(t - 1) minimum of previous errors THEN
Let f(t) = top p2 % of the best features in A(t - 1)
ELSE IF error(t - 1) mean of previous errors THEN
Let f(t) = top p3 % of the best features in A(t - 1) plus the best feature in H
ELSE IF error(t - 1) maximum of previous errors THEN
Let f(t) = top p4 % of the best features in A(t - 1) plus the best 2 features in H
ELSE Let f(t) = top p5 % of the best features in A(t - 1) plus the best j f(t - 1)
    = 2 j features in H
Update H ; Sort H over the history-averaged ANNIGMA scores
Run a training cycle with f(t), calculate ANNIGMA ranking A(t)
t = t + 1
END WHILE and RETURN f(t)
```

La Figura 3.4 fornisce una panoramica di un generico algoritmo di feature selection nel contesto dell'ANNIGMA-wrapper. La parte interessante è come vengono selezionate le feature candidate al prossimo ciclo di apprendimento (realizzato nell'ultimo box *feature subset selection heuristic* in figura 3.4). Questa parte dell'algoritmo viene istanziata tramite una strategia di ricerca. Una strategia che ben integra l'euristica ANNIGMA nel wrapper è la *backward stepwise elimination*(BSE) [45].
Questo algoritmo, descritto dettagliatamente nel riquadro Listing 3.1, è stato progettato per accelerare la funzionalità di selezione per grandi insiemi di dati. L'idea principale è quella di eliminare un gran numero di feature apparentemente irrilevanti nei cicli iniziali e regolare la funzione di selezione del sottoinsieme nei cicli successivi. Quando le prestazioni degradano, la migliore delle feature scartate viene portata indietro nel sottoinsieme candidato.

3.4 L'Analisi di Serie di Dati

Come visto, esistono svariate tecniche per la risoluzione del problema della feature selection. In base all'insieme di feature che si deve trattare, può risultare più vantaggioso utilizzare un metodo piuttosto che un altro, questo anche in base ai criteri di valutazione che si utilizzano nella scelta.
Nell'ambito delle reti neurali, come illustrato nella Sezione 3.3.1, un'euristica utilizzata nella strategia di ricerca e basata sui pesi addestrati della rete [45], mostra dei buoni risultati sia in termini di accuratezza che di tempi computazionali.
Per la selezione di feature nel contesto di generiche sequenze di dati non sono ancora state condotte molte ricerche riguardanti i wrapper, esistono invece, svariati studi specifici di tipo filter [46, 47, 48].
Nei segnali ECG, ad esempio, è stato utilizzato un metodo filter detto *Range-Overlaps* [46] che seleziona con notevole successo le feature rilevanti in questo particolare tipo di segnale.
Nell'ambito delle reti wireless, con lo scopo di individuare intrusioni invece, è stata utilizzata una particolare tecnica a filtro detta radio *frequency fingerprinting*(RFF) [47]. Questa, unita ad un filtraggio Bayesiano, è in grado di individuare e selezionare quelle feature che presentano particolari caratteristi-

che nei casi in cui il segnale venga alterato da un intrusione.
Un altro particolare caso, per concludere, è rappresentato dall'analisi di sequenze di basi azotate. Anche se in questo ambito si ha a che fare con la generazione di feature piuttosto che con la selezione. Una tecnica che vale la pena citare [48] e che ha prodotto buoni risultati in termini del numero di feature, consiste nell'aggiunta di un'altra partizione del data-set rispetto ai classici insiemi di training, validation e test. Questo nuovo insieme chiamato *discovery set* viene utilizzato allo scopo di trovare dei *motivi* discriminatori nelle sequenze. I *motivi* saranno poi le feature per le altre partizioni, ed il loro valore sarà rappresentato dalla frequenza con la quale appariranno nelle sequenze.
È da sottolineare che tutti questi metodi sono strettamente legati al loro contesto applicativo, rendendoli difficilmente adattabili ad altri ambiti.
Un metodo filter generico per la feature selection nelle serie temporali, che è stata utilizzata nell'ambito del progetto RUBICON, è la tecnica della correlazione incrociata di teoria dei segnali. Questo metodo rappresenta la misura di similitudine di due segnali come funzione di uno spostamento o traslazione temporale applicata ad uno di essi. Intuitivamente, considerando due segnali a valori reali x e y che differiscono solamente per uno spostamento sull'asse del tempo t, si può calcolare la correlazione incrociata per mostrare di quanto y deve essere anticipato per renderlo identico ad x. La formula essenzialmente anticipa il segnale y lungo l'asse temporale t, calcolando l'integrale del prodotto per ogni possibile valore dello spostamento. Formalmente, per due segnali di energia finita x ed y la correlazione incrociata è definita come:

$$R_{xy}(t) = (x \star y)(t) \ \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} x^*(\tau)\ y(t+\tau)\, d\tau$$

in cui x^* denota il complesso coniugato di x.
Per due sequenze tempo-discreto, la correlazione incrociata è definita come:

$$R_{xy}[n] = (x \star y)[n] \ \stackrel{\text{def}}{=} \sum_{k=-\infty}^{\infty} x^*[k]\ y[n+k]$$

Per ogni segnale facente parte delle feature di input, si calcola la correlazione incrociata con ogni altro, ottenendo in tal modo una matrice detta matrice di correlazione incrociata. Le sequenze vengono eliminate mediante un al-

goritmo iterativo di selezione-eliminazione basato sui valori della matrice di correlazione.

Capitolo 4

Sviluppo della Selezione del Modello e delle Feature per ESN

In questo capitolo si descrivono le principali componenti implementate per automatizzare il processo di apprendimento e identificare automaticamente i segnali di ingresso maggiormente rilevanti.
Dopo una Sezione introduttiva 4.1 nella quale si descrivono brevemente le funzionalità sviluppate, seguono altre 4 sezioni ognuna delle quali dettaglia una delle singole componenti rilevanti.La Sezione 4.2 è incentrata sulla spiegazione del comitato di ESNs.
La Sezione 4.3 presenta un procedimento per aggiungere nuovi sample a quelli esistenti.
La Sezione 4.4 espone una procedura per il rescaling dei pesi del reservoir.
La Sezione 4.5 descrive la procedura di selezione del modello.
La Sezione 4.6 illustra gli algoritmi per la selezione di feature nelle serie temporali, presentando una nuova euristica per wrapper, detta ESNIGMA(Echo State Network Input Gain Measurement Approximation).

4.1 Componenti Sviluppati

Le funzionalità sviluppate nel contesto del progetto RUBICON fanno parte del *learning layer*, in particolare del sottosistema logico *training manager* (vedere

Sezioni 2.3.1 e 2.3.2). Le procedure implementate hanno lo scopo di automatizzare l'apprendimento per le ESN in modo efficiente. Oltre all'implementazione di una procedura di selezione del modello sono state implementate una serie di funzionalità volte all'ottimizzazione dell'apprendimento. Nel seguito si descrivono le funzionalità implementate includendo, tra parentesi quadre, di quali componenti software fanno parte all'interno del software Learning Layer (con riferimento alla figura 2.2 di Sezione 2.3.1).

- Un metodo detto di comitato che utilizza molteplici ESN [Network Mirror e Training Agent, entrambi nel modulo Training Manager].
- Una procedura che calcola e modifica i pesi del reservoir in modo da regolare le dinamiche di memoria e soddisfare la echo state property (vedere Sezione 2.2) [Network Mirror nel modulo Training Manager].
- Un procedimento che genera nuovi sample al data-set a partire da quelli esistenti. Le sequenze originali sono distorte aggiungendovi del rumore gaussiano [Training Agent nel modulo Training Manager].
- Una funzionalità di cross-fold validation che permette di utilizzare l'intero insieme dei dati a disposizione (esclusi i dati di test) sia nella fase di addestramento della rete che per la fase di validazione e selezione del modello [Network Mirror e Training Agent, entrambi nel modulo Training Manager].
- Una serie di algoritmi di feature selection per la riduzione della dimensionalità dell'input [Network Mirror e Training Agent, entrambi nel modulo Training Manager].

Oltre alle componenti volte all'apprendimento, è stato implementato un sistema per la scrittura dei risultati in file tabellari. Questa funzionalità utile in fase sperimentale permette un'agevole lettura dei risultati.
Tutti i moduli per l'apprendimento sono anche stati separati dal contesto RUBICON. Questo ha prodotto un'applicazione indipendente (chiamata SoloESN) che può essere utilizzata per l'automatizzazione dell'apprendimento con le ESN. Lo scopo è avere a disposizione un'applicazione più snella che consenta di fare sperimentazione senza essere legati al contesto del software RUBICON.

4.2 Il comitato di ESNs

Dato un insieme D di dati di training il risultato di un algoritmo di apprendimento è un predittore. Il predittore, è un ipotesi della reale funzione che genera i dati D o che presi questi dati restituisce un'informazione aggiuntiva. Gli algoritmi di apprendimento hanno quindi il compito di cercare nello spazio delle ipotesi per trovarne una che riesca a fare le migliori predizioni possibili su un particolare problema. Questa ricerca degli algoritmi di apprendimento è un problema estremamente difficoltoso.
Il comitato combina molteplici ipotesi con lo scopo di formarne una che dia migliori risultati predittivi [49]. Il termine comitato (altrimenti detto apprendimento d'insieme) nell'apprendimento automatico è un insieme concreto e finito di ipotesi alternative (differenti) utilizzate contemporaneamente per ottenere una migliore prestazione predittiva rispetto ai modelli (ipotesi) da cui è costituito.
La valutazione della predizione di un comitato richiede però una maggiore computazione rispetto alla predizione di un singolo modello. Così per giustificare la computazione maggiormente onerosa, il comitato si utilizza tipicamente nei casi in cui: l'accuratezza del modello predittivo non soddisfi le esigenze qualitative, i modelli utilizzati hanno una rapida risposta predittiva per lo scopo, o non richiedono tempi di apprendimento troppo lunghi. Nonostante ciò la strategia del comitato è ampiamente utilizzata in molteplici i casi, e persino algoritmi molto lenti possono beneficiare con successo di questa tecnica.
Il metodo che è stato sviluppato per realizzare il comitato si appoggia alle caratteristiche tipiche delle ESN. Ogni ESN infatti è costituita da un reservoir (vedere Sezione 2.2). In un ESN il reservoir è una parte che non viene addestrata ed è inizializzata tipicamente (come nel nostro caso) in modo casuale. Ovvero le connessioni e i relativi pesi sono scelti in modo casuale. Questo ci permette di ottenere diverse ipotesi utilizzando gli stessi dati per l'addestramento. Quello che viene fatto nella fase pre-apprendimento è generare diverse ESN distinte per i soli valori dei pesi di input, e diversi reservoir (non nel numero di unità ma nelle connessioni). Ognuna di queste ESN viene addestrata e valutata con gli stessi dati. In questo modo si ottengono differenti predittori (ipotesi). La predizione del comitato consiste nella media di tutte le predizioni delle ESN coinvolte.

4.3 Distorsione del Data-set

Molti data-set raccolti con esperimenti reali (non con dati generati artificialmente) tendono a non essere particolarmente ricchi di esempi, in particolare quando si trattano sequenze di dati. In questi casi il tempo necessario per raccogliere un solo esempio è almeno nell'ordine di svariati minuti. Si rende utile avere a disposizione una qualche funzione che dia la possibilità di aumentare il numero di esempi di un data-set. A partire dai dati a disposizione si è implementa una procedura che aggiunge dei nuovi esempi. Questo tipo di tecnica distorce i dati originali per averne di nuovi.
Il procedimento consiste nei seguenti passi.

1. Si prende una sequenza dei dati a disposizione e se ne produce una copia. Nel seguito si modifica soltanto la copia.
2. Per ogni passo della sequenza si aggiunge del rumore gaussiano con una distribuzione uniforme con media e varianza come parametri.
3. Si aggiunge la nuova sequenza così ottenuta al data-set.
4. Si ripetono i passi da 1 a 3 per ogni esempio del data-set.

Ogni volta che questo processo viene eseguito genera un numero di esempi aggiuntivi pari a quelli che si hanno a disposizione. Poiché la procedura distorce anche il target non è adatta ai problemi di classificazione ma solo a quelli di regressione.
L'utilizzo di questa tecnica deve essere ristretto soltanto a particolari casi. L'idea di provare questa strada infatti è sorta nell'analisi di un particolare data-set, detto turtle (vedere capitolo 5 per maggiori dettagli sul data-set stesso), i cui dati sono particolarmente rumorosi. Inoltre dall'analisi dei dati di input del data-set è emerso che diverse delle sequenze in ingresso restano costanti per alternati e lunghi periodi. In casi come questo, tramite questa tecnica lo svantaggio introdotto del rumore aggiuntivo, può essere mitigato dall'aumento del numero di esempi a disposizione per il data-set. Un data-set di training più ampio (cioè con più esempi), può apportare dei miglioramenti al livello di varianza sui risultati di validazione utilizzati per la selezione del modello, e dei miglioramenti nell'apprendimento della rete (al livello di accuratezza nel test set).

4.4 Rescaling dei pesi del reservoir

Il reservoir e le dinamiche che ne controllano lo stato interno sono una componente importante per il buon funzionamento di una ESN. Il raggio spettrale del reservoir determina:

1. le reazioni temporali del reservoir del ESN (un raggio spettrale più grande implica un degrado più lento delle informazioni in ingresso);

2. l'ammontare dell'interazione non lineare dell'input attraverso il tempo (un raggio spettrale maggiore implica maggiori interazioni a lungo termine).

Al variare del raggio spettrale variano contemporaneamente la memoria e la non linearità del reservoir [50, 51].
Infatti sia ρ il raggio spettrale per la matrice dei pesi del reservoir (o di $\tilde{\mathbf{W}}$ vedere Sezione 2.2.2), è possibile definire la capacità di memoria MC [52] del reservoir come:

$$MC = \sum_{k=1}^{\infty} \rho^2(\mathbf{u}(t-k), y_k(t)). \tag{4.1}$$

Dove $\mathbf{u}(t)$ è il vettore di ingresso al passo t della sequenza di input, e $y_k(t)$ è invece l'uscita della rete al tempo t. Per far valere la Echo State Property è necessario che ρ sia minore di uno. D'altronde il valore di MC è direttamente proporzionale al quadrato di ρ (equazione 4.1). Quindi avere valori di ρ molto vicini a uno può rivelarsi vantaggioso.
Dopo l'inizializzazione della matrice del reservoir tramite un processo che genera connessioni e valori casuali la procedura implementata calcola il risultato dell'equazione 2.13 che permette di regolare il valore di ρ.
I passi che vengono svolti sono i seguenti.

- Si ricava la matrice $\tilde{\mathbf{W}} = (1-a)I + a\hat{\mathbf{W}}$.

- Si calcola il valore attuale del raggio spettrale della matrice $\rho(\tilde{\mathbf{W}})$.

- Quindi basandoci su $\rho(\tilde{\mathbf{W}})$ si ricava il valore con cui devono essere scalati i valori di $\tilde{\mathbf{W}}$.

- Si ottiene la nuova matrice dei pesi del reservoir seguendo le rimanenti operazioni dell'equazione 2.13.

Tramite l'aggiunta di un iper-parametro si consente di immettere direttamente il valore di ρ desiderato. In tal modo si possono gestire le dinamiche di memoria del reservoir in base alle esigenze del problema trattato.

4.5 Procedura di Selezione e Validazione del Modello

La procedura di selezione del modello consiste nello scegliere determinati valori per l'insieme degli iper-parametri che definiscono il modello della rete neurale.
Come prima cosa, il data-set è partizionato in due sottoinsiemi distinti: il training set e il test set. Questa partizione può essere scelta a priori, nel qual caso può essere mantenuta la stessa nelle diverse esecuzioni, oppure viene determinata in modo casuale. La procedura che determina il modello da utilizzare può essere distinta in due fasi.
Una fase di inizializzazione descritta in Sezione 4.5.1, nella quale si definisce lo spazio della ricerca determinando tutti i possibili modelli tra i quali la procedura potrà scegliere.
Un'altra fase di validazione dettagliata in Sezione 4.5.2, nella quale la procedura seleziona il modello migliore tra quelli dello spazio precedentemente definito. Il training set viene utilizzato durante questa fase.
Per poter selezionare il modello correttamente l'insieme di addestramento primario dovrebbe essere suddiviso ulteriormente in due sottoinsiemi. L'insieme di addestramento vero e proprio e l'insieme di validazione. I dati del primo sono utilizzati esclusivamente per l'apprendimento delle reti. I dati del secondo vengono usati esclusivamente per fare una stima dell'errore delle reti e quindi per selezionare il modello. I due sottoinsiemi devono restare disgiunti. Infatti la stima dell'errore, per essere attendibile, non deve essere fatta con dati contenuti nell'insieme di addestramento.
La tecnica di cross-fold validation consente di utilizzare gli stessi dati sia per l'apprendimento che per la validazione (cioè la selezione del modello). L'insieme di addestramento primario viene partizionato in K sottoinsiemi. Ogni rete è allenata distintamente k volte. Ogni volta si seleziona un differente sottoinsieme (tra i k) da utilizzare come insieme di validazione, mentre i restanti $k-1$ sottoinsiemi vanno a formare il sottoinsieme usato per l'apprendimento.

Per cui ad ogni passo si hanno: un distinto insieme di validazione, e un diverso insieme di addestramento. Questa particolare partizione è una fold. Alla fine si ottengono quindi k errori di validazione (derivanti dalle k fold), i quali sono mediati per ottenere l'errore di validazione finale.
L'insieme di addestramento primario è quindi utilizzato sia per l'apprendimento delle ESNs, che per la stima delle prestazioni e quindi la selezione del modello.
Il test set viene invece utilizzato successivamente all'addestramento per stimare le prestazioni delle ESNs col modello selezionato.

4.5.1 Fase di Inizializzazione

Nella fase di inizializzazione:

1. Si definisce la griglia degli iper-parametri specificando gli insiemi dei valori che possono assumere. Gli iper-parametri sono:
   (a) N_R, la dimensione del reservoir,
   (b) λ, coefficiente di regolarizzazione del readout,
   (c) a, il parametro di *leaky* usato per controllare la velocità delle dinamiche del reservoir,
   (d) ρ, il raggio spettrale desiderato per la matrice dei pesi del reservoir, (può essere assegnato il valore -1 che corrisponde ad un valore fittizio, nel qual caso infatti non si ri-scalano i pesi della matrice del reservoir).
2. Si definisce il numero di differenti Echo State Networks (ESNs) E da inizializzare per ogni combinazione degli iper-parametri. Ognuna delle E ESNs, pur facendo riferimento alla stessa scelta degli iper-parametri, differirà dall'altra per: i pesi del reservoir, e per i valori dei pesi di input al reservoir.
3. Si definisce il numero di fold K per la procedura di cross fold validation.

4.5.2 Fase di Validazione

La fase di validazione avviene tramite una procedura di K-fold cross validation.

1. Per ogni fold, per ogni combinazione degli iper-parametri:
   (a) le ESN associate a tale combinazione vengono addestrate sul training set. L'errore associato alla specifica combinazione di iper-parametri viene calcolato in corrispondenza dei dati del validation set e quindi mediato sulle diverse fold.
   (b) (opzionale) Viene eseguito il wrapper (vedere Sezioni 3.2 e 4.6). Vengono rimosse una o più feature dall'input e rieseguito il passo 1a allo scopo di ridurre l'errore calcolato.
2. Viene selezionata la combinazione degli iper-parametri che minimizza l'errore di validazione mediato sulle ESNs associate.
3. Tra tutte le ESNs associate alla combinazione degli iper-parametri selezionata viene preferita quella che ha il minor errore di validazione. La ESN selezionata viene nuovamente addestrata sull'intero training set (incluso il validation-set).
4. Le prestazioni della ESN selezionata sono calcolate in corrispondenza dei dati del test set.
5. Come ultimo passo al fine dell'installazione finale della rete, dove viene usata per fare predizioni, si ri-addestra nuovamente la ESN sull'intero data set, stavolta inclusa la partizione di test.

La fase di validazione subisce alcune modifiche nel caso della tecnica con comitato, descritta in Sezione 4.2.

1. Per ogni fold, per ogni combinazione degli iper-parametri:
   (a) le ESN associate a tale combinazione vengono addestrate sul training set. Le uscite di tali ESN in corrispondenza dei dati del validation-set vengono mediate per ottenere l'uscita del comitato.

L'errore associato alla specifica combinazione di iper-parametri viene quindi calcolato sull'uscita del comitato e mediato sulle diverse fold.

(b) (opzionale) Viene eseguito il wrapper. Vengono rimosse una o più feature dall'input e rieseguito il passo 1a allo scopo di ridurre l'errore calcolato.

2. Viene quindi selezionata la combinazione degli iper-parametri con il minor errore medio di validazione.

3. Tutte le ESNs associate alla combinazione degli iper-parametri selezionata sono nuovamente addestrate sull'intero training set (incluso validation-set).

4. Le prestazioni del comitato sono calcolate in corrispondenza dei dati del test set (come al punto 1a).

5. Come ultimo passo al fine dell'installazione finale della rete, dove viene usata per fare predizioni, si ri-addestra nuovamente l'insieme di ESNs sull'intero data set, stavolta inclusa la partizione di test.

4.6 Feature Selection per Echo State Network

Nell'ambito della selezione delle feature per Echo State Network sono state implementate diverse procedure.
Una è la ricerca esaustiva, nella quale vengono calcolate le prestazioni di ogni possibile sottoinsieme di feature tra tutti quelli possibili. Questo tipo di ricerca restituisce il sottoinsieme di feature ottimale, ovvero il miglior risultato possibile. Ma questa procedura non è applicabile perché in termini di tempo la sua complessità risulta esponenziale. Infatti sia N_U il numero totale di feature del data-set, vi sono $(\sum_{l=1}^{N_U} \binom{N_U}{l})$ sottoinsiemi possibili. Per cui tale tipo di ricerca è stata sviluppata soltanto a scopo di confronto sperimentale.
Le altre procedure implementate selezionano i sottoinsiemi di feature da valutare tramite un algoritmo greedy del tipo hill-climbing. Inizialmente si considera l'intero insieme delle feature come il migliore sottoinsieme. Ad ogni passo, sono valutati tutti i sottoinsiemi con una feature in meno del migliore sottoinsieme. Se esiste tra questi uno che migliora le prestazioni dell'attuale

migliore sottoinsieme, esso viene selezionato per sostituirlo. Tale procedura si ripete finché si riesce ad ottenere un miglioramento.
Genericamente, per ogni sottoinsieme delle feature da valutare, si deve riaddestrare tutta la rete neurale e calcolarne le prestazioni di validazione. Tale valore viene così utilizzato sia come valore per la ricerca nello spazio dei sottoinsiemi di feature, che come valore per sostituire il migliore sottoinsieme attuale.
Visti gli alti costi computazionali della procedura su detta, nell'ambito delle Echo State Network è stata sviluppata una nuova euristica (Echo State Network Input Gain Measurement Approximation) che guida questo tipo di ricerca. In tal modo non è necessario il ri-addestramento e il calcolo delle prestazioni di validazione per tutti i sottoinsiemi presi in considerazione. A partire dai valori dei pesi dell'ESN si calcola un indice positivo per ogni feature di input che ne rappresenta la rilevanza. Di fatti, i pesi del readout addestrato di un ESN possono essere visti come come un indicatore di quanto un segnale di input contribuisce per il risultato di output. A partire da questa considerazione è possibile calcolare questo indice.

4.6.1 Sviluppo dell'Euristica ESNIGMA

Consideriamo l'equazione dello stato interno della LI-ESN 2.11; possiamo riscriverla per la singola unità j del reservoir come segue

$$\mathbf{x}_j(t) = (1-a)\mathbf{x}_j(t-1) + af(\mathbf{W}_j^{in}\mathbf{u}(t)) + \hat{\mathbf{W}}_j\mathbf{x}(t-1)) \tag{4.2}$$

in funzione di una singola feature di input i essa si può riscrivere come:

$$\mathbf{x}_j(t) = (1-a)\mathbf{x}_j(t-1) + af(\mathbf{W}_{ji}^{in}\mathbf{u}_i(t)) + \mathbf{C}_{ji} + \sum_{z=1}^{N_R}\hat{\mathbf{W}}_{jz}\mathbf{x}_z(t-1)) \tag{4.3}$$

dove $\mathbf{C}_{ji}$ rappresenta il valore, che può essere considerato costante, di tutti gli altri input (incluso il bias) all'unità del reservoir j.
Essendo f in generale una funzione non lineare l'equazione 4.3 risulta ancora molto complessa da calcolare. Ma essendo interessati ad un'approssimazione della misura con la quale gli input influiscono sull'output, sia F una funzione lineare tale che $segno(F(x)) = segno(f(x))$ allora la possiamo sostituire F ad f. Con tale osservazione, si può scrivere la funzione delle uscite della ESN a

partire dalla 2.2 per la k-esima unità di output come:

$$\mathbf{y}_k(\mathbf{t}) = \sum_{j=1}^{N_R} \check{\mathbf{W}}_{kj}((1-a)\mathbf{x}_j(t-1) + aF(\mathbf{W}_{ji}^{in}\mathbf{u}_i(t) + \mathbf{C}_{ji} + \sum_{z=1}^{N_R} \hat{\mathbf{W}}_{jz}\mathbf{x}_z(t-1))). \tag{4.4}$$

Quindi è possibile definire il guadagno locale LG_{ki} che un particolare input i apporta ad uno specifico output k come:

$$LG_{ki} = |\frac{\partial \mathbf{y}_k(n)}{\partial \mathbf{u}_i(n)}|. \tag{4.5}$$

Poiché non siamo interessati al valore assoluto delle somme ma alla magnitudine del contributo dei si singoli elementi, siano essi positivi o negativi, si calcano le somme dei valori assoluti piuttosto che il valore assoluto delle somme.
Sviluppando la derivata parziale abbiamo:

$$|\frac{\partial \mathbf{y}_k(n)}{\partial \mathbf{u}_i(n)}| = \sum_{j=1}^{N_R} |\check{\mathbf{W}}_{kj}\frac{\partial \mathbf{x}_j(n)}{\partial \mathbf{u}_i(n)}| \tag{4.6}$$

dove

$$\frac{\partial \mathbf{x}_j(n)}{\partial \mathbf{u}_i(n)} = ((1-a)\frac{\partial \mathbf{x}_j(n-1)}{\partial \mathbf{u}_i(n)} + aF(\mathbf{W}_{ji}^{in} + \sum_{z=1}^{N_R} \hat{\mathbf{W}}_{jz}\frac{\partial \mathbf{x}_z(n-1)}{\partial \mathbf{u}_i(n)})) \tag{4.7}$$

infatti $\frac{\partial \mathbf{u}_i(n)}{\partial \mathbf{u}_i(n)} = 1$ e $\frac{\partial \mathbf{C}_{ji}}{\partial \mathbf{u}_i(n)} = 0$.
Resta da risolvere $\frac{\partial \mathbf{x}_j(n-1)}{\partial \mathbf{u}_i(n)}$.

$$\frac{\partial \mathbf{x}_j(n-1)}{\partial \mathbf{u}_i(n)} = (1-a)\frac{\partial \mathbf{x}_j(n-2)}{\partial \mathbf{u}_i(n)} + aF(\mathbf{W}_{ji}^{in}\frac{\partial \mathbf{u}_i(n-1)}{\partial \mathbf{u}_i(n)} + \sum_{z=1}^{N_R} \hat{\mathbf{W}}_{jz}\frac{\partial \mathbf{x}_z(n-2)}{\partial \mathbf{u}_i(n)}). \tag{4.8}$$

Dal momento che $\mathbf{x}(0)$ non dipende dall'input otteniamo $\frac{\partial \mathbf{x}_j(0)}{\partial \mathbf{u}_i(n)} = 0$.
Il problema si sposta perciò nel risolvere $\frac{\partial \mathbf{u}_i(n-1)}{\partial \mathbf{u}_i(n)}$. Naturalmente se il valore di un determinato input in un certo istante di tempo della sequenza fosse scorrelato dai precedenti, non saremmo interessati nel fare una qualche codifica della sequenza. Assumendo quindi che il valore di $\mathbf{u}_i(t)$ dipenda dai precedenti allora esiste una qualche funzione che possa determinare il valore di $\mathbf{u}_i(t)$ in base ai precedenti. Ma se tale funzione esiste può esistere anche l'inversa che

calcoli $\mathbf{u}_i(t-l)$ in base ai successivi (per un qualunque $l > 0$). In generale quindi $\frac{\partial \mathbf{u}_i(t-l)}{\partial \mathbf{u}_i(t)} \neq 0$. Poiché non siamo interessati ai particolari valori assunti dalla sequenza degli input, possiamo considerare $\frac{\partial \mathbf{u}_i(t-l)}{\partial \mathbf{u}_i(t)} = q$ con q una qualunque costante diversa da 0.
Possiamo riscrivere la 4.8 come:

$$\frac{\partial \mathbf{x}_j(n-1)}{\partial \mathbf{u}_i(n)} = (1-a)\frac{\partial \mathbf{x}_j(n-2)}{\partial \mathbf{u}_i(n)} + aF(\mathbf{W}^{in}_{ji} q + \sum_{z=1}^{N_R} \hat{\mathbf{W}}_{jz} \frac{\partial \mathbf{x}_z(n-2)}{\partial \mathbf{u}_i(n)}).$$

Ponendo $q = 1$.

$$\frac{\partial \mathbf{x}_j(n-1)}{\partial \mathbf{u}_i(n)} = (1-a)\frac{\partial \mathbf{x}_j(n-2)}{\partial \mathbf{u}_i(n)} + aF(\mathbf{W}^{in}_{ji} + \sum_{z=1}^{N_R} \hat{\mathbf{W}}_{jz} \frac{\partial \mathbf{x}_z(n-2)}{\partial \mathbf{u}_i(n)}). \quad (4.9)$$

Si mostra un esempio di derivazione con $n = 2$

$$|\frac{\partial \mathbf{y}_k(2)}{\partial \mathbf{u}_i(2)}| = \sum_{j=1}^{N_R} |\check{\mathbf{W}}_{kj}((1-a)\frac{\partial \mathbf{x}_j(1)}{\partial \mathbf{u}_i(2)} + aF(\mathbf{W}^{in}_{ji} + \sum_{z=1}^{N_R} \hat{\mathbf{W}}_{jz} \frac{\partial \mathbf{x}_z(1)}{\partial \mathbf{u}_i(2)}))| \quad (4.10)$$

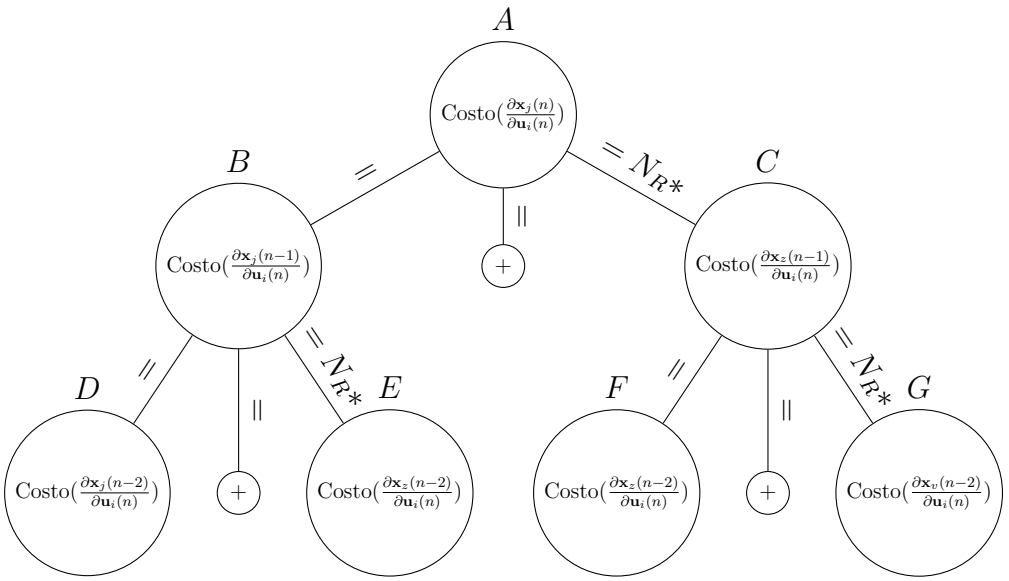
essendo $\frac{\partial \mathbf{x}_j(0)}{\partial \mathbf{u}_i(2)} = 0$ e $\frac{\partial \mathbf{u}_i(1)}{\partial \mathbf{u}_i(2)} = 1$ otteniamo:

$$\frac{\partial \mathbf{x}_j(1)}{\partial \mathbf{u}_i(2)} = aF\mathbf{W}^{in}_{ji} \quad (4.11)$$

risolvendo così la 4.10.

Per l'implementazione del calcolo del ESNIGMA si calcola l'ordine del costo computazionale in base all'approccio risolutivo scelto.
Il modo naive di risolvere l'equazione 4.9 è con un approccio top-down. Cioè a partire da $\frac{\partial \mathbf{x}_j(n)}{\partial \mathbf{u}_i(n)}$ si risolvono le derivate parziali con $n-1$, poi quelle con $n-2$ fino a $n = 0$. Sia la funzione F la più semplice funzione lineare utilizzabile, cioè l'identità. Per risolvere $\frac{\partial \mathbf{x}_j(n)}{\partial \mathbf{u}_i(n)}$ secondo l'equazione 4.9 sono necessarie $2 + N_R$ moltiplicazioni più i costi computazionali delle altre derivate parziali. Continuando in questo modo possiamo disegnare uno schema ad albero (come quello di Figura 4.1) che riassume i costi computazionali.

Figura 4.1: Grafo per il supporto nell'analisi dei costi dell'algoritmo ESNIGMA.



L'altezza di quest'albero è $n-1$. Escludendo i nodi che indicano l'operazione somma l'albero ha 2^{n-1} foglie. Per un costo computazionale totale esponenziale rispetto ad n.

É possibile notare dall'albero che si calcola più volte lo stesso valore. Considerando l'albero illustrato nel livello due, i nodi E e G condivideranno gli stessi indici. Ovvero $\forall z \in E$, $\exists v \in G$ tale che $z = v$. Inoltre nel nodo F esiste uno $z = j$ del l'indice j del nodo D. É allora possibile diminuire i costi computazionali mantenendo in memoria i valori delle derivate parziali $\frac{\partial \mathbf{x}(t)}{\partial \mathbf{u}_i(n)}$ per ogni t. Per utilizzare meno memoria è inoltre preferibile un approccio bottom-up, così che una volta calcolato il valore del vettore $\frac{\partial \mathbf{x}(t)}{\partial \mathbf{u}_i(n)}$ è possibile liberare la memoria dai valori del vettore $\frac{\partial \mathbf{x}(t-1)}{\partial \mathbf{u}_i(n)}$.

Quindi per risolvere

$$\frac{\partial \mathbf{x}(t)}{\partial \mathbf{u}_i(n)} = (1-a)\frac{\partial \mathbf{x}(t-1)}{\partial \mathbf{u}_i(n)} + a(\mathbf{W}_i^{in} + \sum_{z=1}^{N_R} \hat{\mathbf{W}}_z \frac{\partial \mathbf{x}_z(t-1)}{\partial \mathbf{u}_i(n)}) \tag{4.12}$$

ad un generico passo t per risolvere $\frac{\partial \mathbf{x}(t)}{\partial \mathbf{u}_i(n)}$ sono necessarie $2N_R + N_R^2$ moltiplicazioni. Quest'operazione viene ripetuta per ogni passo t. Così facendo si ha una complessità lineare rispetto ad n.

Capitolo 5

Risultati Sperimentali

Questo capitolo presenta i risultati sperimentali ottenuti del sistema di apprendimento. Le principali componenti delle quali si verifica il funzionamento sono:

1. Il sistema di selezione automatica del modello.
2. La tecnica del comitato.
3. La procedura (di tipo wrapper) di selezione automatica delle feature.

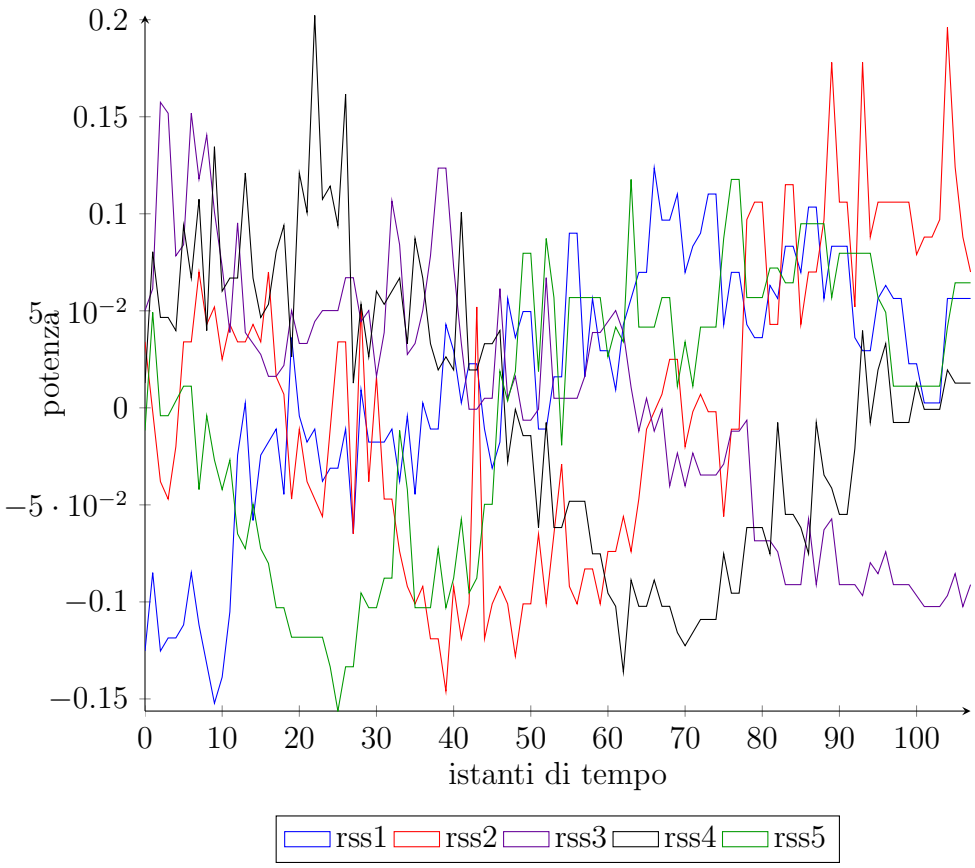
Il capitolo si apre con la Sezione 5.1, che presenta una panoramica sugli scenari e i differenti data-set utilizzati. La Sezione 5.2 mostra i risultati ottenuti su i data-set con lo scopo di verificare il funzionamento del sistema.

5.1 Componenti e Scenario degli Esperimenti

Gli esperimenti condotti riguardano principalmente due data-set, uno detto *Turtle* e l'altro *Stella Maris*. Lo scopo dei data-set è quello di individuare la posizione di un robot (di tipo carrello) in un ambiente reale.
La raccolta dei dati per i due data-set è stata fatta in due luoghi distinti: in un corridoio del dipartimento di informatica (data-set *Turtle*), e in un corridoio dell'ospedale Stella Maris (data-set omonimo). La frequenza di campionamento dei dati è di 4hz.
Per costruire la necessaria rete di sensori negli esperimenti sono stati usati una serie di dispositivi wireless detti *mote* [53].
Negli esperimenti i mote possono avere due ruoli, uno dei *mote*, che funge da

trasmettitore, è posizionato su un robot carrello, mentre gli altri messi in punti predeterminati e fissi dell'ambiente sono detti *ancore*. Le ancore calcolano i *Received Singnal Strength* (RSS), che è una misura della potenza del segnale radio ricevuto dal trasmettitore (si può vedere un esempio delle sequenze in Figura 5.1).

Figura 5.1: Esempio di sequenze di RSS normalizzate (prese dal data-set *Turtle*)



Quest'informazione, è utilizzata dalla ESN per predire la posizione del trasmettitore e quindi del robot in tempo reale. Il tipo di compito che si realizza su questi data-set è detto di localizzazione, in quanto l'output comprende due coordinate (ascissa e ordinata) che indicano la posizione del robot in un determinato ambiente. La mappa e i target sono stati ricavati tramite un sistema di localizzazione laser basato su un approccio SLAM [54]. Nella raccolta di entrambi i data-set sono stati utilizzati anche dei landmark di riferimento.

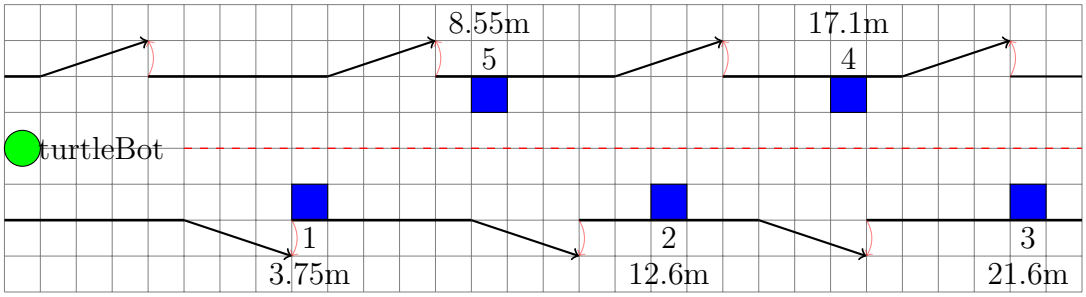
5.1.1 Turtle

Nel data-set *Turtle* il robot carrello è un Turtle bot di prima generazione (vedere Figura 5.2) [**?**], dalle ridotte capacità di carico (4-5 chilogrammi in base al tipo di pavimento).

Figura 5.2: Modello di robot utilizzato nella raccolta dati del data-set *Turtle*

Il percorso, in linea retta, è stato fatto in un corridoio nel dipartimento di informatica dell'Università di Pisa (vedere Figura 5.3). Il numero di ancore posizionate nel corridoio è 5, le ancore sono a livello del pavimento. La distanza percorsa dal robot è di circa 20m, in ogni sequenza il punto di partenza e arrivo sono sempre gli stessi (alla fine di una corsa il robot veniva riposizionato nel punto iniziale). I dati di target sono stati raccolti per mezzo di una camera RGBD kinect (amcl) [55]. Le sequenze raccolte sono in tutto 12, e sono state numerate da 0 a 11. Di queste 12 sequenze, 4 sono state utilizzate per comporre il test set (la sequenza 0, la 3 la 7, e la 11).

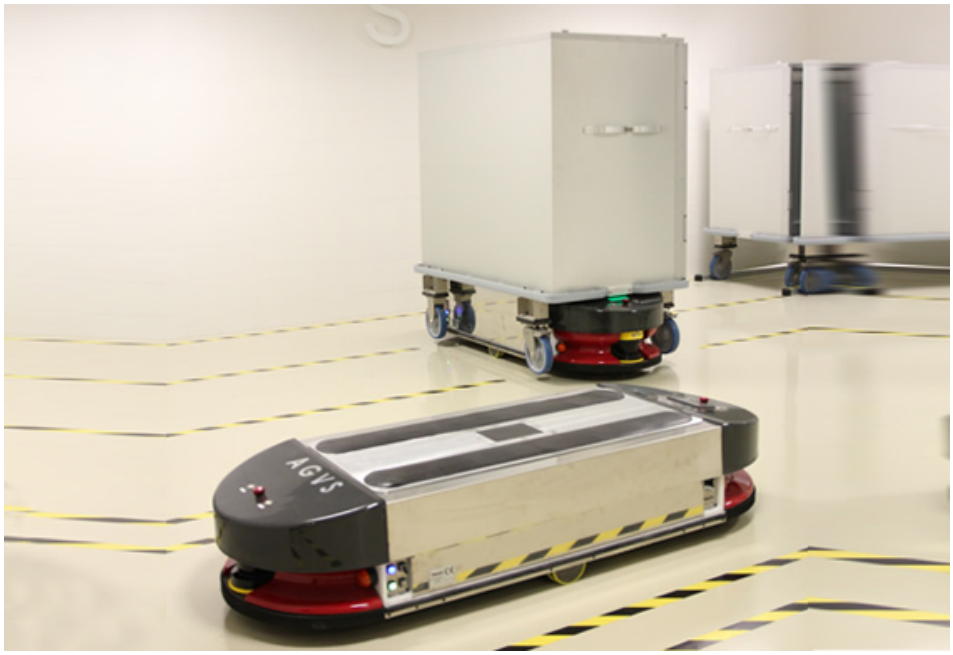
Figura 5.3: Mappa del corridoio del data-set *Turtle*. I quadrati blu sono le ancore con il loro ID e la distanza dalla posizione iniziale del Turtle bot (cerchio in verde).



5.1.2 Stella Maris

Nel data-set *Stella Maris* il robot carrello è un Robotnik AGVS (vedere Figura 5.4) capace di trasportare svariati kg di peso (fino a 500Kg).

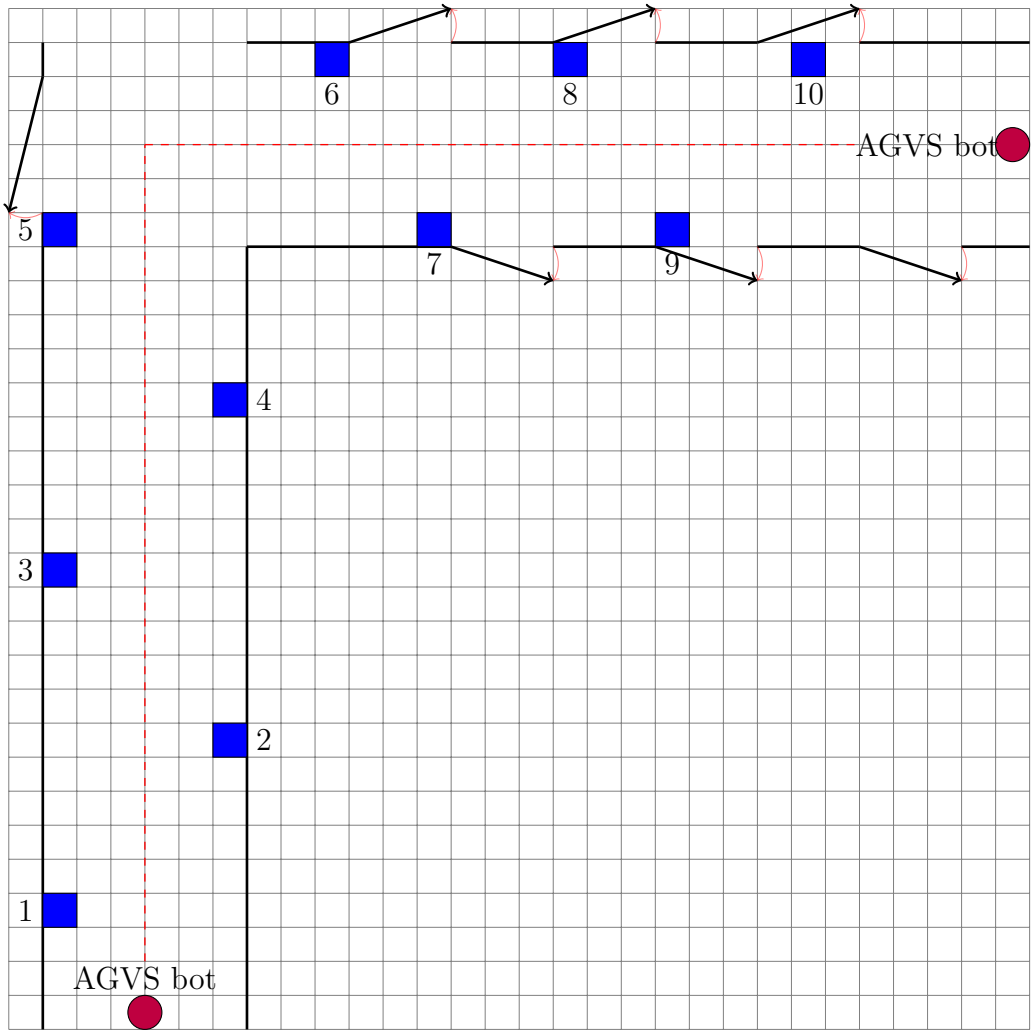
Figura 5.4: Modello di robot utilizzato nella raccolta dati del data-set *Stella Maris*.



Il percorso effettuato all'ospedale Stella Maris di Marina di Pisa prevede una curva a 90^o (vedere Figura 5.5), andando effettivamente a coprire due corridoi. Il numero di ancore posizionate lungo i 2 corridoi è 10, le prime 5

ad una altezza di circa 80cm dal pavimento nel primo (quello verticale nella Figura 5.5), le seconde 5 ad un'altezza di circa 2m, nel secondo. La distanza percorsa dal robot è di circa 40m, nelle sequenze raccolte la partenza e l'arrivo del robot sono alternate.

Figura 5.5: Mappa del corridoio del data-set *Stella Maris*. I quadrati blu sono le ancore con il loro ID, i cerchi in rosso sono le posizioni di partenza del robot AGVS.



Alla fine di ogni corsa il robot non viene riposizionato nel punto di partenza ma è stato invece lasciato nella posizione finale, la quale è quindi la posizione iniziale per la sequenza successiva. I dati del target sono stati raccolti per mezzo del laser del robot AGVS [56]. Le sequenze di dati usate sono state

raccolte durante la notte e sono 17, numerate da 0 a 16. Di queste 17 sequenze, 4 sono state utilizzate per costruire il test set (le sequenze da 13 a 16).

5.1.3 Impostazioni degli Esperimenti

Altri iper-parametri salienti sono le dimensioni del reservoir $N_R \in \{50,100,200,500\}$ unità, la sua connettività è pari al 10%, e il raggio spettrale è per il data-set *Stella Maris* $\rho \in \{0.8, 0.9, 0.99\}$ oppure $\rho = 0.99$ nel caso si faccia feature selection, mentre nel data-set *Turtle* $\rho \in \{nc., 0.8, 0.9, 0.99\}$ (dove nc. sta per non calcolato, utilizzato soltanto nei casi in cui ρ è certamente molto minore di uno).
Un aspetto interessante riguarda la dimensione dei pesi, nel senso del numero di bits utilizzati per codificare $\mathbf{W}^{in}$ e $\hat{\mathbf{W}}$. Negli esperimenti presentati il valore dei pesi, per le matrici $\mathbf{W}^{in}$ e $\hat{\mathbf{W}}$, utilizzerà un piccolo insieme di valori possibili diversi da zero. Il numero dei differenti valori di questa *codifica ridotta* dei pesi è denotato da N_w [14]. Ogni peso continua ad essere un numero in virgola mobile a precisione singola, ma stavolta, per codificare ogni peso nelle matrici della rete sono sufficienti $\log_2 N_w$ bits.
Riguardo questa codifica ridotta dei pesi si ha $N_w = 16$ come il numero dei possibili valori nell'intervallo [-0.4,0.4], e ciò può ridurre a soltanto 4 il numero di bits per la codifica di un peso nelle matrici $\mathbf{W}^{in}$ e $\hat{\mathbf{W}}$. Sono state utilizzate LI-ESNs, e l'insieme dei valori possibili per il Leaky parameter cambia in base al data-set di riferimento: per il data-set *Stella Maris* sono stati utilizzati due insiemi $a \in \{0.05, 0.1, 0.5\}$ e visti i risultati sperimentali ottenuti successivamente regolato a $a \in \{0.05, 0.1, 0.2\}$; nel caso del data-set *Turtle* $a \in \{0.05, 0.1, 0.2, 0.5\}$. I risultati descritti nel seguito sono la media di 5 ESN indipendenti e inizializzate randomicamente, utilizzate anche nei casi di comitato. Il readout è stato addestrato utilizzando il metodo ridge regression con parametro di regolarizzazione $\lambda \in \{0.001, 0.01, 0.1, 1\}$.
Il 10% di ogni sequenza di training è presentata al reservoir perché si stabilizzi lo stato interno, in accordo col transiente iniziale.
Gli iper-parametri da utilizzare prima della valutazione con il test set sono stati scelti tramite *4-fold cross validation* come descritto in 4.5.

5.2 Risultati

I dati riportati nelle tabelle successive sono in metri. Per ogni sequenza campione il valore dell'errore è calcolato come la media della distanza euclidea di ogni singolo output rispetto al relativo target. Essendo gli scopi dei data-set di regressione in seguito ci si riferisce all'accuratezza come all'inverso dell'errore.
La sperimentazione ha coinvolto una fase di analisi preliminare fatta durante lo sviluppo del software (i cui dettagli sono in Appendice B.1), ed una fase sperimentale principale svoltasi a sviluppo ultimato.
L'aspetto primario degli esperimenti è stato verificare che il sistema implementato permettesse di generare un modello di ESN in modo, accurato e automatico.

5.2.1 Sperimentazione della Model Selection

Per ogni iper-parametro è impostato un insieme di valori, il sistema implementato considera tutte le possibili configurazioni dei valori degli iper-parametri. In base ai risultati ottenuti in fase di validazione sceglie il modello che è caratterizzato dal valore di errore medio (di validazione) più basso.
Basandosi sull'errore medio di validazione (e non di addestramento) il sistema resta sensibile al rischio di overfitting. Si scartano infatti quei modelli che si specializzano eccessivamente sui dati dell'insieme di addestramento, e che commettono invece errori maggiori sui dati riguardanti lo stesso compito ma che non fanno parte dell'insieme di allenamento.
Nel seguito si illustrano i risultati ottenuti sia con il data-set *Turtle* che con il data-set *Stella Maris*, e si mostra come il sistema sia efficace nello scegliere in modo automatico il modello migliore dal punto di vista dell'accuratezza.
I risultati nelle figure 5.6, 5.7 mostrano i valori dell'errore medio (calcolati come distanza Euclidea) in fase di training e in fase di validazione al crescere del numero delle unità del reservoir.

Figura 5.6: Risultati ottenuti con il data-set *Stella Maris* al crescere del numero di unità del reservoir.

(a) *I seguenti parametri* $\rho = 0.99$ $\lambda = 0.1$*(parametro di regolarizzazione)* $a = 0.1$*(parametro Leaky) sono fissati. I valori scelti per gli iper-parametri si basano sul modello selezionato automaticamente dal sistema in fase di validazione (per dettagli sui parametri vedere sezione 4.5.1).*

NU	TrainErr	ValidErr
50	0.6071273	0.767472
100	0.37427172	0.61598176
200	0.23197356	0.4836428
500	0.12869021	0.4406168

(b) *Grafico al crescere del numero di unità del reservoir. In blu l'errore di training in rosso quello di validazione.*

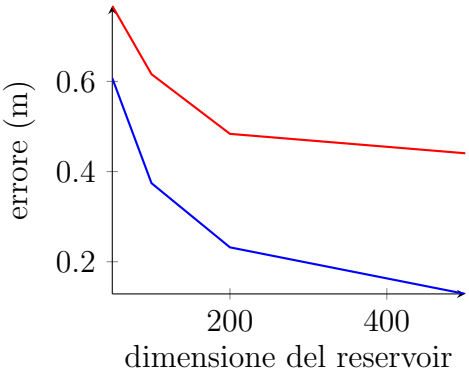
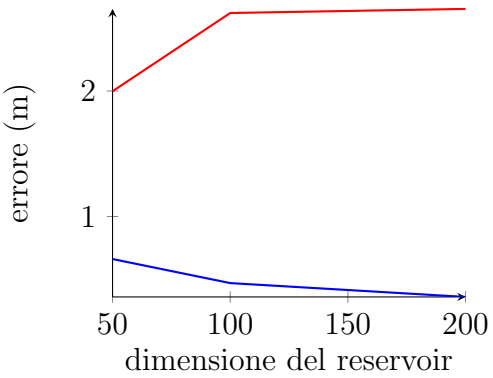


Figura 5.7: Risultati ottenuti con il data-set *Turtle* al crescere del numero di unità del reservoir.

(a) *I seguenti parametri* $\rho = 0.99$ $\lambda = 0.1$*(parametro di regolarizzazione)* $a = 0.2$*(parametro Leaky) sono fissati. I valori scelti per gli iper-parametri si basano sul modello selezionato automaticamente dal sistema in fase di validazione (per dettagli sui parametri vedere sezione 4.5.1).*

ResDim	TrainErr	ValidErr
50	0.66069263	1.9979329
100	0.4689618	2.621224
200	0.35869282	2.6539204

(b) *Grafico al crescere del numero di unità del reservoir. In blu l'errore di training in rosso quello di validazione.*



Per il data-set *Stella Maris*, il sistema implementato ha selezionato autonomamente il modello con dimensione del reservoir pari a 200 unità. La scelta automatica fatta dal sistema è corretta infatti la Figura 5.6 mostra che non c'è rischio di overfitting al crescere del numero delle unità del reservoir. La Figura 5.7 mostra chiaramente come il data-set *Turtle* vada invece subito in overfitting al crescere della dimensione del reservoir. Quindi anche in questo caso il sistema ha agito esattamente selezionando automaticamente il modello

con 50 neuroni nel reservoir.
Un altro importante iper-parametro che regola l'apprendimento è quello di regolarizzazione.
Le figure 5.8, 5.9 illustrano come variano i risultati degli errori medi (calcolati come distanza Euclidea) in fase di apprendimento e in fase di validazione, rispettivamente per il data-set *Turtle* e per il data-set *Stella Maris*. I grafici mostrano gli andamenti degli errori medi al decrescere del parametro di regolarizzazione λ.

Figura 5.8: Risultati ottenuti con il data-set *Stella Maris* al variare del valore del parametro λ di regolarizzazione.

(a) *I seguenti parametri* $\rho = 0.99$ $N_R = 500$*(dimensione del reservoir)* $a = 0.1$*(parametro Leaky) sono fissati. I valori scelti per gli iper-parametri si basano sul modello selezionato automaticamente dal sistema in fase di validazione (per dettagli sui parametri vedere sezione 4.5.1).*

Reg	TrainErr	ValidErr
0.001	0.067746505	0.47069523
0.01	0.083442	0.45116574
0.1	0.12869021	0.4406168
1.0	0.22295836	0.4738701

(b) *Grafico al decrescere del valore del parametro λ di regolarizzazione. In blu l'errore di training in rosso quello di validazione.*

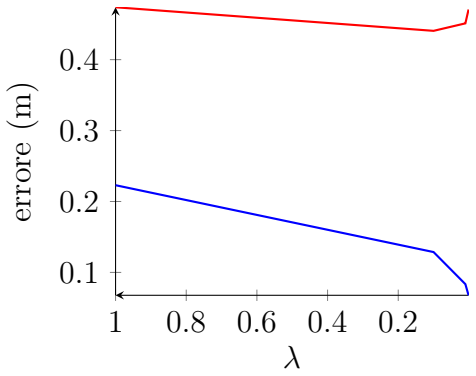
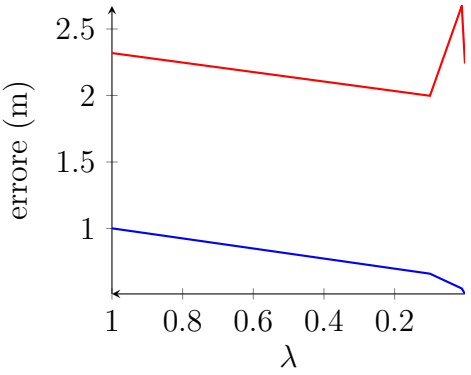


Figura 5.9: Risultati ottenuti con il data-set *Turtle* al variare del valore del parametro λ di regolarizzazione.

(a) *I seguenti parametri* $\rho = 0.99$ $N_R = 50$*(dimensione del reservoir)* $a = 0.2$*(parametro Leaky) sono fissati. I valori scelti per gli iper-parametri si basano sul modello selezionato automaticamente dal sistema in fase di validazione (per dettagli sui parametri vedere sezione 4.5.1).*

Reg	TrainErr	ValidErr
0.001	0.50864613	2.2402496
0.01	0.5496285	2.673524
0.1	0.66069263	1.9979329
1.0	1.0016991	2.3189726

(b) *Grafico al decrescere del valore del parametro λ di regolarizzazione. In blu l'errore di training in rosso quello di validazione.*



In entrambi i casi se il parametro di regolarizzazione è troppo piccolo i modelli risultanti hanno un alto rischio di overfitting. Infatti dai grafici si può facilmente vedere un incremento dell'errore di validazione in corrispondenza dei valori piccoli di λ (parametro di regolarizzazione). Il sistema implementato in entrambi i casi ha selezionato automaticamente i modelli con $\lambda = 0.1$. Per cui considerando anche il parametro di regolarizzazione i risultati mostrano che il sistema riesce a scartare i modelli con un alto rischio di overfitting, e contemporaneamente a scegliere automaticamente il modello con l'accuratezza migliore tra i rimanenti.

Gli iper-parametri rimanenti da considerare sono il parametro di Leaky a ed il parametro ρ ovvero il raggio spettrale desiderato per la matrice del reservoir. Il primo parametro a influenza soltanto le dinamiche temporali del reservoir (vedere Sezione 2.2.2), mentre il secondo, oltre a influenzare le dinamiche temporali del reservoir, è direttamente proporzionale alle capacità di memoria dello stesso.

Riguardo il parametro di Leaky a le figure 5.10, 5.11 illustrano i risultati degli errori medi (calcolati come distanza Euclidea) in fase di apprendimento e in fase di validazione. I grafici presentano gli andamenti degli errori medi al decrescere dello stesso parametro. In tal modo si mostra come si modifica l'errore al rallentare della reazione del reservoir rispetto alla sequenza di dati in ingresso.

Figura 5.10: Risultati ottenuti con il data-set *Stella Maris* al variare del valore del parametro di Leaky a.

(a) *I seguenti parametri* $\rho = 0.99$ $\lambda = 0.1$*(parametro di regolarizzazione)* $N_R = 500$*(dimensione del reservoir) sono fissati. I valori scelti per gli iper-parametri si basano sul modello selezionato automaticamente dal sistema in fase di validazione (per dettagli sui parametri vedere sezione 4.5.1).*

Leaky	TrainErr	ValidErr
0.05	0.12950729	0.5576133
0.1	0.12869021	0.4406168
0.2	0.20659342	0.56024003

(b) *Grafico al decrescere del valore del parametro di Leaky a. In blu l'errore di training in rosso quello di validazione.*

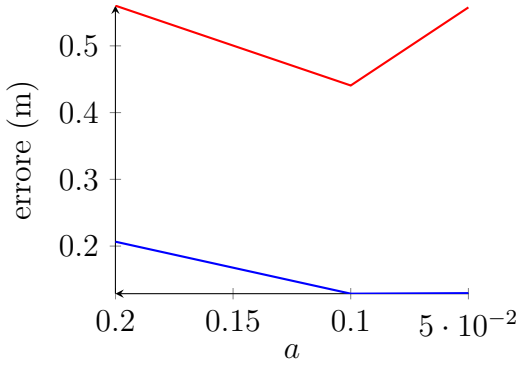
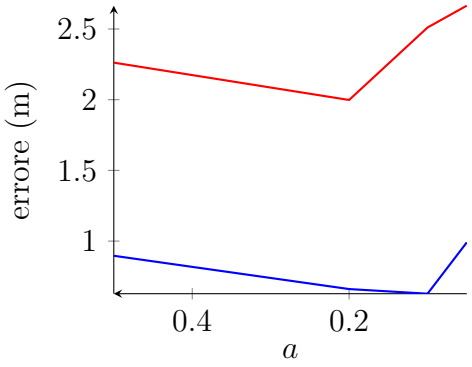


Figura 5.11: Risultati ottenuti con il data-set *Turtle* al variare del valore del parametro di Leaky a.

(a) *I seguenti parametri* $\rho = 0.99$ $\lambda = 0.1$*(parametro di regolarizzazione)* $N_R = 50$*(dimensione del reservoir) sono fissati. I valori scelti per gli iper-parametri si basano sul modello selezionato automaticamente dal sistema in fase di validazione (per dettagli sui parametri vedere sezione 4.5.1).*

Leaky	TrainErr	ValidErr
0.05	0.99047416	2.664854
0.1	0.6279086	2.5105538
0.2	0.66069263	1.9979329
0.5	0.8961387	2.2620044

(b) *Grafico al decrescere del valore del parametro di Leaky a. In blu l'errore di training in rosso quello di validazione.*



Nel data-set *Turtle* si può osservare che l'errore in fase di apprendimento con $a = 0.1$ è minore rispetto all'errore con $a = 0.2$, ma ciò non è altrettanto vero in fase di validazione dove l'errore invece è chiaramente maggiore. Questo si verifica perché con il valore di $a = 0.1$ i modelli di ESN allenati si specializzano troppo sui dati dell'insieme di apprendimento, commettendo così un

errore maggiore sui dati che non ne fanno parte. Il sistema sceglie quindi correttamente il modello con il valore di $a = 0.2$. Nel data-set *Stella Maris* analogamente $a = 0.1$(scelto dal sistema) e $a = 0.05$ hanno errori in fase di apprendimento molto simili, ma l'errore di $a = 0.05$ in fase di validazione è più alto. Questo ancora una volta perché per quel valore del parametro di Leaky (0.05) i modelli di ESN allenati si specializzano eccessivamente sui dati dell'insieme di training. Per cui anche per il parametro di Leaky il sistema riesce a scegliere in modo automatico il modello corretto evitando il rischio di overfitting.
Si analizza quindi il comportamento al variare dell'iper-parametro ρ (raggio spettrale della matrice del reservoir).
Le figure 5.12, 5.13 illustrano i risultati degli errori medi (calcolati come distanza Euclidea) in fase di apprendimento e in fase di validazione, i grafici presentano gli andamenti degli errori medi al crescere del parametro ρ.

Figura 5.12: Risultati ottenuti con il data-set *Stella Maris* al variare del valore del parametro ρ.

(a) *I seguenti parametri* $a = 0.1$ $\lambda = 0.1$*(parametro di regolarizzazione)* $N_R = 500$*(dimensione del reservoir) sono fissati. I valori scelti per gli iper-parametri si basano sul modello selezionato automaticamente dal sistema in fase di validazione (per dettagli sui parametri vedere sezione 4.5.1).*

Rho	TrainErr	ValidErr
0.8	0.16572975	0.51152146
0.9	0.14952007	0.51301
0.99	0.12063388	0.4421693

(b) *Grafico al crescere del valore del parametro ρ.*
In blu l'errore di training in rosso quello di validazione.

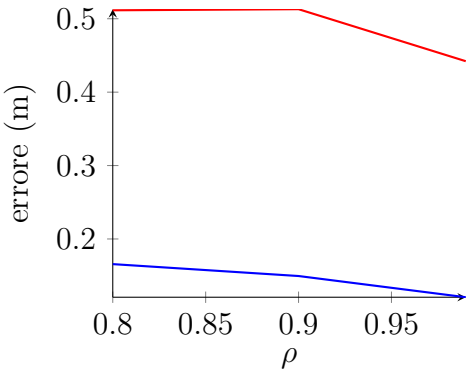
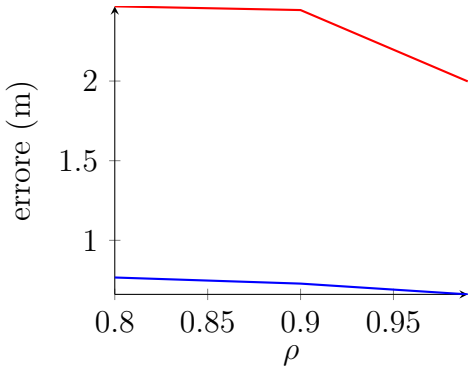


Figura 5.13: Risultati ottenuti con il data-set *Turtle* al variare del valore del parametro ρ, il raggio spettrale della matrice del reservoir.

(a) *I seguenti parametri* $a = 0.2$ $\lambda = 0.1$*(parametro di regolarizzazione)* $N_R = 50$*(dimensione del reservoir) sono fissati. I valori scelti per gli iper-parametri si basano sul modello selezionato automaticamente dal sistema in fase di validazione (per dettagli sui parametri vedere sezione 4.5.1).*

Rho	TrainErr	ValidErr
0.8	0.7666389	2.4690437
0.9	0.7282516	2.4457014
0.99	0.66069263	1.9979329

(b) *Grafico al crescere del valore del parametro ρ. In blu l'errore di training in rosso quello di validazione.*



Al crescere di ρ si mostra come si modifica l'errore al crescere della memoria e della non linearità del reservoir (vedere Sezione 4.4).
Sia nel data-set *Turtle* che nel data-set *Stella Maris* l'errore nella fase di apprendimento decresce al crescere di ρ, che deve rimanere minore di 1 per rispettare la ESP 2.2.1. Nella fase di validazione la funzione dell'errore ha un andamento simile e non si incorre nel rischio di overfitting. Anche nel caso di quest'iper-parametro il sistema implementato seleziona in modo automatico il modello migliore dal punto di vista dell'accuratezza.
Osservando e confrontando i risultati ottenuti al variare dei differenti iper-parametri il sistema riesce quindi a in modo automatico a scartare i modelli con alto rischio di overfitting e a selezionare tra i rimanenti il modello migliore in termini di accuratezza.

5.2.2 Risultati con l'Utilizzo del Comitato

Per misurare l'effettivo contributo del comitato si è scelto di procedere con una fase sperimentale dedicata allo scopo.
In particolare i passi seguiti per ricavare dei risultati significativi sono i seguenti.

1. É stata inizializzata una sola ESN ed il modello è stato scelto sulla base dei risultati di questa soltanto.

2. Una volta selezionato il modello sono state inizializzate altre 9 ESN differenti che sono state addestrate sulla base del modello selezionato.

3. Infine tutte e 10 le ESN sono state utilizzate per calcolare l'errore in test set con 2 diverse modalità.

   (a) La prima volta è stata calcolata la media degli errori delle singole ESN.

   (b) La seconda si è utilizzata la tecnica del comitato (vedere Sezione 4.2).

Le indicazioni che si possono trarre dai risultati avuti con i due data-set (Turtle e Stella Maris) sono simili, rendendo così tali osservazioni non strettamente dipendenti dal data-set utilizzato.
Dalle tabelle 5.1, 5.2 si può osservare un chiara riduzione dell'errore con l'impiego della tecnica del comitato rispetto all'utilizzo semplice della media degli errori. Quest'ultima ha però una deviazione standard relativa all'errore più bassa.

Tabella 5.1: Nel data-set *Turtle* la configurazione del modello che in fase di validazione ha ottenuto i migliori risultati è la seguente:
Reservoir Dim: 50, Regularization: 0.01, Leaky Param: 0.05, Rho: -1.0

(a) **Risultati in test Senza Comitato**. Deviazione Standard dell'errore in test 0,202723018 m.

Description	Value
MeanTestError on testSample 0	1.5806894
MeanTestError on testSample 1	1.3860795
MeanTestError on testSample 2	1.9160715
MeanTestError on testSample 3	1.7934967
Final average TestError	1.6690843

(b) **Risultati in test con il Comitato**. Deviazione Standard dell'errore in test 0,261967031 m.

Description	Value
MeanTestError on testSample 0	0.90993893
MeanTestError on testSample 1	0.90896356
MeanTestError on testSample 2	1.4681289
MeanTestError on testSample 3	1.3932807
Final average TestError	1.170078

Tabella 5.2: Nel data-set *Stella Maris* la configurazione del modello che in fase di validazione ha ottenuto i migliori risultati è la seguente:
Reservoir Dim: 200, Regularization: 0.1, Leaky Param: 0.05, Rho: 0.9

(a) **Risultati in test Senza Comitato**. Deviazione Standard dell'errore in test 0,066604768 m.

Description	Value
MeanTestError on testSample 0	0.5361649
MeanTestError on testSample 1	0.62519693
MeanTestError on testSample 2	0.4394557
MeanTestError on testSample 3	0.55903685
Final average TestError	0.5399636

(b) **Risultati in test con il Comitato**. Deviazione Standard dell'errore in test 0,079905525 m.

Description	Value
MeanTestError on testSample 0	0.47404382
MeanTestError on testSample 1	0.50622785
MeanTestError on testSample 2	0.2964483
MeanTestError on testSample 3	0.42161417
Final average TestError	0.42458355

Nei risultati si vede che per entrambi i data-set l'errore commesso in test con l'utilizzo della tecnica del comitato (anche considerando le deviazioni standard), è sensibilmente più basso. Come ci aspettavamo dunque il comitato comporta un vantaggio in termini di accuratezza.

5.2.3 Sperimentazione della Feature Selection e dell'Euristica ESNIGMA

Alla luce dei precedenti risultati ottenuti senza la selezione di feature si è deciso nei casi seguenti di usare anche la tecnica del comitato.
Le strategie di ricerca adottate per la selezione del sottoinsieme di feature sono due:

- con la selezione di feature tramite wrapper con la strategia di ricerca hill-climbing (Hill-climbing) (vedere Sezione 4.6).

- con la selezione di feature tramite wrapper adottando l'euristica ESNIGMA (ESNIGMA) (vedere Sezione 4.6.1).

I risultati riportati in Tabella 5.3, sono stati ottenuti rispettivamente con l'algoritmo a wrapper Hill-climbing (a) e il nuovo algoritmo ESNIGMA (b).

Tabella 5.3: Con il data-set *Stella Maris* le configurazioni dei modelli che in fase di validazione hanno ottenuto i migliori risultati sono le seguenti

(a) Risultati Con l'algoritmo **Hill-climbing**.
Deviazione Standard dell'errore in test 0,033129851 m.
I risultati sono stati ottenuti con le feature riconosciute dal seguente insieme di identificatori 0,2,3,4,6,7,8.
Tempo di completamento: 4857,843 s.
Reservoir Dim: 200, Regularization: 0.01, Leaky Param: 0.1, Rho: 0.99

Description	Value
MeanTestError on testSample 0	0.52052766
MeanTestError on testSample 1	0.44842377
MeanTestError on testSample 2	0.43301937
MeanTestError on testSample 3	0.4614707
Final average TestError	0.4658604

(b) Risultati Con l'algoritmo **ESNIGMA**.
Deviazione Standard dell'errore in test 0,087701578 m.
I risultati sono stati ottenuti con le feature riconosciute dal seguente insieme di identificatori 0,1,3,4,6;.
Tempo di completamento: 2963,481 s.
Reservoir Dim: 200, Regularization: 0.01, Leaky Param: 0.1, Rho: 0.99

Description	Value
MeanTestError on testSample 0	0.41164964
MeanTestError on testSample 1	0.44943386
MeanTestError on testSample 2	0.4639164
MeanTestError on testSample 3	0.6393528
Final average TestError	0.49108818

La differenza degli errori in test dei due comitati è di 0,02522778 m, alla luce delle deviazioni standard le prestazioni al livello dell'accuratezza sono da considerarsi paritarie. Quindi ciò che distingue il rendimento dei due algoritmi adottati sono i tempi di completamento e il sottoinsieme di feature selezionato.
L'algoritmo con l'euristica ESNIGMA impiega 1894,362 s in meno che corrisponde a circa il 39% in meno di tempo di calcolo rispetto all'algoritmo Hill-Climbing.
Riguardo l'insieme delle feature selezionate, non potendo dire a priori quali sono quelle più significative, si considera migliore il sottoinsieme con la cardinalità più bassa. Infatti ad un numero minore di feature corrisponde un numero minore di ancore con un conseguente risparmio sia energetico che nei costi di installazione del sistema. La cardinalità degli insiemi di feature selezionati è di 7 e 5 a vantaggio della strategia con euristica ESNIGMA.
Perciò l'algoritmo ESNIGMA ottiene delle prestazioni migliori sia dal punto di vista dei tempi di completamento che del sottoinsieme di feature selezionato.
La sperimentazione è proseguita con reservoir a 500 unità utilizzando soltanto l'algoritmo wrapper di selezione delle feature ESNIGMA. I risultati di addestramento sono sintetizzati nella Tabella 5.4, mentre la tabella 5.5 riporta i risultati ottenuti in fase di test. La Figura 5.14 riporta i grafici che mostrano

a confronto, per una delle sequenze di test, l'andamento del target e dell'output del modello di ESN selezionato e l'andamento della loro distanza ad ogni istante campionato. Come si può osservare l'errore in fase di test durante quasi tutto il percorso del robot resta ampiamente sotto il metro. Nella fase finale invece l'errore cresce notevolmente e probabilmente ciò è dovuto ad un disturbo creatosi nel segnale di input o nella rilevazione del target.

Tabella 5.4: **Con Feature Selection Tramite Wrapper ed euristica ESNIGMA.**
La configurazione del modello (id 43) che in fase di validazione ha ottenuto i migliori risultati è la seguente:
Reservoir Dim: 500, Regularization: 0.1, Leaky Param: 0.1, Rho: 0.99
Tempo di completamento: 18704,053 s.

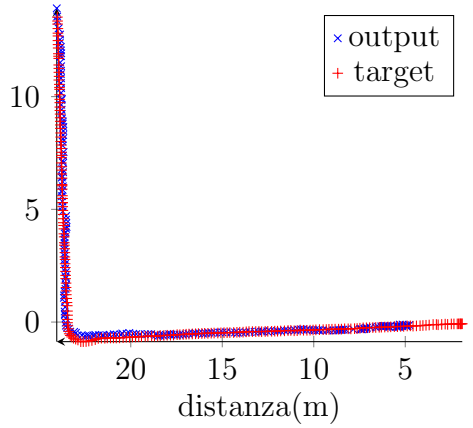
Description	Value
MeanValidationError on fold 0	0.59164935
MeanTrainingError on fold 0	0.108281836
MeanValidationError on fold 1	0.29342365
MeanTrainingError on fold 1	0.12729688
MeanValidationError on fold 2	0.34250915
MeanTrainingError on fold 2	0.124763764
MeanValidationError on fold 3	0.26930687
MeanTrainingError on fold 3	0.123816684
Final mean error in Training	0.12103979
Mean Error in Validation for ESN[0]	0.37422228
Final mean error in Validation	0.37422228

Tabella 5.5: I risultati in test sono stati ottenuti con le feature riconosciute dal seguente insieme di identificatori 0,1,2,4,5,6, ed hanno una Deviazione Standard dell'errore di 0,150197408 m

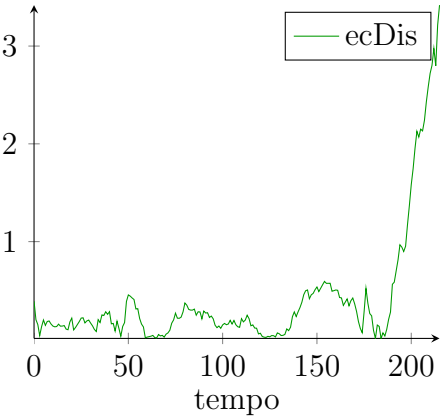
Description	Value
MeanTestError on testSample 0	0.41125673
MeanTestError on testSample 1	0.31097254
MeanTestError on testSample 2	0.29303798
MeanTestError on testSample 3	0.6693181
Final average TestError	0.42114633

Figura 5.14: Risultati sul test sample identificato con il numero 13(corrispondente al sample 0 nella Tabella 5.5)

(a) *grafico che confronta il target con l'output del comitato.*



(b) *Distanza euclidea/errore del comitato istante per istante.*



Confrontando i risultati ottenuti con le prestazioni raggiunte dal modello selezionato senza l'utilizzo di alcuna tecnica di selezione delle feature, i cui risultati sono riportati in Tabella 5.2(b), l'errore finale in test del modello di ESN selezionato con l'algoritmo ESNIGMA è minore. Si devono però considerare le rispettive deviazioni standard dell'errore nel test-set (riportate in Tabella 5.5 e 5.2(b)), che ci portano a considerare l'accuratezza dei modelli confrontati nello stesso intervallo di confidenza.
La differenza nel rendimento dei due modelli a confronto è quindi data stavolta dal numero di feature selezionate (per gli stessi motivi già sopracitati). Perciò il modello ottenuto con l'algoritmo ESNIGMA risulta più vantaggioso perché utilizza in tal caso soltanto 6 (vedere Tabella 5.5) feature invece che 10 (vedere Tabella 5.2(b)).
Si possono confrontare i risultati ottenuti con i modelli di ESN a 200 e 500 unità di dimensione del reservoir ottenuti con l'algoritmo di selezione delle feature ESNIGMA. Le prestazioni al livello dell'errore medio in test sono di 0.42m per il modello con reservoir a 500 unità, e di 0.49m per il modello con reservoir a 200 unità. La cardinalità degli insiemi di feature selezionati è di 5 nel caso del modello con un reservoir di 200 neuroni, e 6 nel caso del modello con un reservoir a 500 neuroni. In fine i tempi di completamento della fase di training sono di circa 2963s per il modello con reservoir di 200 unità, mentre

sono di circa 18704s per il modello con il reservoir da 500 unità. Il modello con reservoir a 500 unità ha un errore significativamente minore, ma utilizza più sensori e i tempi di addestramento per il modello sono notevolmente maggiori. Nel caso RUBICON quindi si può optare per la scelta del modello che utilizza meno sensori e con un reservoir composto da soltanto 200 unità.
Si è quindi ripetuta una fase di confronto degli algoritmi di selezione delle feature utilizzando il data-set *Turtle*. I risultati dell'algoritmo Hill-climbing sono riportati nella Tabella 5.6 per quanto riguarda l'addestramento e nella Tabella 5.7 per quanto riguarda il test. Mentre la figura 5.15 mostra a confronto l'andamento del target e dell'output del modello di ESN selezionato e l'andamento della loro distanza ad ogni istante campionato (per una delle sequenze di test).
I risultati dell'algoritmo ESNIGMA sono invece riportati nella Tabella 5.8 per quanto riguarda l'addestramento e nella Tabella 5.9 per quanto riguarda il test. Ancora una volta la figura 5.16 mostra a confronto l'andamento del target e dell'output del modello di ESN selezionato (stavolta dall'algoritmo ESNIGMA) e l'andamento della loro distanza ad ogni istante campionato (per una delle sequenze di test).
Dai grafici delle figure 5.15 e 5.16 si può vedere chiaramente come l'errore calcolato nel data-set *Turtle* sia maggiore di quello calcolato nel data-set *Stella Maris*. La differenza è dovuta al rumore nei dati di input ma anche in quelli del target che nel data-set *Turtle* sono molto meno accurati rispetto a quelli del data-set *Stella Maris*.

Tabella 5.6: **Risultati con Feature Selection tramite l'algoritmo Hill-Climbing.**
La configurazione del modello che in fase di validazione ha ottenuto i migliori risultati è la seguente:
Reservoir Dim: 50, Regularization: 0.001, Leaky Param: 0.2, Rho: 0.99
Tempo di completamento: 3427,729 s.

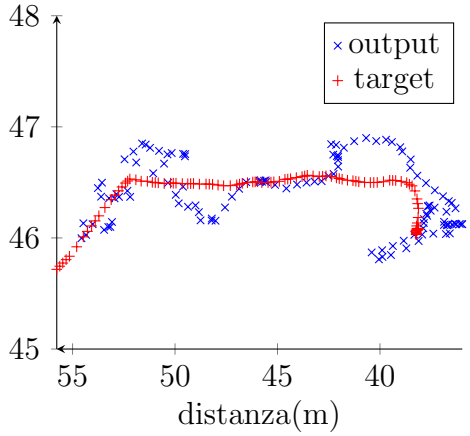
Description	Value
MeanValidationError on fold 0	1.7514515
MeanValidationError on fold 1	1.9256835
MeanValidationError on fold 2	1.2609441
MeanValidationError on fold 3	2.2002335
Mean Error in Validation for ESN[0]	1.7845781
Final mean error in Validation	1.7845781

Tabella 5.7: I risultati in test sono stati ottenuti con le feature riconosciute dal seguente insieme di identificatori 0,3,4, ed hanno una Deviazione Standard dell'errore di 0,410848758 m

Description	Value
MeanTestError on testSample 0	0.89394474
MeanTestError on testSample 1	1.5907916
MeanTestError on testSample 2	1.8522534
MeanTestError on testSample 3	1.9415338
Final average TestError	1.5696309

Figura 5.15: Risultati sul test sample identificato con il numero 0

(a) *grafico che confronta il target con l'output del comitato.*



(b) *Distanza euclidea/errore del comitato istante per istante.*

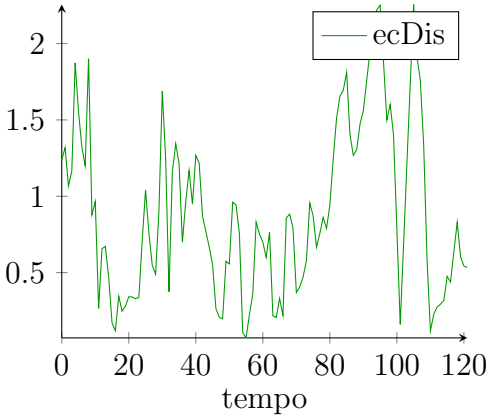


Tabella 5.8: **Risultati con Feature Selection tramite l'algoritmo ESNIGMA.**
La configurazione del modello che in fase di validazione ha ottenuto i migliori risultati è la seguente:
Reservoir Dim: 100, Regularization: 0.1, Leaky Param: 0.2, Rho: -1.0
Tempo di completamento: 2314,859 s.

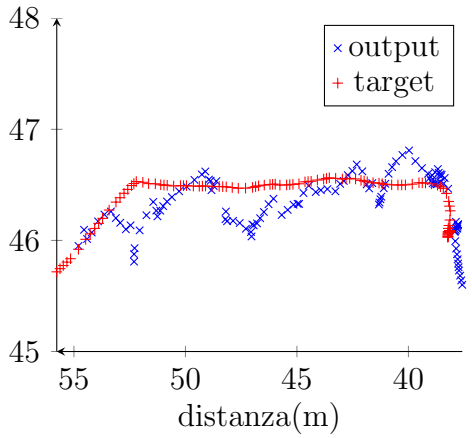
Description	Value
MeanValidationError on fold 0	1.8271086
MeanValidationError on fold 1	1.4430823
MeanValidationError on fold 2	1.8065003
MeanValidationError on fold 3	1.9562826
Mean Error in Validation for ESN[0]	1.7582434
Final mean error in Validation	1.7582434

Tabella 5.9: I risultati in test sono stati ottenuti con le feature riconosciute dal seguente insieme di identificatori 0,1,4, ed hanno una Deviazione Standard dell'errore di 0,504016731 m

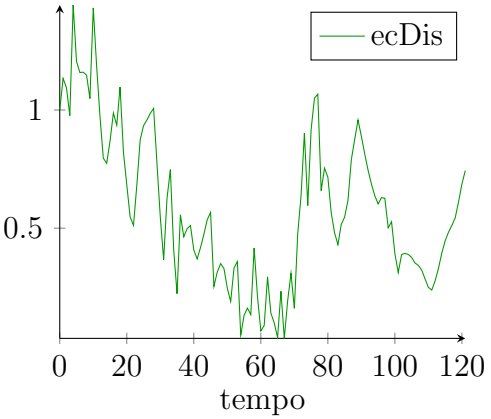
Description	Value
MeanTestError on testSample 0	0.5846246
MeanTestError on testSample 1	0.9531109
MeanTestError on testSample 2	1.9507512
MeanTestError on testSample 3	1.3134191
Final average TestError	1.2004764

Figura 5.16: Risultati sul test sample identificato con il numero 0

(a) *grafico che confronta il target con l'output del comitato.*



(b) *Distanza euclidea/errore del comitato istante per istante.*



Il modello selezionato a partire dagli stessi iper-parametri, ma senza una tecnica di feature selection, vedere tabella 5.1(b), ha come risultato un comitato di ESN che commette un errore sul test-set di 1,170078 m con una deviazione standard di 0,261967031 m.
Gli ultimi risultati riportati hanno un errore che considerando l'alta deviazione standard resta simile. Ma a vantaggio dei metodi di wrapper feature selection il numero di feature si abbassa da 5 a 3. Anche questa volta il metodo ESNIGMA wrapper impiega un tempo di calcolo minore rispetto all'Hill-climbing classico, con un risparmio di circa il 33%. Inoltre anche se considerando la deviazione standard i due errori commessi dai comitati di ESN

trovati con i wrapper sono simili, l'errore sul test-set di ESNIGMA wrapper è più basso. É dunque quest'ultimo il modello preferibile.

Capitolo 6

Conclusioni e Sviluppi Futuri

Nel contesto delle ecologie robotiche del progetto europeo RUBICON vi era l'esigenza di sviluppare una serie di funzionalità che garantissero all'ecologia le capacità di gestire in maniera autonoma l'acquisizione dinamica di nuovi compiti di apprendimento.
Il lavoro di tesi svolto contribuisce all'obiettivo del progetto RUBICON tramite lo sviluppo di due componenti all'interno del modulo Learning Layer. É stato sviluppato un sistema che permette l'automatizzazione delle procedure di apprendimento e tramite la tecnica di cross-fold validation ricava dei valori di valutazione per tutti i modelli considerati. Questi modelli sono determinati da una serie di iper-parametri (per ognuno dei quali è stabilito un insieme di valori) che definiscono tutte le possibili ESN che il sistema può generare. La procedura basandosi così sui valori di valutazione dei modelli ha consentito l'implementazione di un approccio di selezione automatica del modello teso a perfezionare la performance predittiva.
Il procedimento sviluppato è stato validato sperimentalmente dai risultati nel Capitolo 5 Sezione 5.2.1, che dimostrano come la procedura riesca tramite la tecnica di cross-fold validation a selezionare in modo autonomo il modello più accurato.
L'altra componente è costituita da un sistema di feature selection supervisionata specifico per ESN e serie temporali. Questa componente è stata implementata per selezionare gli attributi del segnale in ingresso maggiormente rilevanti, al fine di consentire l'ottimizzazione dei tempi di comunicazione e potenzialmente a perfezionare le performance predittive del sistema. La procedura di feature selection è stata integrata nel sistema di selezione del modello,

in particolare il risultato dell'intero sistema (con questa componente integrata) è ancora un modello di reti ESN ma che funziona con un sottoinsieme delle feature dell'insieme originale.
La tesi prone un approccio originale alla selezione di feature specifico per le ESN e le sequenze di dati, tramite lo sviluppo di un algoritmo innovativo detto ESNIGMA, e confrontato con un algoritmo implementato detto Hill-Climbing.
Come dimostrano i dati sperimentali riportati nel Capitolo 5 Sezione 5.2.3, con entrambi gli algoritmi, Hill-Climbing ed ESNIGMA, il sistema riesce a trovare un modello con un sottoinsieme di feature dell'insieme originario. L'accuratezza dei modelli risultanti è paritaria, sia con l'algoritmo Hill-Climbing che con ESNIGMA che con il modello che utilizza tutte le feature. In particolare però il nuovo algoritmo sviluppato ESNIGMA, riesce a trovare modelli anche con sottoinsiemi di cardinalità minore rispetto a quelli trovati da HillClimbing, ed inoltre i tempi di completamento di Hill-Climbing sono maggiori di circa il 50% in confronto con i tempi di completamento di ESNIGMA.

Durante l'implementazione delle componenti principali sono state aggiunte due importanti funzionalità. Una riguarda l'introduzione della tecnica del comitato e l'altra il rescaling dei pesi nelle matrici del reservoir (vedere Capitolo 4 per ulteriori dettagli).
I risultati riportati nel Capitolo 5 Sezione 5.2.2, dimostrano che la tecnica del comitato sviluppata apporta notevoli miglioramenti nell'accuratezza dei risultati del sistema.
Il rescaling dei pesi del reservoir come commentato nelle Sezioni 2.2.1,2.2.2,4.4, era necessario per mantenere la correttezza delle proprietà delle ESN. Inoltre si può osservare dai risultati riportati nel capitolo 5 che gli esiti migliori dal punto di vista dell'accuratezza finale delle reti ESN, si hanno come atteso per valori di ρ (raggio spettrale della matrice del reservoir, vedere Sezioni 2.2.1,2.2.2,4.4) minori e vicini ad uno.
Inoltre è stata prodotta una versione minimale del software indipendente dal sistema RUBICON, che ha preso il nome di SoloESN, al fine di fornire uno strumento per:

1. L'utilizzo in fase sperimentale ed off-line con diversi tipi di data-set.

2. Il risparmio di memoria e tempo computazionale dovuto all'esecuzione

di thread non strettamente connessi all'apprendimento.

3. La generazione di un modello di ESN che computa sequenze di dati non strettamente legate al contesto RUBICON.

Riguardo gli ulteriori sviluppi al software possiamo considerare diverse soluzioni.
Oltre alle procedure di selezione delle feature verificate sperimentalmente nella Sezione 5.2.3 (e descritte nella Sezione 4.6), il codice sviluppato contiene altre due procedure. Queste sono speculari a quelle utilizzate nella fase sperimentale ma aggiungono la possibilità di fare back-tracking (vedere Sezione 3.3).Queste procedure devono però ancora essere validate sperimentalmente, potrebbero perciò essere necessarie ulteriori modifiche al codice per correggerne il comportamento.
Oltre a quanto appena su detto, si può migliorare ulteriormente l'applicazione SoloESN (vedere Sezione 4.1). Quest'applicazione, è stata realizzata dal codice del software RUBICON. Nell'architettura dell'ecologia robotica (come visto in Sezione 2.3) e nel Learning Layer (vedere Sezione 2.3.2) vi sono molte componenti che operano in parallelo, una di queste è il codice estratto. Il codice di SoloESN (estratto dal software RUBICON) è sequenziale ma per le sue caratteristiche si presterebbe ad una implementazione parallela. La riscrittura in modo parallelo di questo codice migliorerebbe notevolmente i tempi necessari al completamento del apprendimento delle ESN.
Altri sviluppi futuri potrebbero includere l'aggiunta di nuove euristiche sia nella fase di selezione del modello che delle Feature. Tali euristiche potrebbero migliorare:

- nella fase di selezione del modello, i tempi computazionali;
- nella fase di selezione delle Feature, sia i tempi di completamento che l'accuratezza finale del modello.

Nel primo caso, ad esempio, si potrebbe cercare di capire l'andamento dell'errore di validazione all'aumentare del numero di neuroni del reservoir. In tal modo si può interrompere prematuramente la cross-fold validation qualora l'andamento sia crescente.
Nel secondo caso invece si potrebbero introdurre i metodi *Oscillating* (vedere Sezione 3.3) i quali potrebbero apportare un miglioramento nell'accuratezza

del modello finale, risultato dell'applicazione.
Si potrebbe anche considerare se e quanto l'aggiunta o la rimozione di una feature influisca in modo positivo o negativo sull'errore di validazione. In tal modo si può velocizzare la fase di selezione delle feature. Se infatti la rimozione di una feature porta ad un notevole peggioramento dell'errore allora questa (qualora venisse reintrodotta) sarà scelta per ultima in una fase successiva.
Un altro possibile sviluppo futuro riguarda la possibilità di integrare al software altri tipi di reti neurali (o persino ulteriori metodi) diversi dalle ESN.
Per concludere è da considerare che grazie alla struttura del software le nuove soluzioni sono facilmente integrabili con modifiche puntuali allo stesso.
Ad esempio l'aggiunta di nuove tipologie di reti neurali o nuovi metodi di apprendimento, sono integrabili con la sola aggiunta di una nuova classe per ogni nuovo metodo (senza la necessità di introdurre nuove modifiche nel resto del codice).

Appendice A

Manuale d'uso

É stata realizzata un applicazione separata dal progetto RUBICON che fornisce le sole funzionalità legate all'apprendimento. Questo manuale d'uso è riferito a quest'applicazione. Lo scopo di quest'applicazione è di poter addestrare delle Echo State Network in modo indipendente dal sistema RUBICON. Le informazioni riportate restano comunque valide anche nell'ambito del progetto RUBICON, dove quest'applicazione è stata utilizzata anche per la fase sperimentale dell'apprendimento per ESN.

A.1 Utilizzo dell'applicazione

Per utilizzare l'applicazione è necessario avere installato nella macchina che la esegue la versione 1.7 della Java Virtual Machine (JVM). Il codice è preimpostato per eseguire una serie di dataset conosciuti. Per poter eseguire il codice con nuovi dataset è necessario modificare lo script *completeScriptedComputation* (vedere Sezione A.1.1) nella classe *ESNmain*, in modo da configurare un esecuzione adatta allo specifico caso. L'eseguibile (oggetto della compilazione) accetta in ingresso 4 diverse configurazioni di parametri.

1. **senza parametri**: viene eseguito lo script *completeScriptedComputation* (vedere Sezione A.1.1) facente parte della classe principale

2. **con un parametro**: il parametro indica con quale dataset si vuole eseguire l'addestramento. Anche in questo caso viene invocato lo script *completeScriptedComputation* (vedere Sezione A.1.1) ma stavolta è specificato il dataset.

3. **con 2 parametri**: il primo parametro indica il path contenente la cartella del dataset che si vuole eseguire, il secondo parametro indica il percorso della cartella nella quale si vuole conservare l'output del risultato

4. **con 8 parametri**: i primi due parametri sono trattati come nel caso precedente. Gli altri parametri in ordine sono i seguenti.

   - **Reinitstate** è un booleano, se il suo valore è *t* durante il calcolo dell'errore lo stato delle ESN viene azzerato tra una sequenza e l'altra.
   - **Onlysimplecomp** è un booleano, se il suo valore è *t* allora l'apprendimento è eseguito su una serie di ESN dalle dimensioni più piccole, altrimenti su una serie di ESN più grandi.
   - **Ensemble** è un booleano, nel caso in cui il valore sia *t* viene utilizzata la tecnica del comitato altrimenti si avvia un esecuzione senza tale tecnica.
   - **NoramlizationAll** è un booleano, se il valore è *t* allora i dati di input sono normalizzati in base ai valori dei dati di tutto il data-set, altrimenti, gli stessi dati di input vengono normalizzati soltanto in base al valore del data-set di addestramento (*training*).
   - **TypeOfExecution** è una stringa, indica la modalità dell'apprendimento che si ha intenzione di eseguire. Le opzioni possibili sono le otto seguenti.
     (a) **woga**: attua l'apprendimento con il wrapper, sfruttando l'euristica ESNIGMA. La selezione del sottoinsieme è fatta con una versione greedy dell'algoritmo hill-climbing.
     (b) **wogh**: effettua l'apprendimento con il wrapper (senza nessuna euristica). La selezione del sottoinsieme è fatta con una versione greedy dell'algoritmo hill-climbing.
     (c) **woba**: esegue l'apprendimento con il wrapper, sfruttando l'euristica ESNIGMA. La selezione del sottoinsieme è fatta tramite un algoritmo del tipo hill-climbing con backtraking.
     (d) **wobh**: lancia l'apprendimento con il wrapper (senza nessuna euristica). La selezione del sottoinsieme è fatta tramite un algoritmo del tipo hill-climbing con backtraking.

(e) **woa**: attua l'apprendimento con il wrapper. La selezione del sottoinsieme è fatta tramite una ricerca esaustiva, ovvero viene fatto l'addestramento con ogni possibile sottoinsieme di input.

(f) **s**: effettua l'apprendimento in modalità semplice, cioè senza wrapper.

(g) **dext**: Modifica il data-set iniziale con nuovi sequenze di dati ottenute aggiungendo del rumore a quelle originali. Lancia l'apprendimento utilizzando il nuovo data-set espanso.

(h) **all**: Inizia con l'eseguire l'apprendimento da prima in modalità semplice e poi come nel caso precedente, utilizzando in entrambi i casi lo stesso pool di ESN non addestrate.

- **WhichConfigDataSet** è una stringa, indica per quale data-set si vuole eseguire l'applicazione. Vi sono dieci differenti opzioni di configurazione delle impostazioni per nove differenti data-set. Le opzioni possibili sono le seguenti.

  (a) **k**: seleziona il data-set detto *kitchen*, la funzione dell'errore è data dalla distanza euclidea dal target di regressione.

  (b) **kc**: seleziona ancora il data-set chiamato *kitchen*, ma la funzione dell'errore è data dall'inverso dell'accuratezza calcolata sul target di classificazione.

  (c) **lt**: seleziona il data-set dei dati raccolti nel dipartimento di informatica con il robot *turtle*.

  (d) **sm**: seleziona il data-set dei dati raccolti durante la fase sperimentale alla *Stella Maris*.

  (e) **smd**: seleziona il data-set dei dati raccolti soltanto durante il giorno durante la fase sperimentale alla *Stella Maris*.

  (f) **smn**: seleziona il data-set dei dati raccolti soltanto durante la notte durante la fase sperimentale alla *Stella Maris*.

  (g) **ex**: seleziona il data-set detto *Exercising*.

  (h) **ab**: seleziona il data-set chiamato *Rehabilitation*.

  (i) **rx**: seleziona il data-set detto *Relax*.

  (j) **sl**: seleziona il data-set chiamato *Sleeping*.

  I dataset *turtle* e *Stella Maris* sono di regressione. Gli ultimi quattro data-set sono di classificazione.

A.1.1 Lo script completeScriptedComputation

Il seguente script serve per l'esecuzione manuale dell'apprendimento così da poter specificare tutte le possibili opzioni. É particolarmente utile anche per provare l'apprendimento su data-set non noti dalle configurazioni previste. Lo script accetta due parametri.
Il primo supervisor è il riferimento alla classe che contiene il metodo che presi i parametri di input esegue i metodi per la creazione dell'ambiente di apprendimento, e quindi lo avvia.
Il secondo whichConfigDataSet serve a specificare per quale data-set si fa apprendimento, può anche essere nullo, nel qual caso il valore viene specificato a mano nello script.
La terza opzione è che whichConfigDataSet abbia il valore *manual* nel qual caso si specificano tutti i parametri per la configurazione dell'applicazione su un determinato data-set non noto.
Nel Listing seguente A.1 si riporta il codice dello script con i relativi commenti per la comprensione delle diverse variabili che compongono il ciclo dell'apprendimento.

Listing A.1: Script per l'esecuzione dell'apprendimento

```
1  private static void completeScriptedComputation(SupervisorEntity
       supervisor, String whichConfigDataSet){
     //directory were are the directories of data-sets that we want use
3    String dataDirectory = "/home/ilpibens/Universita/Tesi/dataset/";
     //directory of output
5    String outputDirectory = "/home/ilpibens/Universita/Tesi/Loc/forTest/";
     //true if want reinitialize the state of ESN at the end of each validation
         sample, false otherwise
7    boolean reinitState = false;
     //true if don't want compute for 500 units of reservoir, false otherwise
9    boolean onlySimpleComputation = true;
     //true if we want use the ensemble method to compute the output, false
         otherwise
11   boolean ensemble = false;
     //true if we want normalize the entire dataset on all data (test dataset
         included),
13   //false otherwise (normalize only on data in training dataset)
     boolean normalizationAll= false;
```

```
15      /*TypeOfExecution
         * if 'woga' then execute only wrapper type greedy aesnigma
17       * if 'wogh' then execute only wrapper type greedy hillclimbing
         * if 'woba' then execute only wrapper type backtraking aesnigma
19       * if 'wobh' then execute only wrapper type backtraking hillclimbing
         * if 'woa' then execute only wrapper type with euxastive reserch
21       * if 'all' then execute all type
         * if 's' then execute only simple cross-fold validation
23       * if 'dext' then execute only with adding random gaussian noise to data to
              have a more large dataset*/
        TypeOfExecution exect = TypeOfExecution.s;
25      /*ONLY if TypeOfExecution is 's' and ensemble false than if
            ModelSelectionWithOneEcho is true than do model selection with only
            one ESN.
         * But after that model is selected initialize a new pool of ESNs and train
              these with model selected.
27       * In the end test all ESN with ensemble and without.*/
        boolean ModelSelectionWithOneEcho=false;
29      /*specify for which data set want compute the training
         * if "k" -> Kitchen treated how regression task
31       * if "kc" -> Kitchen treated how classification task
         * if "lt" -> Turtle
33       * if "sm" -> all data samples collected at Stella Maris
         * if "smn" -> data samples collected at Stella Maris only in night time
35       * if "smd" -> data samples collected at Stella Maris only in day time
         * if "ex" -> Exercising
37       * if "ab" -> Rehabilitation
         * if "rx" -> Relax
39       * if "sl" -> Sleeping*/
        if(whichConfigDataSet==null) whichConfigDataSet = "manual";
41      if(whichConfigDataSet.equals("manual")){
            //folder name of the dataset
43          String dataSetName = "Rehabilitation/";
            //number of sample, we have one sample for each file
45          int numberOfSamples=50;
            //output dimensionality
47          int numberOfTarget=1;
            /* numberOfData: is the number of different feature of input
```

```
49          * numberOfSource: is the number of source of the input (example how
                many mote in the localization task)
            * in general the input dimensionality is the numberOfData*
                numberOfSource*/
51          int numberOfData=7; int numberOfSource=1;
            /* if the input include how frist column the time stamp or other initial
                column that we do not want treat
53          * as input, how of these coumn are in the file*/
            int numberOfIndesideredColumn=0;
55          // true if the index of test are static, false otherwise
            boolean staticIndexOfTest = false;
57          // if index of test are static indicate here which
            int[] indexOfTest = new int[]{};
59          //indexOfTest.length
            int howManySampleForTest=13;
61          //if the type of task is classification here we specify the classification
                range with 2 float
            float[] classificationRange= new float[]{-1f,1f};
63          //for wiring instruction in this level there isn't important
            String[] transducerString= new String[]{"in1","in2","in3","in4","in5","
                in6","in7"};//"rrs1","rss5","rss2","rss4","rss3"};,"in8","in9","in10
                ","in11","in12","in13","in14","in15"};
65          //type of task (regeression or classification). To day function only with
                binary classification task.
            String taskType = "classification";
67          //can be sequence-to-sequence or sequence-to-element, indicate the
                output frequency
            String taskGranularity = "sequence-to-sequence";
69          //Can be sensory (input data come from the sensor transducers) or
                event (input/output data couples refer to event)
            String taskDataType = "sensory";
71          //type of associated predictive output (weights, event classification/
                prediction or sensor prediction)
            String taskOutputType = "sensor";
73          /* Specify how loss function use:
            * if 'ed' use Euclidean distance
75          * if 'mae' use mean absolute error*/
            String typeOfLossFunction = "ed";
```

```
77          supervisor.trainingTask(dataDirectory, outputDirectory, reinitState,
                onlySimpleComputation, ensemble, normalizationAll, exect,
                whichConfigDataSet,
                  true, ModelSelectionWithOneEcho,
79                dataSetName, numberOfSamples, numberOfData,
                  numberOfIndesideredColumn, numberOfTarget, numberOfSource,
81                staticIndexOfTest, indexOfTest, howManySampleForTest,
                  classificationRange,
83                transducerString,
                  taskType, taskGranularity,
85                taskDataType, taskOutputType,
                  typeOfLossFunction);
87      }else
          supervisor.trainingTask(dataDirectory, outputDirectory, reinitState,
              onlySimpleComputation, ensemble, normalizationAll, exect,
              whichConfigDataSet, ModelSelectionWithOneEcho);
89  }
```

A.2 Descrizione dell'output

I risultati prodotti dall'esecuzione dell'applicazione sono raccolti in molteplici file all'interno della cartella di output specificata al momento dell'avvio. I file prodotti sono raccolti in diverse cartelle in base al tipo di contenuto e al tipo di esecuzione. Tra le diverse opzioni c'è la possibilità di lanciare l'apprendimento utilizzando un comitato oppure creare altri esempi di sequenze di training da quelli esistenti distorcendo i dati originali con del rumore gaussiano.
Se si sceglie di utilizzare il comitato (Sezione 4.2) viene creata una sotto-cartella, dal nome *WithEnsamble/*, che sarà utilizzata come cartella principale in cui verranno create tutte le cartelle e tutti i file di output, se si sceglie l'opzione di aggiungere nuovi dati da quelli esistenti invece questa sotto-cartella prende il nome di *WithDistortion/*.
Nella cartella principale si trovano i seguenti file:

1. *Accuracy.txt* file testuale che riporta i valori degli errori medi di validazione per ogni modello e l'errore medio sui campioni del test-set.
   Ogni riga del file riguardante gli errori di validazione ha la seguente forma:

> # ESN <identificatore del modello> with ResDim= XX ReadReg= XX leakyPar= XX has an average error on the Validation samples of -> XX m

Dove <identificatore del modello> è un intero crescente a partire da zero che viene assegnato ad ogni configurazione degli iper-parametri, mentre XX è il valore relativo alla descrizione che lo precede.

(a) ResDim dimensione del reservoir N_R (vedere Sezione 4.5.1).

(b) ReadReg valore del parametro di regolarizzazione del readout λ (vedere Sezione 4.5.1).

(c) leakyPar valore del parametro leaky integrator a (vedere Sezione 4.5.1).

Le righe riguardanti i risultati sul test-set hanno la seguente forma:

> # ESN <identificatore del modello> has an average error on a sample of test of -> XXm

L'ultima riga ha la seguente forma:

> and on average has an error of -> XX m

2. *AccuracyOfReTraining.txt* analogo al file *Accuracy.txt* ma di seguito agli errori di validazione sono riportati gli errori delle ESNs ri-addestrate su tutto il data-set (compreso il test-set).

3. *FoldResult.txt* file testuale contiene per ogni modello valutato la seguente frase:

> Il risultato per il modello con id <identificatore del modello> è dato dai seguenti indici: <elenco degli identificatori delle feature selezionate>;
> l'esecuzione del cross-fold validation con <modalità di esecuzione>

ha ottenuto un errore di XX in Validation
impiegandoci XXms

Dove <modalità di esecuzione> è una delle seguenti stringhe: ESNIGMAGreedyVersion, HillClimbingAlg, HillClimbingWithBackTrack, ESNIGMAWithBackTrack o AllCombinationGenerator.

4. *TestResultsForPlot.txt* file testuale contenente i dettagli del modello selezionato
   (a) *Id*, identificatore del modello.
   (b) *ResD*, dimensione del reservoir N_R (vedere Sezione 4.5.1).
   (c) *Reg*, valore del parametro di regolarizzazione λ (vedere Sezione 4.5.1).
   (d) *Leaky*, valore del parametro leaky integrator a (vedere Sezione 4.5.1).
   (e) *Rho*, valore del parametro rho ρ (vedere Sezione 4.5.1).
   (f) *TestErr*, errore calcolato sul test-set
   (g) *SelFeatureId*, (opzionale presente soltanto nei casi in cui si stia facendo uso del wrapper) elenco degli identificatori delle feature selezionate.

Il resto dei risultati sono quindi divisi tra 5 sotto-cartelle di quella principale:

1. **ESNs** contiene i file delle ESNs addestrate. Il nome dei file è *esn*<id>*.ser*. La parte <id> è un identificatore crescente calcolato come <identificatore del modello> $*$ <numero totale di fold utilizzate> + <identificatore della fold>.
   Dove <identificatore della fold> è un intero crescente a partire da 0 che serve ad identificare quale partizione dell'insieme di training tra le K partizioni della cross validation (vedere Sezione 4.5.1) viene utilizzato come insieme di validazione.

2. **Tables** contiene principalmente i file delle tabelle in formato LaTeX che riassumono i risultati per ogni modello. In ogni tabella la prima colonna

Description indica a che cosa si riferisce il valore nella seconda colonna *Value*. La nomenclatura dei file è *ResultConfigTable*<id>*.tex*.
La porzione <id> è l'identificatore del modello (cioè della configurazione degli iper-parametri usati).
In questa cartella si trovano altri due file, *SummaryConfigTable.tex* e *ResultsForPlot.txt*, contengo gli stessi dati in forma tabellare rispettivamente nel formato LaTeX e in un formato testuale.
Le colonne in ordine sono:

(a) *Id*, identificatore del modello.

(b) *ResD*, dimensione del reservoir N_R (vedere Sezione 4.5.1).

(c) *Reg*, valore del parametro di regolarizzazione λ (vedere Sezione 4.5.1).

(d) *Leaky*, valore del parametro leaky integrator a (vedere Sezione 4.5.1).

(e) *Rho*, valore del parametro rho ρ (vedere Sezione 4.5.1).

(f) *TrainErr*, errore commesso sugli esempi di addestramento.

(g) *ValidErr*, errore commesso sugli esempi di validazione.

(h) *SelFeatureId*, (opzionale presente soltanto nei casi in cui si stia facendo uso del wrapper) elenco degli identificatori delle feature selezionate.

(i) *Time*, tempo impiegato per completare la procedura di addestramento (incluso il tempo della selezione delle feature).

3. **Variables** contiene i file di variabili, in formato LaTeX, che servono per memorizzare l'identificatore assoluto di una determinata sequenza di input. La nomenclatura dei file è *AbsIdTraining*<id>*.tex*.
La sezione <id> è un identificatore ottenuto giustapponendo <identificatore della fold>·<identificatore della sequenza in quella fold>.

4. **TrainingAndValidation** contiene una serie file che racchiudono i dati di input e relativi output in formato tabellare. Per ogni file le righe rappresentano ogni istante di tempo di una sequenza di input usata come training o validation. Le colonne descrivono nell'ordine: le feature di input, l'output, il target e l'errore.

Il nome dei file può essere *Training*<id>*.txt* oppure *Validation*<id>*.txt*, rispettivamente se contiene i risultati ottenuti sull'insieme di addestramento o validazione.
La porzione <id> è un identificatore ottenuto giustapponendo <identificatore del modello>·<identificatore della fold>·<identificatore della sequenza relativa in quella fold>.

5. **TestAfterValidation** contiene molteplici file sia di tipo LaTeX che testuale.

   (a) *Test*<id>*.txt*. Sono file equivalenti a quelli di *TrainingAndValidation* ma per le sequenze facenti parte del test-set.
   La sezione <id> è un identificatore ottenuto giustapponendo <identificatore della configurazione del modello>·<identificatore della sequenza>.

   (b) *AbsIdreTrainingPerformance*<id>*.tex*. Contiene una variabile in formato LaTeX per memorizzare l'identificatore assoluto di una determinata sequenza di input.
   La parte <id> è un identificatore ottenuto giustapponendo <identificatore del modello scelto per il test>·<identificatore della sequenza di input nel data-set>.

   (c) *AbsIdTest*<id>*.tex*. Contiene una variabile in formato LaTeX per memorizzare l'identificatore assoluto di una determinata sequenza di input.
   La porzione <id> è un identificatore ottenuto giustapponendo <identificatore del modello scelto per il test>·<identificatore della sequenza di input nel test-set>.

   (d) *ResultTestTable.tex*. File con una tabella in formato LaTeX che riassume i risultati del modello sul test-set. La prima colonna *Description* indica a che cosa si riferisce il valore nella seconda colonna *Value*.

   (e) *ResultReTrainingTable.tex*. Questo file contiene risultati analoghi al file ResultTestTable.tex ma relativi al ri-addestramento su tutto il data-set.

   (f) *MatrixConfusionTable.tex*. File in formato LaTeX della matrice di confusione relativa.

(g) *Accuracy.tex*. Contiene il valore di una variabile in formato LaTeX che rappresenta l'accuratezza.

(h) *Precision.tex*. Contiene il valore di una variabile in formato LaTeX che rappresenta la precisione.

(i) *Sensitivity.tex*. Contiene il valore di una variabile in formato LaTeX che rappresenta la sensitività.

(j) *Specificity.tex*. Contiene il valore di una variabile in formato LaTeX che rappresenta la specificità.

(k) *reTrainingPerformance*<id>*.txt*. Sono file equivalenti a quelli di *TrainingAndValidation* ma sono il riepilogo dei risultati di training durante il ri-addestramento delle ESNs relative al modello scelto su tutto il data-set (incluso test set).

Appendice B

Dettaglio dei risultati Sperimentali

B.1 Analisi Preliminare

Alcuni aspetti che volevamo verificare durante la costruzione del sistema rigurdano il dataset turtle. Gli esperimenti sono stati condotti per capire quanto fosse possibile migliorare l'apprendimento su dati di questo genere e se ci fossero configurazioni del modello ESN migliori, tramite l'esplorazione fatta con la cross-validation (CV) sugli iperparamteri. Sono state condotte 8 esecuzioni sulla base di tre differenti parametri legati al preprocessing utili in questa fase di fattibilità. Lo scopo è stato comprendere quanto fossero rilevanti questi parametri nell'apprendimento e nelle prestazioni.
I tre parametri sono i seguenti.

1. *Il comitato* (vedere Sezione 4.2) se utilizzare le uscite delle 5 ESN per fare comitato e calcolare su questo l'errore oppure mediare l'errore che queste commettono.

2. *La normalizzazione* i dati utilizzati vengono normalizzati, in base ai soli dati del training set o in base a tutti i dati disponibili compreso quindi il test set.

3. *L'azzeramento dello stato del reservoir* alla fine del calcolo dell'errore su ogni validation sample, lo stato del reservoir può essere lasciato quello attuale oppure viene resettato e ripristinato al valore iniziale 0.

Le 8 esecuzioni riassumono tutte le combinazioni possibili di queste opzioni.

Tabella B.1: **Senza azzeramento dello stato del reservoir, senza comitato, e con normalizzazione solo sul training**.

La configurazione del modello (id 33) che in fase di validazione ha ottenuto i migliori risultati è la seguente:

(a) **Reservoir Dim: 50, Regularization: 0.1, Leaky Param: 0.5, Rho: -1.0**.

Description	Value
MeanValidationError on fold 0	1.8155854
MeanValidationError on fold 1	2.0727894
MeanValidationError on fold 2	2.5706573
MeanValidationError on fold 3	3.326797
Mean Error in Validation for ESN[0]	2.62461
Mean Error in Validation for ESN[1]	2.4829564
Mean Error in Validation for ESN[2]	2.524444
Mean Error in Validation for ESN[3]	2.3195605
Mean Error in Validation for ESN[4]	2.2807155
Final mean error in Validation	2.4464574

(b) **Result in Test on the data-set LocTurtle/**.

Description	Value
MeanTestError on testSample 0	1.1665668
MeanTestError on testSample 1	1.2927263
MeanTestError on testSample 2	2.0083296
MeanTestError on testSample 3	1.7327526
Final average TestError	1.5500939

Deviazione Standard dell'errore in test 0,337888767 m

Tabella B.2: **Senza azzeramento dello stato del reservoir, senza comitato, e con normalizzazione su tutto il data-set**.

La configurazione del modello (id 135) che in fase di validazione ha ottenuto i migliori risultati è la seguente:

(a) **Reservoir Dim: 200, Regularization: 1.0, Leaky Param: 0.1, Rho: 0.9**.

Description	Value
MeanValidationError on fold 0	1.4455359
MeanValidationError on fold 1	2.2559466
MeanValidationError on fold 2	2.1363456
MeanValidationError on fold 3	1.7783178
Mean Error in Validation for ESN[0]	1.8076496
Mean Error in Validation for ESN[1]	1.7010868
Mean Error in Validation for ESN[2]	1.9171865
Mean Error in Validation for ESN[3]	2.070169
Mean Error in Validation for ESN[4]	2.024091
Final mean error in Validation	1.9040365

(b) **Result in Test on the data-set LocTurtle/**.

Description	Value
MeanTestError on testSample 0	0.7567145
MeanTestError on testSample 1	1.3732916
MeanTestError on testSample 2	1.3023757
MeanTestError on testSample 3	1.2404892
Final average TestError	1.1682177

Deviazione Standard dell'errore in test 0,242183673 m

Tabella B.3: **Senza azzeramento dello stato del reservoir, con comitato, e con normalizzazione solo sul training**.

La configurazione del modello (id 31) che in fase di validazione ha ottenuto i migliori risultati è la seguente:

(a) **Reservoir Dim: 50, Regularization: 0.1, Leaky Param: 0.2, Rho: 0.99**.

Description	Value
MeanValidationError on fold 0	1.3981106
MeanValidationError on fold 1	1.8466268
MeanValidationError on fold 2	2.2393193
MeanValidationError on fold 3	2.5076754
Final mean error in Validation	1.9979329

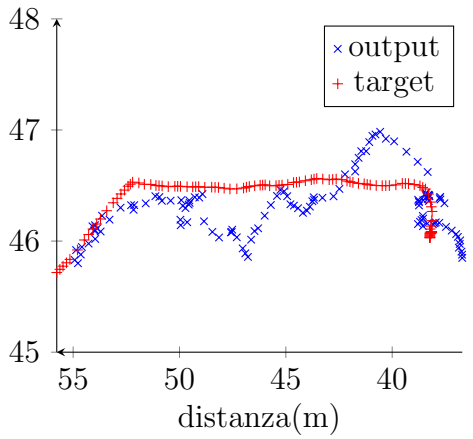
(b) **Result in Test on the data-set LocTurtle/.**

Description	Value
MeanTestError on testSample 0	0.67532563
MeanTestError on testSample 1	0.9129906
MeanTestError on testSample 2	2.0372546
MeanTestError on testSample 3	1.5193007
Final average TestError	1.2862179

Deviazione Standard dell'errore in test 0,531712258 m

Figura B.1: Risultati sul test sample 0

(a) *grafico che confronta il target con l'output del comitato.*



(b) *Distanza euclidea/errore del comitato istante per istante.*

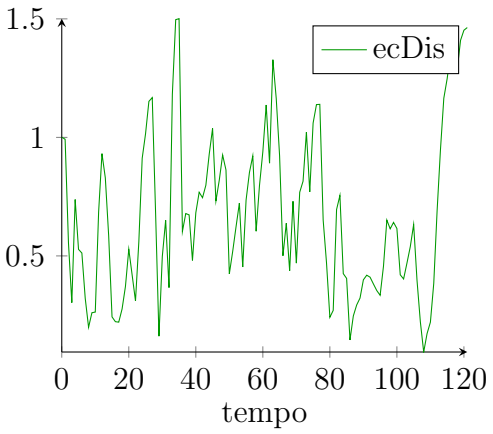


Tabella B.4: **azzeramento dello stato del reservoir, con comitato, e con normalizzazione su tutto il data-set**.

La configurazione del modello (id 123) che in fase di validazione ha ottenuto i migliori risultati è la seguente:

(a) **Reservoir Dim: 200, Regularization: 0.1, Leaky Param: 0.1, Rho: -1.0**.

Description	Value
MeanValidationError on fold 0	1.6053982
MeanValidationError on fold 1	2.388435
MeanValidationError on fold 2	1.9684024
MeanValidationError on fold 3	1.1613228
Final mean error in Validation	1.7808895

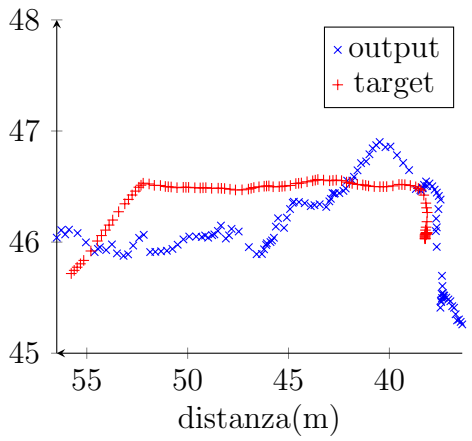
(b) **Result in Test on the data-set LocTurtle/**.

Description	Value
MeanTestError on testSample 0	0.70229256
MeanTestError on testSample 1	1.0756341
MeanTestError on testSample 2	0.91093993
MeanTestError on testSample 3	0.92433554
Final average TestError	0.9033005

Deviazione Standard dell'errore in test 0,132856968 m

Figura B.2: Risultati sul test sample 0

(a) *grafico che confronta il target con l'output del comitato.*



(b) *Distanza euclidea/errore del comitato istante per istante.*

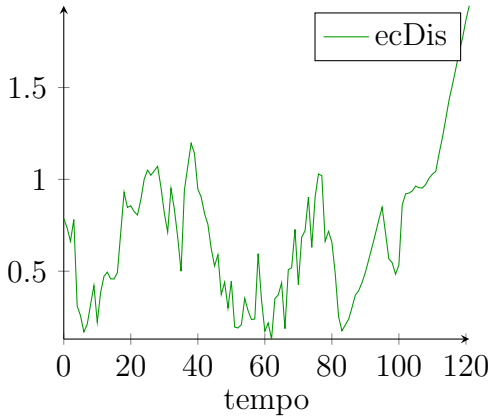


Tabella B.5: **Con azzeramento dello stato del reservoir, senza comitato, e con normalizzazione solo sul training.**

La configurazione del modello (id 27) che in fase di validazione ha ottenuto i migliori risultati è la seguente:

(a) **Reservoir Dim: 50, Regularization: 0.1, Leaky Param: 0.1, Rho: 0.9**.

Description	Value
MeanValidationError on fold 0	2.2943966
MeanValidationError on fold 1	1.9662971
MeanValidationError on fold 2	2.0433426
MeanValidationError on fold 3	2.5769024
Mean Error in Validation for ESN[0]	2.1415129
Mean Error in Validation for ESN[1]	2.4012234
Mean Error in Validation for ESN[2]	2.248621
Mean Error in Validation for ESN[3]	2.203518
Mean Error in Validation for ESN[4]	2.106298
Final mean error in Validation	2.2202346

(b) **Result in Test on the data-set LocTurtle/**.

Description	Value
MeanTestError on testSample 0	1.0868225
MeanTestError on testSample 1	1.2456586
MeanTestError on testSample 2	1.9947819
MeanTestError on testSample 3	1.7602012
Final average TestError	1.5218661

Deviazione Standard dell'errore in test 0,36946126 m

Tabella B.6: **Con azzeramento dello stato del reservoir, senza comitato, e con normalizzazione su tutto il data-set.**

La configurazione del modello (id 76) che in fase di validazione ha ottenuto i migliori risultati è la seguente:

(a) **Reservoir Dim: 100, Regularization: 0.1, Leaky Param: 0.1, Rho: 0.99**.

Description	Value
MeanValidationError on fold 0	1.6285124
MeanValidationError on fold 1	1.8120359
MeanValidationError on fold 2	1.978471
MeanValidationError on fold 3	2.259315
Mean Error in Validation for ESN[0]	1.9100235
Mean Error in Validation for ESN[1]	1.6616794
Mean Error in Validation for ESN[2]	1.6819665
Mean Error in Validation for ESN[3]	2.0534222
Mean Error in Validation for ESN[4]	2.2908263
Final mean error in Validation	1.9195837

(b) **Result in Test on the data-set LocTurtle/**.

Description	Value
MeanTestError on testSample 0	0.8623781
MeanTestError on testSample 1	1.3616712
MeanTestError on testSample 2	1.4786685
MeanTestError on testSample 3	1.5489576
Final average TestError	1.3129189

Deviazione Standard dell'errore in test 0,268584715 m

Tabella B.7: **Con azzeramento dello stato del reservoir, con comitato, e con normalizzazione solo sul training.**

La configurazione del modello (id 40) che in fase di validazione ha ottenuto i migliori risultati è la seguente:

(a) **Reservoir Dim: 50, Regularization: 1.0, Leaky Param: 0.1, Rho: 0.99**.

Description	Value
MeanValidationError on fold 0	1.9412203
MeanValidationError on fold 1	2.1925063
MeanValidationError on fold 2	2.1078377
MeanValidationError on fold 3	2.0160418
Final mean error in Validation	2.0644016

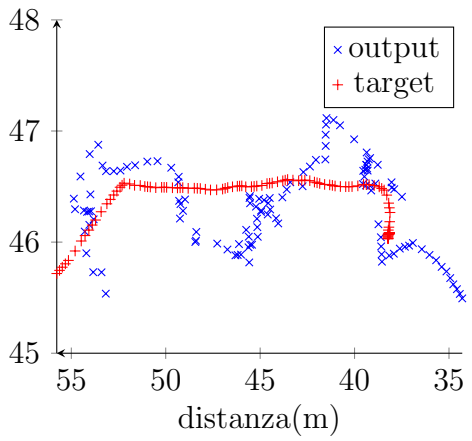
(b) **Result in Test on the data-set LocTurtle/.**

Description	Value
MeanTestError on testSample 0	1.1902522
MeanTestError on testSample 1	1.1935464
MeanTestError on testSample 2	2.0276775
MeanTestError on testSample 3	1.269133
Final average TestError	1.4201523

Deviazione Standard dell'errore in test 0,352171119 m

Figura B.3: Risultati sul test sample 0

(a) *grafico che confronta il target con l'output del comitato.*



(b) *Distanza euclidea/errore del comitato istante per istante.*

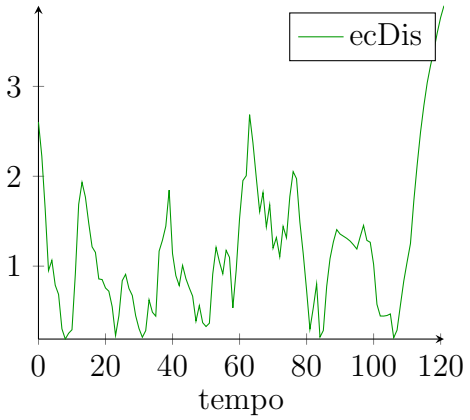


Tabella B.8: **Con azzeramento dello stato del reservoir, con comitato, e con normalizzazione su tutto il data-set.**

La configurazione del modello (id 101) che in fase di validazione ha ottenuto i migliori risultati è la seguente:

(a) **Reservoir Dim: 200, Regularization: 0.001, Leaky Param: 0.1, Rho: -1.0**.

Description	Value
MeanValidationError on fold 0	1.6610851
MeanValidationError on fold 1	2.3440018
MeanValidationError on fold 2	1.6742983
MeanValidationError on fold 3	1.2575841
Final mean error in Validation	1.7342423

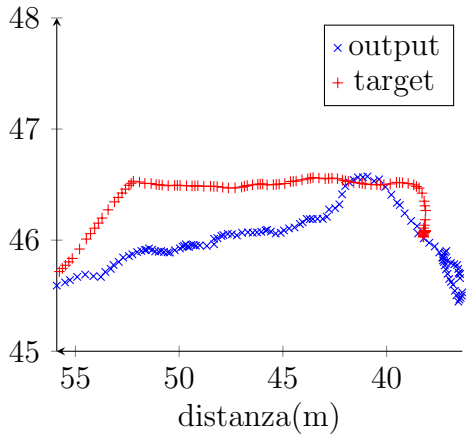
(b) **Result in Test on the data-set LocTurtle/.**

Description	Value
MeanTestError on testSample 0	1.0320219
MeanTestError on testSample 1	0.99629664
MeanTestError on testSample 2	1.0371338
MeanTestError on testSample 3	1.6871774
Final average TestError	1.1881573

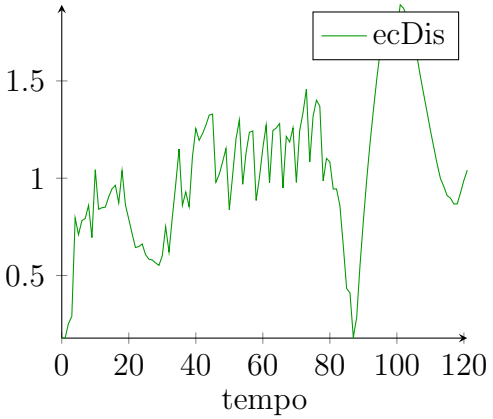
Deviazione Standard dell'errore in test 0,333017934 m

Figura B.4: Risultati sul test sample 0

(a) *grafico che confronta il target con l'output del comitato.*



(b) *Distanza euclidea/errore del comitato istante per istante.*



Come emerge dai risultati il data-set si è rivelato altamente rumoroso. Infatti la deviazione standard ha quasi sempre dei valori molto alti in rapporto all'ordine dell'errore.

Vediamo quali sono i parametri che possono portare dei benefici. Per farlo si esaminano i risultati relativi all'errore sul test-set.

Senza azzeramento dello stato del reservoir e rispettivamente con l'azzeramento dello stato del reservoir, abbiamo i seguenti errori.

- *risultati senza azzeramento dello stato del reservoir*
  $\{1.5500939 \pm 0,337888767;\ 1.1682177 \pm 0,242183673;$
  $1.2862179 \pm 0,531712258;\ 0.9033005 \pm 0,132856968\}$

- *risultati con azzeramento dello stato del reservoir*
  $\{1.5218661 \pm 0,36946126;\ 1.3129189 \pm 0,268584715;$
  $1.4201523 \pm 0,352171119;\ 1.1881573 \pm 0,333017934\}$

Si può osservare che i risultati senza azzeramento dello stato del reservoir hanno quasi sempre una accuratezza migliore. L'unica eccezione si ha nel primo caso, cioè quando si fa l'addestramento senza utilizzare la tecnica del comitato e con la normalizzazione dei dati utilizzando solo il training-set. In questo caso i valori corrispondenti sono però estremamente vicini, differiscono di soltanto 0,0290729 m. Viste quindi le deviazioni standard associate agli errori, questi si possono considerare coincidenti.
Illustriamo quindi i risultati in base all'utilizzo della tecnica del comitato o meno.

- *con la tecnica del comitato*
  $\{1.2862179 \pm 0,531712258;\ 0.9033005 \pm 0,132856968;$
  $1.4201523 \pm 0,352171119;\ 1.1881573 \pm 0,333017934\}$

- *senza il comitato*
  $\{1.5500939 \pm 0,337888767;\ 1.1682177 \pm 0,242183673;$
  $1.5218661 \pm 0,36946126;\ 1.3129189 \pm 0,268584715\}$

L'errore medio nel caso di utilizzo della tecnica del comitato risulta sempre più basso rispetto ai casi in cui non lo si utilizza.
Considerando tutti i risultati visti fino ad ora l'alta varianza tende a far si che questi siano da considerarsi simili. Le eccezioni sono rappresentate dai valori corrispondenti al secondo caso in quest'ultima serie di risultati e all'ultima posizione in quella ancora prima. In entrambi questi casi si utilizza la normalizzazione su tutto il data-set che tende a far abbassare la varianza. Tali osservazioni ci permettono di confermare che l'utilizzo del comitato

senza azzeramento dello stato del reservoir restituisce i valori migliori in termini dell'accuratezza. Ordinando infatti i risultati dal punto di vista della normalizzazione otteniamo i seguenti elenchi.

- *con normalizzazione su tutto il data-set*
  $\{1.1682177 \pm 0,242183673;\ 0.9033005 \pm 0,132856968;$
  $1.3129189 \pm 0,268584715;\ 1.1881573 \pm 0,333017934\}$

- *con normalizzazione solo sul data-set di training*
  $\{1.5500939 \pm 0,337888767;\ 1.2862179 \pm 0,531712258;$
  $1.5218661 \pm 0,36946126;\ 1.4201523 \pm 0,352171119\}$

Come ci si poteva aspettare sia la varianza che l'errore, con la normalizzazione calcolata sull'intero data-set anziché soltanto sul training-set, sono minori. I risultati dell'errore in test ottenuti con questa tecnica però non possono risultare del tutto affidabili in quanto i dati del data-set di test influiscono sul training.
Questo parametro è stato utilizzato per capire le due seguenti.

1. Quanto influisca sul training stesso.

2. Per verificare le osservazioni sui due precedenti parametri.

Si consideri il punto uno. Se facciamo i rapporti tra i rispettivi dell'ultima serie di errori e mediamo si ottiene che il miglioramento dell'errore è di circa il 28% in media. Facendo la stessa cosa sulla varianza si ottiene che il miglioramento è di circa il 96% in media. La normalizzazione su tutto il data-set influisce quindi significativamente sia sull'errore che sulla varianza.
Si consideri il punto due. A causa dell'alta varianza le osservazioni fatte sui due precedenti parametri non sono molto affidabili. Ma nonostante la normalizzazione su tutto il data-set migliori la deviazione standard e l'errore, non dovrebbe modificare l'andamento di quest'ultimo in base agli altri parametri. Ovvero, siano a, b due risultati qualunque nel caso di normalizzazione solo sul training e a', b' i corrispondenti nel caso di normalizzazione su tutto il data-set, si può dire che l'andamento dell'errore è immutato se per ogni coppia a, b è vero che:

$$errore(a) \leq errore(b) \Rightarrow errore(a') \leq errore(b'). \tag{B.1}$$

Se l'andamento resta immutato si possono utilizzare i risultati a varianza più bassa (ottenuti con la normalizzazione su tutto il data-set) per confermare le osservazioni riguardanti i primi due parametri trattati.
Dai valori sopra riportati emerge che l'andamento dell'errore non è rispettato se si considera il primo con il terzo risultato o il primo con il quarto. Ma consideriamo anche la deviazione standard, il primo e il terzo errore con la normalizzazione solo sul training sono i risultati più coincidenti tra tutti. Possiamo quindi considerare in questo caso il loro errore uguale rendendo così vera l'implicazione B.1. La stessa cosa è valida per il primo con il quarto risultato nel caso con la normalizzazione su tutto il data-set. Siamo in grado quindi di concludere che l'andamento dell'errore è immutato, e quindi le osservazioni sui due precedenti parametri possono essere verificate dai risultati ottenuti con la normalizzazione sull'intero data-set.
In seguito per avere dei risultati coerenti in termini di valore dell'errore nel test set la normalizzazione è sempre stata fatta basandoci soltanto sui dati dell'insieme di training. I risultati seguenti sono stati ottenuti sul data-set Stella Maris, considerando i soli sample raccolti durante la notte. **Senza Feature Selection.** La configurazione del modello (id 43) che in fase di validazione ha ottenuto i migliori risultati è la seguente:

Tabella B.9

(a) **Reservoir Dim: 500, Regularization: 0.1, Leaky Param: 0.1, Rho: 0.99**.

Description	Value
MeanValidationError on fold 0	0.641509
MeanTrainingError on fold 0	0.11802024
MeanValidationError on fold 1	0.38118824
MeanTrainingError on fold 1	0.13341317
MeanValidationError on fold 2	0.45191407
MeanTrainingError on fold 2	0.12956205
MeanValidationError on fold 3	0.28785583
MeanTrainingError on fold 3	0.13376537
Final mean error in Training	0.12869021
Mean Error in Validation for ESN[0]	0.4406168
Final mean error in Validation	0.4406168

(b) **Result in Test on the data-set StellaMaris/noNormalizationNight/.**

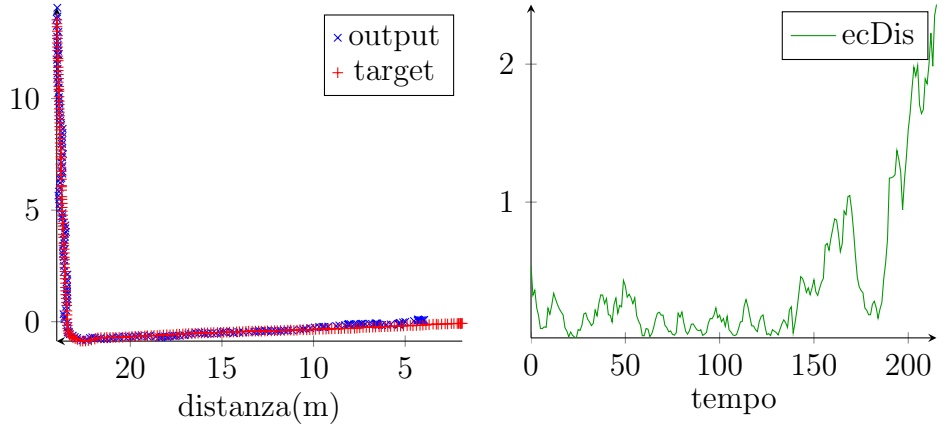
Description	Value
MeanTestError on testSample 0	0.41036588
MeanTestError on testSample 1	0.34207833
MeanTestError on testSample 2	0.31103832
MeanTestError on testSample 3	0.5482947
Final average TestError	0.40294433

Deviazione Standard dell'errore in test 0,091286923 m.

Figura B.5: Risultati sul test sample 13

(a) *grafico che confronta il target con l'output del comitato.*

(b) *Distanza euclidea/errore del comitato istante per istante.*



L'accuratezza del modello finale nel data-set Stella Maris risulta essere molto migliore rispetto al data-set turtle. Inoltre si può osservare che a differenza del data-set turtle la deviazione standard è significativamente più bassa. Questo rende i risultati ottenuti decisamente più affidabili.

B.2 Rapporto dei risultati

Come descritto nell'appendice A l'applicazione produce in automatico una serie di risultati anche in formato LaTeX. Questi risultati possono essere facilmenti raccolti tramite dei template LaTeX pensati appositamente per l'applicazione stessa. Ne segue un esempio completo per chiarire tutti i dati che possono essere raccolti e come sono visualizzati.

B.2.1 Rapporto di un Esperimento con il Dataset Stella Maris Night con Feature Selection ed Euristica per ESN

Results

La configurazione del modello con id 16 che in fase di validazione ha ottenuto i migliori risultati è la seguente:
Reservoir Dim: 100, Regularization: 0.01, Leaky Param: 0.1, Rho: 0.99

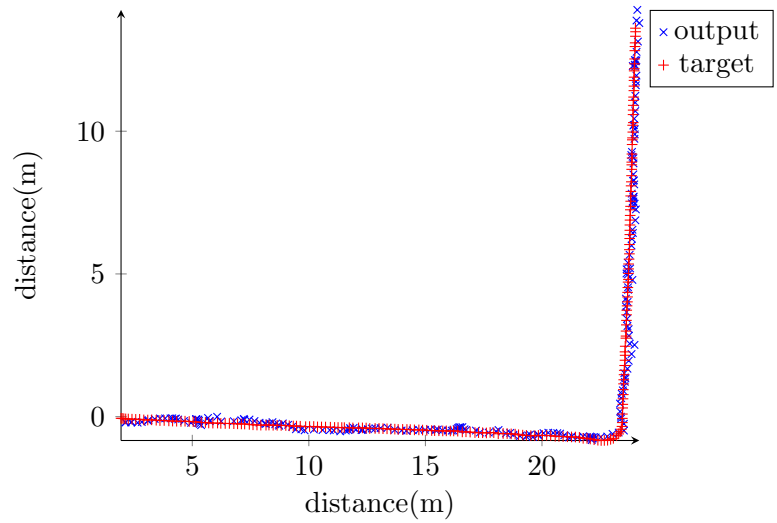
Description	Value
MeanValidationError on fold 0	0.58587456
MeanTrainingError on fold 0	0.26324683
MeanValidationError on fold 1	0.43788752
MeanTrainingError on fold 1	0.2871589
MeanValidationError on fold 2	0.51330626
MeanTrainingError on fold 2	0.2869715
MeanValidationError on fold 3	0.40988788
MeanTrainingError on fold 3	0.29235476
Final mean error in Training	0.282433
Mean Error in Validation for ESN[0]	0.48673907
Final mean error in Validation	0.48673907

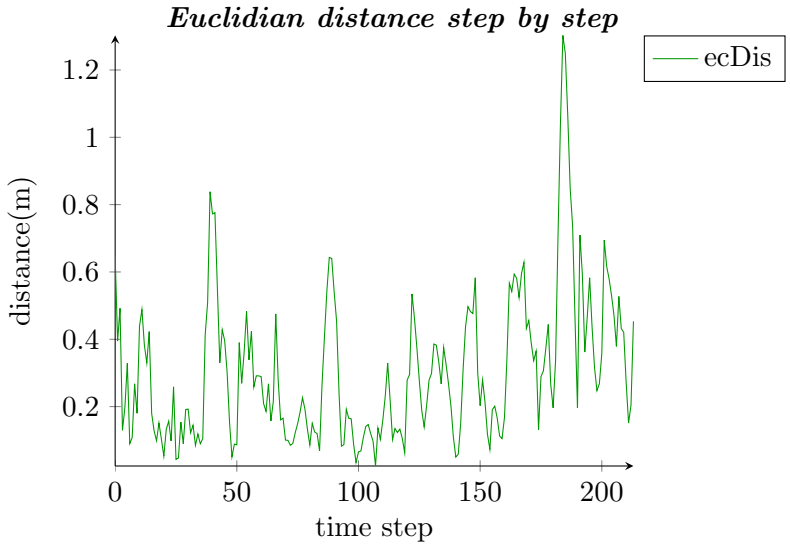
Result in Test on the data-set StellaMaris/noNormalizationNight/

Description	Value
MeanTestError on testSample 0	0.5619835
MeanTestError on testSample 1	0.4278331
MeanTestError on testSample 2	0.49452472
MeanTestError on testSample 3	0.6130098
Final average TestError	0.52433777

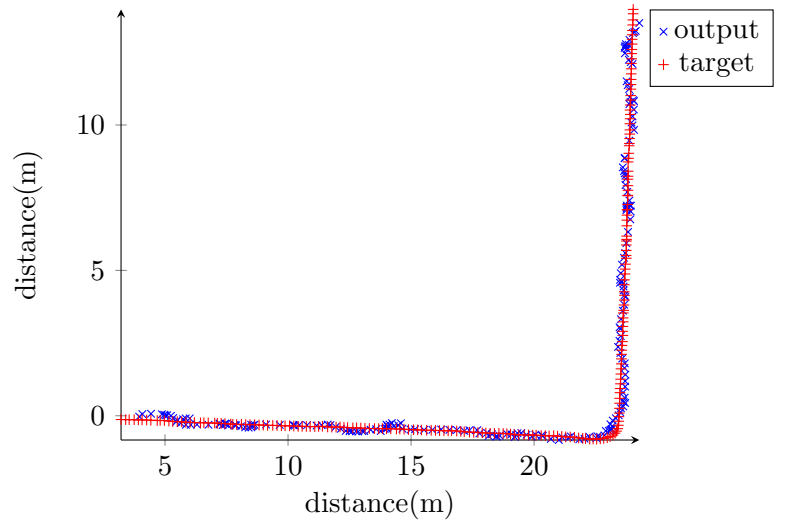
1 Training Results

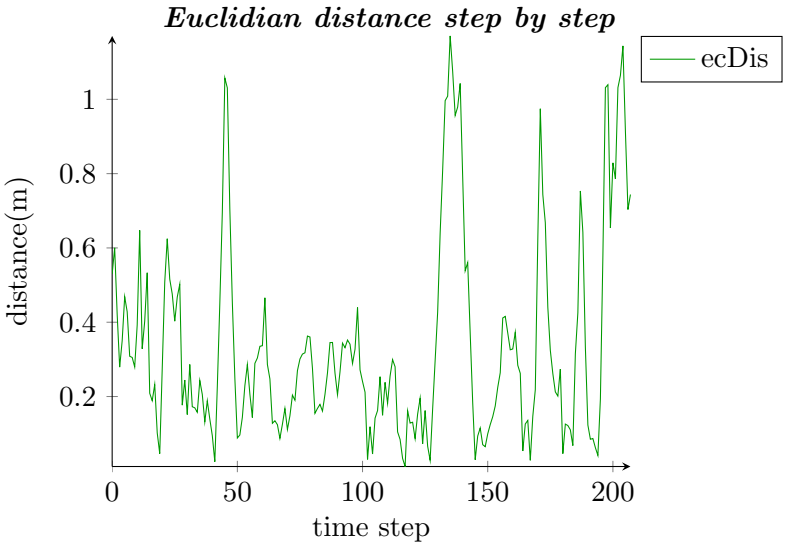
1.1 Training Results on fold 0 and sample 3



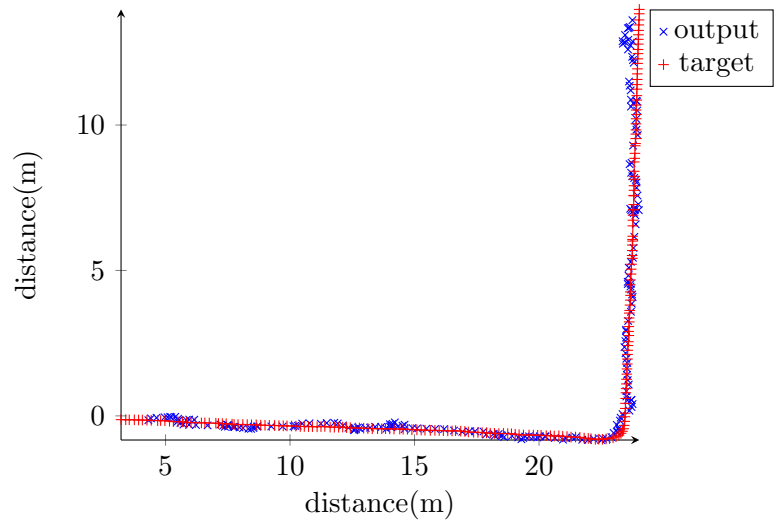


1.2 Training Results on fold 1 and sample 0

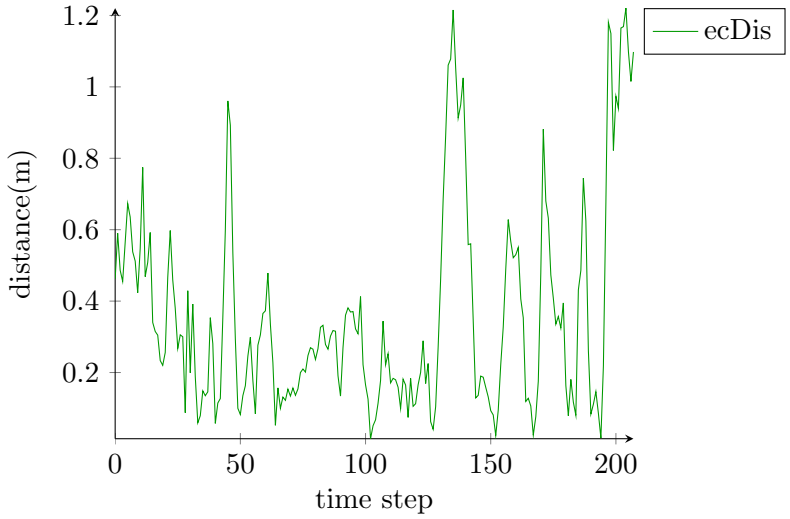




1.3 Training Results on fold 2 and sample 0

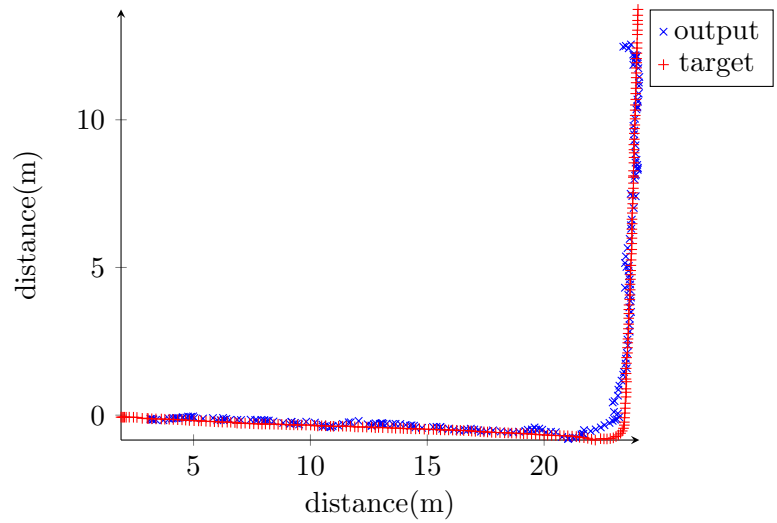


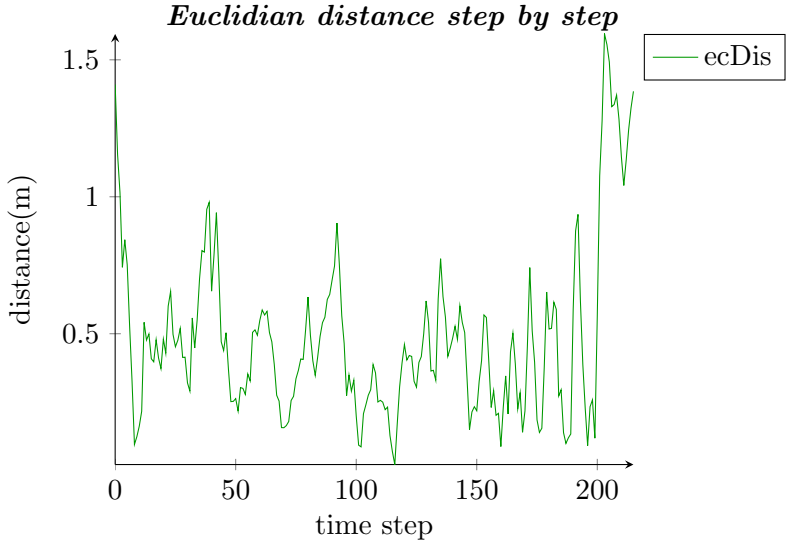
Euclidian distance step by step



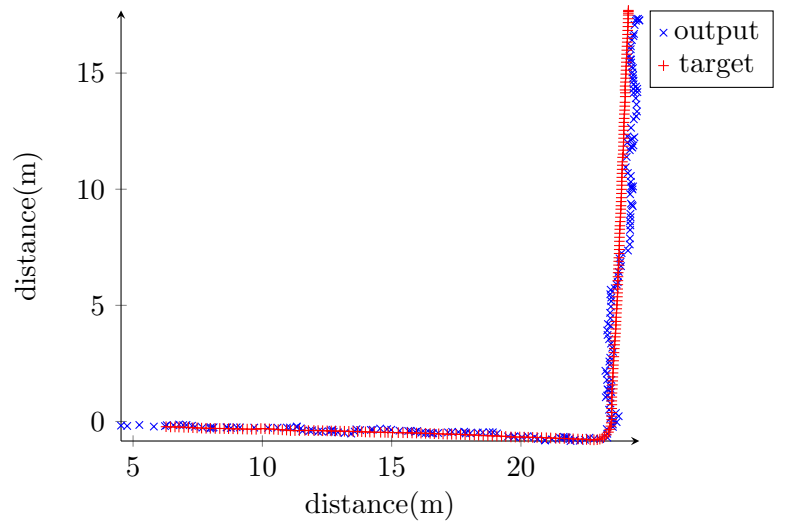
2 Validation Results

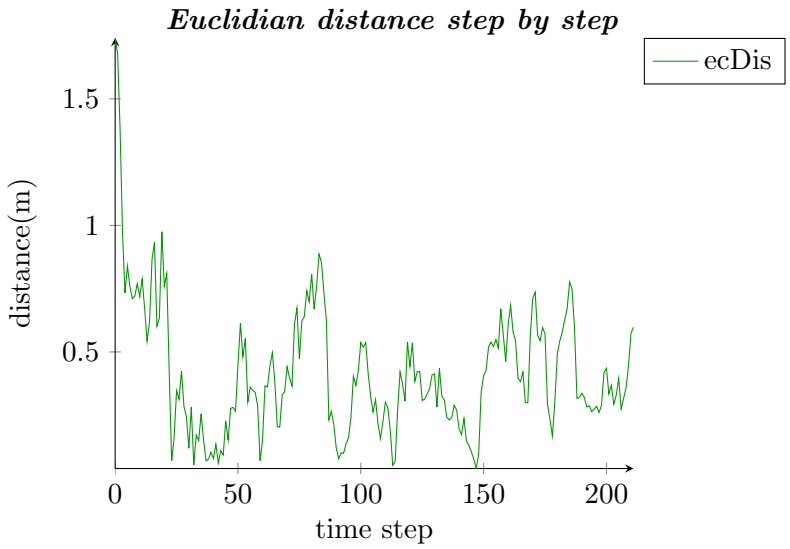
2.1 Validation Results on fold 0 and Sample 1



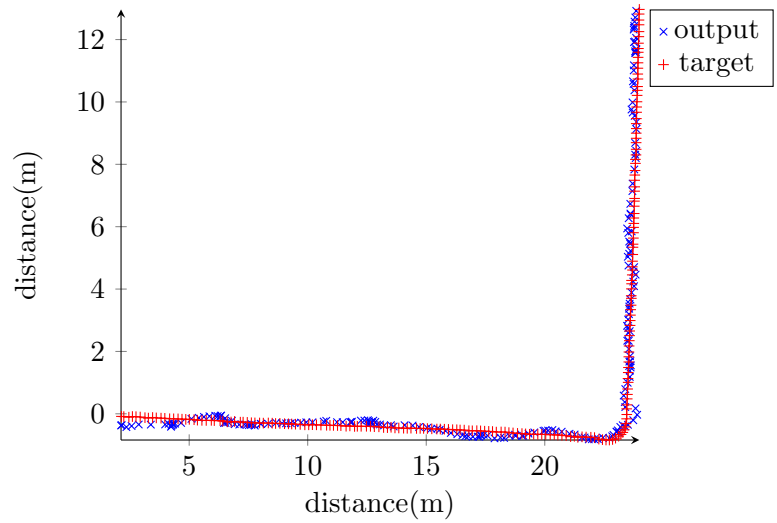


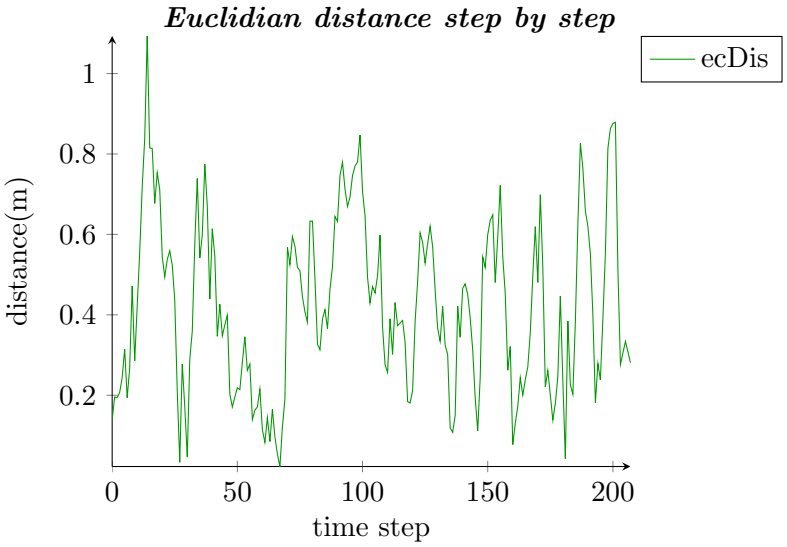
2.2 Validation Results on fold 1 and Sample 4





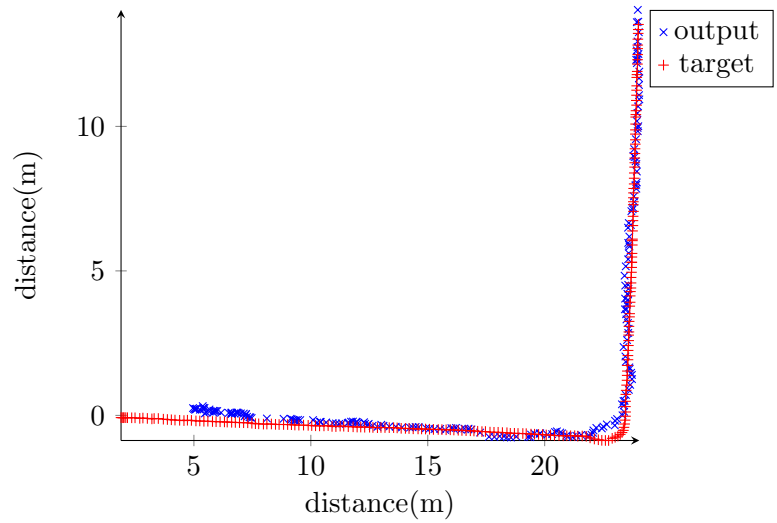
2.3 Validation Results on fold 2 and Sample 7

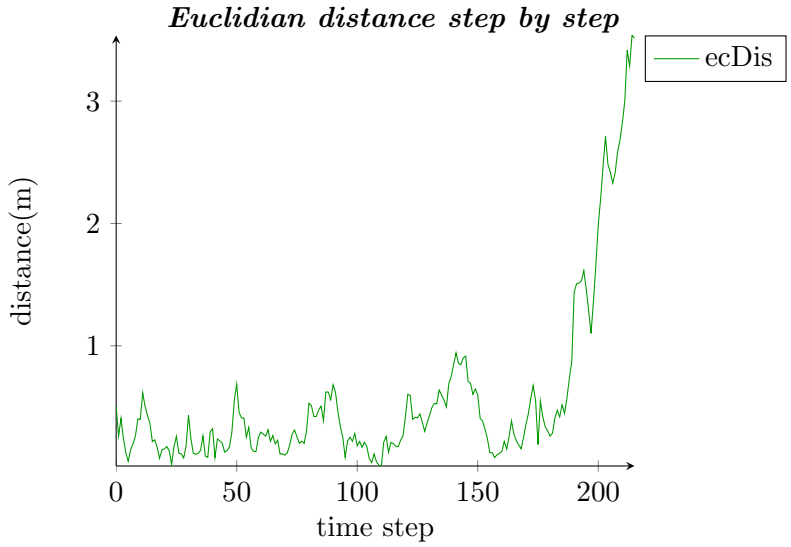




3 Test Results

3.1 Test Results on Sample 13

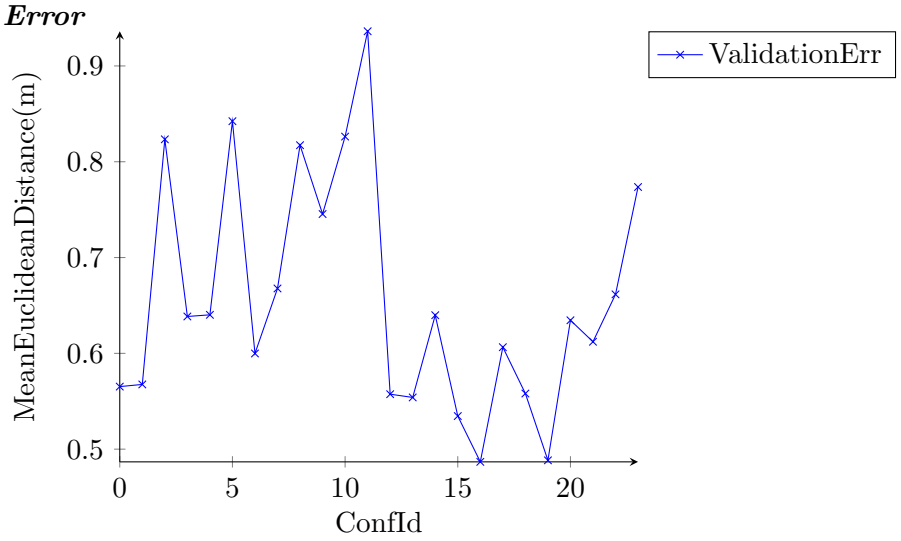




4 All Results Summary

Wrapper Result Summary with AESNIGMAGreedyVersion

ConfId	ResDim	Reg	Leaky	Rho	TrainErr	ValidErr	SelectedFeatureId	Time
0	50	0.001	0.05	0.99	0.35432696	0.5652931	0,1,2,4,6,8;	18.265
1	50	0.001	0.1	0.99	0.4066378	0.56755847	0,4,5;	28.875
2	50	0.001	0.2	0.99	0.6912416	0.82339597	0,4,5;	26.336
3	50	0.01	0.05	0.99	0.39282843	0.6384842	0,1,4,5,6,7,8,9;	17.853
4	50	0.01	0.1	0.99	0.5004096	0.6402093	0,1,2,3,4,5,6,7,9;	20.495
5	50	0.01	0.2	0.99	0.7324643	0.8423241	0,5;	38.116
6	50	0.1	0.05	0.99	0.41078466	0.5998751	0,2,3,4,6;	22.464
7	50	0.1	0.1	0.99	0.52892464	0.6677317	0,1,3,4,5,6,8,9;	17.818
8	50	0.1	0.2	0.99	0.7325711	0.81726164	0,3,4,5;	25.109
9	50	1.0	0.05	0.99	0.5777201	0.7454239	0,2,3,4,5,8,9;	21.705
10	50	1.0	0.1	0.99	0.69095135	0.82626057	0,2,3,4,5,6,7,8;	15.059
11	50	1.0	0.2	0.99	0.87222135	0.9362063	0,1,4,5;	31.661
12	100	0.001	0.05	0.99	0.20849985	0.5573755	0,1,2,3,4,5,6,8,9;	43.886
13	100	0.001	0.1	0.99	0.31997088	0.5539846	0,2,3,5,6,7,8;	58.718
14	100	0.001	0.2	0.99	0.4676294	0.63982797	0,1,4,5;	100.123
15	100	0.01	0.05	0.99	0.22405341	0.53453904	0,1,2,3,4,5,6,8;	67.196
16	100	0.01	0.1	0.99	0.282433	0.48673907	0,1,3,4,5,6;	81.69
17	100	0.01	0.2	0.99	0.45316023	0.60641825	0,1,4,5;	130.89
18	100	0.1	0.05	0.99	0.29959068	0.55806696	0,1,2,4,6,9;	74.222
19	100	0.1	0.1	0.99	0.34409156	0.4883232	0,1,4,5,6;	113.021
20	100	0.1	0.2	0.99	0.5020522	0.63452685	0,3,4,5;	113.204
21	100	1.0	0.05	0.99	0.442275	0.61212456	0,1,3,4,5;	74.829
22	100	1.0	0.1	0.99	0.52326214	0.6615807	0,2,3,4,5,9;	64.034
23	100	1.0	0.2	0.99	0.64003694	0.77362806	0,1,3,4,5,6,9;	82.418



Bibliografia

[1] "RUBICON Project." [Online]. Available: http://www.fp7rubicon.eu

[2] K. Ducatel, M. Bogdanowicz, F. Scapolo, J. Leijten, and J.-C. Burgelman, "Scenarios for ambient intelligence in 2010," 2001. [Online]. Available: http://scholar.google.de/scholar.bib?q=info:tEBLvUuJDTMJ:scholar.google.com/&output=citation&hl=de&oi=citation;http://www.bibsonomy.org/bibtex/2f3dc232f0a10a04a32f67b144b67b0e7/altmann

[3] P. Baronti, P. Pillaia, V. W. C. Chooka, S. Chessa, A. Gotta, and Y. F. Hu, "Wireless Sensor Networks: A Survey on the State of the Art and the 802.15.4 and ZigBee Standards," *Computer Communications*, vol. 30, no. 7, pp. 1655–1695, 2007. [Online]. Available: http://www.bibsonomy.org/bibtex/21e0fb6a3554235074c6cb1293641ff76/flint63

[4] T. Mitchell, *Machine Learning*. McGraw Hill, 1997. [Online]. Available: http://www.bibsonomy.org/bibtex/2e1eee0d0daaef20092093d6643b53c4f/prlz77

[5] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1999. [Online]. Available: http://www.bibsonomy.org/bibtex/2e13a49ec715f4e52f1dcca0d4c5c8b2c/prlz77

[6] H. Jaeger, "The echo state approach to analysing and training recurrent neural networks," GMD - German National Research Institute for Computer Science, GMD Report 148, 2001. [Online]. Available: http://www.faculty.jacobs-university.de/hjaeger/pubs/EchoStatesTechRep.pdf;http://www.bibsonomy.org/bibtex/23d434b04cf1479acf45be9af65f8bc78/idsia

[7] H. Jaeger and H. Haas, "Harnessing Nonlinearity: Predicting Chaotic Systems and Saving Energy in Wireless Communication," *Science*, vol.

304, pp. 78–80, 2004. [Online]. Available: http://www.bibsonomy.org/bibtex/26f24a8b3c1858e19b32292f11aa72f71/butz

[8] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Computer Science Review*, vol. 3, no. 3, pp. 127–149, August 2009. [Online]. Available: http://minds.jacobs-university.de/sites/default/files/uploads/papers/2261_LukoseviciusJaeger09.pdf

[9] W. Maass, T. Natschläger, and H. Markram, "Real-Time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations," *Neural Computation*, vol. 14, no. 11, pp. 2531–2560, 2002. [Online]. Available: http://dx.doi.org/10.1162/089976602760407955;http://www.bibsonomy.org/bibtex/217074dd0f955ec917ef94dd8d8357931/meduz

[10] O. Obst, "Distributed Fault Detection in Sensor Networks using a Recurrent Neural Network," *Neural Processing Letters*, pp. 1–13, 2013. [Online]. Available: http://dx.doi.org/10.1007/s11063-013-9327-4

[11] M. Chang, A. Terzis, and P. Bonnet, "Mote-Based Online Anomaly Detection Using Echo State Networks." in *DCOSS*, ser. Lecture Notes in Computer Science, B. Krishnamachari, S. Suri, W. R. Heinzelman, and U. Mitra, Eds., vol. 5516. Springer, 2009, pp. 72–86. [Online]. Available: http://dblp.uni-trier.de/db/conf/dcoss/dcoss2009.html#ChangTB09; http://dx.doi.org/10.1007/978-3-642-02085-8_6;http://www.bibsonomy.org/bibtex/2b457f4ec08071046527e64da588718bc/dblp

[12] C. Gallicchio, A. Micheli, P. Barsocchi, and S. Chessa, "User Movements Forecasting by Reservoir Computing Using Signal Streams Produced by Mote-Class Sensors," in *Mobile Lightweight Wireless Systems*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, J. Del Ser, E. Jorswieck, J. Miguez, M. Matinmikko, D. Palomar, S. Salcedo-Sanz, and S. Gil-Lopez, Eds. Springer Berlin Heidelberg, 2012, vol. 81, pp. 151–168. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-29479-2_12

[13] D. Bacciu, S. Chessa, C. Gallicchio, A. Micheli, and P. Barsocchi, "An Experimental Evaluation of Reservoir Computation for Ambient Assisted

Living," in *Neural Nets and Surroundings*, ser. Smart Innovation, Systems and Technologies, B. Apolloni, S. Bassis, A. Esposito, and F. C. Morabito, Eds. Springer Berlin Heidelberg, 2013, vol. 19, pp. 41–50. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-35467-0_5

[14] D. Bacciu, P. Barsocchi, S. Chessa, C. Gallicchio, and A. Micheli, "An experimental characterization of reservoir computing in ambient assisted living applications," *Neural Computing and Applications*, vol. 24, no. 6, pp. 1451–1464, 2014. [Online]. Available: http://dx.doi.org/10.1007/s00521-013-1364-4

[15] G. Amato, M. Broxvall, S. Chessa, M. Dragone, C. Gennaro, R. López, L. Maguire, T. Mcginnity, A. Micheli, A. Renteria, G. M. O'Hare, and F. Pecora, "Robotic UBIquitous COgnitive Network," in *Ambient Intelligence - Software and Applications*, ser. Advances in Intelligent and Soft Computing. Springer Berlin Heidelberg, 2012, vol. 153, pp. 191–195. [Online]. Available: http://link.springer.com/chapter/10.1007%2F978-3-642-28783-1_23

[16] D. Bacciu, M. Broxvall, S. Coleman, M. Dragone, C. Gallicchio, C. Gennaro, R. Guzmán, R. López, H. Lozano-Peiteado, A. K. Ray, A. Rentería, A. Saffiotti, and C. Vairo, "Self-sustaining Learning for Robotic Ecologies," in *SENSORNETS 2012 - Proceedings of the 1st International Conference on Sensor Networks, Rome, Italy, 24-26 February, 2012*, M. van Sinderen, O. Postolache, and C. Benavente-Peces, Eds. SciTePress, 2012, pp. 99–103. [Online]. Available: http://academia.edu/2832581/Self-Sustaining_Learning_for_Robotic_Ecologies

[17] J.-h. Kim, Y.-d. Kim, and K.-h. Lee, "The Third Generation of Robotics: Ubiquitous Robot Abstract," in *Autonomous Robots And Agents*, 2nd International Conference. ICARA, 2004.

[18] C. Gallicchio and A. Micheli, "Architectural and Markovian factors of echo state networks." *Neural Networks*, vol. 24, no. 5, pp. 440–456, 2011. [Online]. Available: http://dblp.uni-trier.de/db/journals/nn/nn24.html#GallicchioM11; http://dx.doi.org/10.1016/j.neunet.2011.02.002;http://www.bibsonomy.org/bibtex/296a4e408bd80a1a2731d00980b9737d7/dblp

[19] H. Jaeger, "The ''echo state'' approach to analysing and training recurrent neural networks - with an Erratum note," Fraunhofer Institute for Autonomous Intelligent Systems, Tech. Rep., 2010, there is an Erratum Note for this techreport. [Online]. Available: http://www.faculty.jacobs-university.de/hjaeger/pubs/EchoStatesTechRep.pdf

[20] H. Jaeger, M. Lukoševičius, D. Popovici, and U. Siewert, "Optimization and Applications of Echo State Networks with Leaky Integrator Neurons," *Neural Networks*, vol. 20, no. 3, pp. 335–352, 2007. [Online]. Available: http://www.faculty.jacobs-university.de/hjaeger/pubs/leakyESN.pdf;http://www.bibsonomy.org/bibtex/29674247330d9697eb91b860d3dbdf636/idsia

[21] M. Lukoševičius, D. Popovici, H. Jaeger, and U. Siewert, "Time Warping Invariant Echo State Networks," International University Bremen, Tech. Rep., 2006, technical Report No. 2.

[22] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Computer Science Review*, vol. 3, no. 3, pp. 127–149, August 2009. [Online]. Available: http://minds.jacobs-university.de/sites/default/files/uploads/papers/2261_LukoseviciusJaeger09.pdf

[23] E. A. Antonelo, B. Schrauwen, and D. Stroobandt, "Event detection and localization for small mobile robots using reservoir computing," *Neural Networks*, vol. 21, no. 6, pp. 862–871, 2008.

[24] Q. Song, "Effect of Leaky Rate on the Stability of Autonomously Echo State Network," *Technology Journal*, vol. 11, no. 7, p. 775, 2012.

[25] C. Gennaro, M. Broxvall, and C. Vairo, "D1.1 – Functional Design & Specification & Mockup Layer ," On-line, RUBICON, 2011, rUBICON Deliverable.

[26] A. Saffiotti and M. Broxvall, "PEIS Ecologies: Ambient Intelligence Meets Autonomous Robotics," in *Proceedings of the 2005 Joint Conference on Smart Objects and Ambient Intelligence: Innovative Context-aware Services: Usages and Technologies*, ser. sOc-EUSAI '05.

New York, NY, USA: ACM, 2005, pp. 277–281. [Online]. Available: http://doi.acm.org/10.1145/1107548.1107615

[27] A. Saffiotti, M. Broxvall, M. Gritti, K. LeBlanc, R. Lundh, J. Rashid, B. Seo, and Y. Cho, "The PEIS-Ecology project: Vision and results," in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, Sept 2008, pp. 2329–2335.

[28] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. Addison Wesley, 2006. [Online]. Available: http://www.bibsonomy.org/bibtex/21312364d45c358ebe9a7d1575d6af725/utahell

[29] R. Kohavi and G. H. John, "Wrappers for Feature Subset Selection," *Artificial Intelligence*, vol. 97, no. 1-2, pp. 273–324, 1997. [Online]. Available: http://www.bibsonomy.org/bibtex/2a1f1716aec9383a7f7c344bf68cfa58b/mumas

[30] H. Almuallim and T. G. Dietterich, "Learning with many irrelevant features," in *Proc. of the 9th National Conf. on Artificial Intelligence*, vol. 2, 1991, pp. 547–552. [Online]. Available: http://www.bibsonomy.org/bibtex/2ca0f1cfe4a5d18983e058b7d653cd385/dalbem

[31] J. H. Gennari, P. Langley, and D. Fisher, "Models of incremental concept formation," *Artif. Intell.*, vol. 40, no. 1-3, pp. 11–61, Sep. 1989. [Online]. Available: http://dx.doi.org/10.1016/0004-3702(89)90046-5;http://www.bibsonomy.org/bibtex/23d6efd40933c04936201e05bc49614e0/jullybobble

[32] R. Kohavi, "Wrappers for performance enhancement and oblivious decision graphs." Ph.D. dissertation, Stanford University, 1995, doctoral dissertation.

[33] M. A. Hall, "Correlation based feature selection for machine learning," The University of Waikato, Tech. Rep., 1999, doctoral dissertation.

[34] S. Das, "Filters, Wrappers and a Boosting-Based Hybrid for Feature Selection," in *Proceedings of the Eighteenth International Conference on Machine Learning*, ser. ICML '01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 74–81. [Online]. Available: http://dl.acm.org/citation.cfm?id=645530.658297

[35] J. Huang, Y. Cai, and X. Xu, "A hybrid genetic algorithm for feature selection wrapper based on mutual information." *Pattern Recognition Letters*, vol. 28, no. 13, pp. 1825–1844, 2007. [Online]. Available: http://dblp.uni-trier.de/db/journals/prl/prl28.html#HuangCX07;http://dx.doi.org/10.1016/j.patrec.2007.05.011;http://www.bibsonomy.org/bibtex/2aab0f55e4db90d5a2ee2897f8e8a1b59/dblp

[36] D. Aha and R. Bankert, "A Comparative Evaluation of Sequential Feature Selection Algorithms," in *Learning from Data*, ser. Lecture Notes in Statistics, D. Fisher and H.-J. Lenz, Eds. Springer New York, 1996, vol. 112, pp. 199–206. [Online]. Available: http://dx.doi.org/10.1007/978-1-4612-2404-4_19

[37] J. Doak, *An Evaluation of Feature Selection Methods and Their Application to Computer Security*. University of California, Department of Computer Science, 1992. [Online]. Available: http://books.google.it/books?id=S_zhtgAACAAJ

[38] R. Caruana and D. Freitag, "Greedy Attribute Selection," in *In Proceedings of the Eleventh International Conference on Machine Learning*, W. W. Cohen and H. Hirsh, Eds. Morgan Kaufmann, 1994, pp. 28–36. [Online]. Available: http://dblp.uni-trier.de/db/conf/icml/icml1994.html#CaruanaF94;http://www.bibsonomy.org/bibtex/2dd3ec470e00e2aa6c16955a446ce318e/dblp

[39] C. Lai, M. J. T. Reinders, and L. F. A. Wessels, "Random subspace method for multivariate feature selection." *Pattern Recognition Letters*, vol. 27, no. 10, pp. 1067–1076, 2006. [Online]. Available: http://dblp.uni-trier.de/db/journals/prl/prl27.html#LaiRW06;http://dx.doi.org/10.1016/j.patrec.2005.12.018;http://www.bibsonomy.org/bibtex/2df5b9b2a6fc55b35865ce25f6229e4a3/dblp

[40] P. Pudil, J. Novovicová, and J. Kittler, "Floating search methods in feature selection." *Pattern Recognition Letters*, vol. 15, no. 10, pp. 1119–1125, 1994. [Online]. Available: http://dblp.uni-trier.de/db/journals/prl/prl15.html#PudilNK94;http://dx.doi.org/10.1016/0167-8655(94)90127-9;http://www.bibsonomy.org/bibtex/266e8ef0de9fb098394dbe81ea009b453/dblp

[41] P. Somol, B. Baesens, P. Pudil, and J. Vanthienen, "Filter-versus wrapper-based feature selection for credit scoring." *Int. J. Intell. Syst.*, vol. 20, no. 10, pp. 985–999, 2005. [Online]. Available: http://dblp.uni-trier.de/db/journals/ijis/ijis20.html#SomolBPV05;http://dx.doi.org/10.1002/int.20103;http://www.bibsonomy.org/bibtex/2e4c9393ca217e1b8bb54c737c5a44e95/dblp

[42] Y. Freund and R. E. Schapire, "A Short Introduction to Boosting," in *In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 1999, pp. 1401–1406.

[43] J. Quinlan, "Induction of decision trees," *Journal of Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986. [Online]. Available: http://www.bibsonomy.org/bibtex/2f0dbd04fef1bb84d3c2705c92ab5ad3d/bsmyth

[44] L. Talavera, "An Evaluation of Filter and Wrapper Methods for Feature Selection in Categorical Clustering," in *Advances in Intelligent Data Analysis VI*, ser. Lecture Notes in Computer Science, A. Famili, J. Kok, J. Peña, A. Siebes, and A. Feelders, Eds. Springer Berlin Heidelberg, 2005, vol. 3646, pp. 440–451. [Online]. Available: http://dx.doi.org/10.1007/11552253_40

[45] C.-N. Hsu, H.-J. Huang, and S. Dietrich, "The ANNIGMA-wrapper approach to fast feature selection for neural nets." *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 32, no. 2, pp. 207–212, 2002. [Online]. Available: http://dblp.uni-trier.de/db/journals/tsmc/tsmcb32.html#HsuHD02;http://dx.doi.org/10.1109/3477.990877;http://www.bibsonomy.org/bibtex/27eaa44873649e0aa230c5155c6613a91/dblp

[46] Y.-C. Yeh, W.-J. Wang, and C. W. Chiou, "Feature selection algorithm for ECG signals using Range-Overlaps Method." *Expert Syst. Appl.*, vol. 37, no. 4, pp. 3499–3512, 2010. [Online]. Available: http://dblp.uni-trier.de/db/journals/eswa/eswa37.html#YehWC10; http://dx.doi.org/10.1016/j.eswa.2009.10.037;http://www.bibsonomy.org/bibtex/2ce1d90044841af8addbe627ed74d00ca/dblp

[47] J. Hall, M. Barbeau, and E. Kranakis, "Enhancing intrusion detection in wireless networks using radio frequency fingerprinting." in *Communications, Internet, and Information Technology*, M. H. Hamza,

Ed. IASTED/ACTA Press, 2004, pp. 201–206. [Online]. Available: http://dblp.uni-trier.de/db/conf/ciit/ciit2004.html#HallBK04;http://www.bibsonomy.org/bibtex/2d42a9128c021a3b0fee7a58f028dfe1e/dblp

[48] H. Xiong, C. Daniel, S. Śaunak, and S. M. R., "Sequence-Based Classification Using Discriminatory Motif Feature Selection," *PLoS ONE*, vol. 6, no. 11, p. e27382, 11 2011. [Online]. Available: http://dx.doi.org/10.1371%2Fjournal.pone.0027382

[49] T. Dietterich, "Ensemble Methods in Machine Learning," in *Multiple Classifier Systems*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2000, vol. 1857, pp. 1–15. [Online]. Available: http://dx.doi.org/10.1007/3-540-45014-9_1

[50] M. Lukosevicius, "A Practical Guide to Applying Echo State Networks." in *Neural Networks: Tricks of the Trade (2nd ed.)*, ser. Lecture Notes in Computer Science, G. Montavon, G. B. Orr, and K.-R. Müller, Eds. Springer, 2012, vol. 7700, pp. 659–686. [Online]. Available: http://dblp.uni-trier.de/db/series/lncs/lncs7700.html#Lukosevicius12;http://dx.doi.org/10.1007/978-3-642-35289-8_36;http://www.bibsonomy.org/bibtex/22815a1ef4a068a5de893c32f501ffcbf/dblp

[51] D. Verstraeten, J. Dambre, X. Dutoit, and B. Schrauwen, "Memory versus non-linearity in reservoirs," in *Neural Networks (IJCNN), The 2010 International Joint Conference on*. IEEE, July 2010, pp. 1–8.

[52] H. Jaeger, *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the echo state network approach*. German National Research Center for Information Technology, 2002, report 159.

[53] "MEMSIC." [Online]. Available: http://www.memsic.com/wireless-sensor-networks/index.cfm

[54] G. Dissanayake, P. Newman, S. Clark, D. H. Whyte, and M. Csorba, "A solution to the simultaneous localization and map building (SLAM) problem," *IEEE Transactions on Neural Networks*, vol. 17, no. 3, pp. 229–241, 2001. [Online]. Available: http://www.bibsonomy.org/bibtex/2e939c41d08182c628a8c54a0a4c2cdce/idsia

[55] T. Catuogno, "Supporto per l'apprendimento e localizzazione di una piattaforma robotica mobile," 2013, tesi del corso triennale in informatica.

[56] "Robotnik agvs." [Online]. Available: http://www.robotnik.eu/mobile-robots/agvs