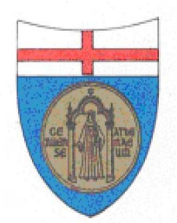
Università degli studi di Genova

Facoltà di Scienze Matematiche Fisiche e Naturali
Corso di Laurea in Fisica



Anno Accademico 2003/2004

Tesi di Laurea

Metodi di ensemble per la previsione di serie storiche

Candidato

Filippone Maurizio

Relatori

Prof. Francesco Masulli

Prof. Massimo Riani

Correlatore

Dott. Claudia Patrignani

A mio padre

Ringraziamenti

Con questo lavoro si chiude un capitolo importante della mia vita e desidero ringraziare tutti coloro che, in questi anni, mi hanno sostenuto. Il ringraziamento più grande va ai miei genitori che mi hanno dato l'opportunità di affrontare questa esperienza. Un pensiero grande è per i miei familiari che non sono qui a gioire con me in questi momenti.

Fra le persone che mi sono state più vicino ringrazio Erica, Andrea e gli Zenith (Simo, Dario, Roby, Fra) per aver suonato con me più di dieci milioni di note! Quindi tutto il gruppo dell'Universale Alba Docilia di Genova, Albisola e Vado (troppi per nominare tutti), lo staff Sheraton per la loro disponibilità e simpatia e la famiglia Dell'Amico che mi ha accolto con grande affetto. Un ringraziamento speciale al Prof. Tartarini, che in questi anni mi ha seguito con pazienza nell'attività agonistica facendomi vivere una grande esperienza formativa. Ancora gli amici del DIFI con cui ho preparato alcuni esami e con cui ho trascorso tante ore nei vari laboratori.

Riguardo alla tesi ringrazio Matteo e Dario per i preziosi consigli su Linux (e non solo), i simpaticissimi colleghi del ViCoLab e Nabil (anche per le lezioni di arabo (شُكرًا نَبِيل!)).
Infine vorrei ringraziare Francesco Masulli per avere messo a mia disposizione molte risorse per realizzare questo lavoro e Valentini, Rovetta e Riani per tutti i consigli che mi hanno dato.

Indice

Introduzione

Lo scopo di questa tesi è quello di valutare l'efficacia dei metodi di ensemble per la previsione di serie storiche. I metodi che si vogliono analizzare hanno già portato a grossi risultati in problemi di classificazione. Questa tesi è uno dei primi tentativi di applicazione estensiva all'utilizzo dei metodi di ensemble per la previsione di serie storiche. Per ottenere tutti i risultati è stata sviluppata una apposita libreria in linguaggio R.

La tesi è suddivisa in due parti. Nella prima vengono descritti gli strumenti necessari per poter affrontare il problema della previsione. In particolare nel primo capitolo viene fatta una panoramica sulle macchine d'apprendimento e nel terzo vengono descritte quelle base utilizzate. Nel secondo si introduce il problema di previsione delle serie storiche e il teorema di embedding, il quale permette di trasformare il problema di previsione in uno di regressione. Nel quarto capitolo si introducono i metodi di ensemble e si descrivono più in dettaglio i metodi implementati.

La seconda parte è dedicata ai risultati sperimentali ottenuti, basati sul confronto di tutti i metodi implementati su diversi data base. Il capitolo cinque introduce la parte sperimentale descrivendo l'ambiente software utilizzato, chiamato R, spiegando come è stata sviluppata una apposita libreria in questo linguaggio implementando i vari algoritmi. I metodi vengono inizialmente messi a confronto in un problema di regressione. Nei tre capitoli successivi viene affrontato il problema di previsione su data base differenti con serie storiche di difficoltà di previsione crescente e vengono riportati i

principali risultati ottenuti.

Nelle conclusioni vengono riportati i principali risultati ottenuti e le prospettive. Nell'appendice A viene fornita una descrizione più tecnica dell'ambiente software utilizzato.

Parte I

Aspetti teorici

Capitolo 1

Apprendimento automatico

n questo capitolo vengono presentati gli aspetti generali dell'apprendimento automatico e delle grandezze che caratterizzano le macchine ad apprendimento automatico. Si approfondisce lo studio del problema della regressione che è alla base dei metodi predittivi delle serie storiche che saranno trattati nei successivi capitoli.

1.1 Macchine ad apprendimento automatico

L'*apprendimento* [9] è il processo che conduce alla stima di una relazione sconosciuta fra ingresso e uscita di un sistema, o della sua struttura, utilizzando un numero finito di osservazioni. L'apprendimento è un processo che viene implementato da una *macchina di apprendimento* (*learning machine* o *learner*). Il termine automatico indica che le macchine d'apprendimento sono *adattive*, ovvero sono in grado di adattarsi al problema che viene loro sottoposto.
I principali compiti che vengono assegnati alle macchine ad apprendimento automatico sono:

- *regressione* o approssimazione di funzioni;
- *classificazione*;
- *stima di densità di probabilità*;
- *quantizzazione vettoriale* o *clustering*.

Esistono due principali categorie di problemi di apprendimento che vengono detti *supervisionato* e *non supervisionato*. L'apprendimento supervisionato, al quale appartengono i problemi di classificazione e regressione, permette di stimare una relazione fra variabili a partire da una serie di esempi noti.
Nell'apprendimento non supervisionato è la macchina che deve cercare le varie associazioni e le strutture fra dati in input (problema di clustering) e quindi non è presente la nozione di output durante la fase di addestramento.

Indicando con $(\mathbf{x}_i, y_i)_{i=1,\cdots,n}$ le n coppie ingresso-uscita di un sistema, si distinguono due fasi all'interno delle operazioni di un algoritmo di apprendimento supervisionato:

- fase di addestramento o di training in cui si effettua la stima della relazione fra le variabili a partire da un set di coppie conosciute chiamato *training set*;

- fase operativa o di test in cui si produce una predizione a partire da un nuovo insieme di input noto come *test set*.

Per *generalizzazione* si intende la capacità di ottenere predizioni corrette in fase operativa.
Perché il problema di apprendimento sia consistente si assume che sia il training set che il test set siano stati generati con la stessa distribuzione statistica $p(\mathbf{x}, y) = p(\mathbf{x})p(y|\mathbf{x})$ (*identical indipendent distributed* o *i.i.d.*).

In generale la macchina di apprendimento implementa un set di funzioni $f(\mathbf{x}, \omega)$, parametrizzate da $\omega \in \Omega$, dove Ω è un insieme astratto di parametri. Lo scopo dell'apprendimento è quello di scegliere, all'interno del set a disposizione, quella che meglio approssima la risposta del sistema a partire da un numero finito di esempi.

La qualità dell'approssimazione prodotta viene quantificata attraverso una funzione detta di perdita (*loss function*) $L(y, f(\mathbf{x}, \omega))$ che fornisce un valore tanto più piccolo quanto l'errore di approssimazione è piccolo. Il valore atteso di L viene chiamato *funzionale di rischio*:

$$R(\omega) = \int L(y, f(\mathbf{x}, \omega))p(\mathbf{x}, y)d\mathbf{x}dy \tag{1.1}$$

L'apprendimento è il processo di stima della funzione $f(\mathbf{x}, \omega_0)$ che minimizza il funzionale $R(\omega)$ all'interno del set di funzioni supportate dalla macchina d'apprendimento. La disponibilità di un set di esempi finito non permette di ottenere esattamente $f(\mathbf{x}, \omega_0)$ bensì una funzione che denotiamo con $f(\mathbf{x}, \omega^*)$ ottenuta minimizzando il *rischio empirico*:

$$R_{\text{emp}}(\omega) = \frac{1}{n}\sum_{i=1}^{n} L(y, f(\mathbf{x}_i, \omega)) \tag{1.2}$$

In generale la conoscenza esatta della distribuzione $p(\mathbf{x}, y)$ consentirebbe di minimizzare l'eq. 1.1, risolvendo il problema di apprendimento. Questo

significa che il problema generale è quello della stima di densità di probabilità, che a partire da un numero finito di esempi, nella terminologia di Hadamard [20], non è ben posto[1]. Nel resto del capitolo ci occuperemo del compito di regressione.

1.2 Regressione

La regressione è un processo di stima di una funzione a valori reali basato sulla conoscenza di un set finito di campioni affetti da rumore. In questa trattazione si farà riferimento al problema di approssimazione di una funzione $\mathbb{R}^d \rightarrow \mathbb{R}$ con d generico. Si suppone che il valore assunto da una variabile di uscita di un sistema deterministico, in seguito al processo di misura, possa essere interpretata come somma di una parte deterministica e di una casuale ε a media nulla[2]:

$$y = g(\mathbf{x}) + \varepsilon \tag{1.3}$$

La parte deterministica si può pensare essere:

$$g(\mathbf{x}) = \int yp(y|\mathbf{x})dy \tag{1.4}$$

In generale il set di funzioni messe a disposizione dalla macchina può non contenere la funzione $g(\mathbf{x})$. Una funzione di perdita comunemente utilizzata in regressione è l'errore quadratico:

$$L(y, f(\mathbf{x}, \omega)) = (y - f(\mathbf{x}, \omega))^2 \tag{1.5}$$

Quindi l'apprendimento si esaurisce nella ricerca della funzione $f(\mathbf{x}, \omega_0)$ che minimizza il rischio:

$$R(\omega) = \int (y - f(\mathbf{x}, \omega))^2 p(\mathbf{x}, y)d\mathbf{x}dy \tag{1.6}$$

[1] Un problema è ben posto quando esiste una soluzione, è unica e dipende continuamente dai dati iniziali. Un problema non ben posto non soddisfa almeno una di queste condizioni.

[2] Se il rumore non avesse media nulla tale valore sarebbe deterministico e andrebbe in $g(\mathbf{x})$

avendo a disposizione solo gli esempi forniti dal training set. Si dimostra che, assumendo che la media del rumore sia nulla, l'eq. 1.6 può essere riscritta:

$$R(\omega) = \int E[\varepsilon^2|\mathbf{x}]p(\mathbf{x})d\mathbf{x} + \int (f(\mathbf{x},\omega) - g(\mathbf{x}))^2 p(\mathbf{x})d\mathbf{x} \tag{1.7}$$

dove con $E[]$ si indica il valore atteso. Siccome la varianza di ε non dipende da ω, minimizzare solo il secondo termine dell'eq. 1.7 equivale a minimizzare l'eq. 1.6. Da questo si evince che la funzione con il minimo rischio è quella che meglio approssima la funzione incognita $g(\mathbf{x})$.

L'adattività delle macchine d'apprendimento mette a disposizione un set di funzioni approssimanti molto vasto cosicché è necessario porre una sorta di vincolo, utilizzando qualche informazione a priori sulla funzione da approssimare. Osservando infatti l'eq. 1.2, se il set di funzioni messo a disposizione della macchina fosse l'insieme delle funzioni continue, la minimizzazione del rischio empirico porterebbe ad avere infinite soluzioni. In altre parole bisogna cercare un compromesso fra la complessità del modello e la sua capacità di approssimazione che si ottiene aggiungendo al rischio empirico un termine di *regolarizzazione*:

$$R_{\text{reg}}(\omega) = R_{\text{emp}}(\omega) + \lambda\varphi(f(\mathbf{x},\omega)) \tag{1.8}$$

dove la φ solitamente è una funzione che misura la complessità di $f(\mathbf{x},\omega)$. In questo modo viene aggiunta un'informazione a priori, tradotta in un termine di penalizzazione per le funzioni più complesse, il cui peso può essere variato attraverso il valore di λ. Un valore ottimale di λ garantisce quindi un buon compromesso fra complessità ed approssimazione in modo da ottenere buona generalizzazione.

1.3 Complessità e generalizzazione

Riguardo alla possibilità di misurare la complessità di una funzione bisogna chiarire alcuni aspetti. Si possono innanzitutto fare alcune considerazioni

intuitive:

- la dimensionalità di una funzione non è una misura della sua complessità anche se essa è sinonimo di potenzialità di essere complessa;
- la presenza di poche oscillazioni o morbidezza (*smoothness*) di una funzione è indice di semplicità;
- la caratterizzazione della complessità deve tenere conto della dimensionalità e della smoothness della funzione.

Dalla teoria sull'approssimazione di funzioni sappiamo che è possibile approssimare uniformemente qualsiasi funzione continua in un insieme compatto tramite combinazione lineare di funzioni base:

$$f_m(x, \mathbf{w}) = \sum_{i=1}^{m} w_i g_i(x) \tag{1.9}$$

Le funzioni $g_i(x)$ possono essere, per esempio, polinomi algebrici (Weierstrass) o polinomi trigonometrici (Fourier). Un modo per caratterizzare la complessità di una funzione può appoggiarsi sulla costruzione data dall'eq. 1.9.
Il termine di penalità può essere costruito in modo parametrico o non parametrico. Nel modo parametrico i vincoli vengono introdotti come funzioni dei parametri di f:

$$\varphi(f(x, \mathbf{w}_m)) = \eta(\mathbf{w}_m) \tag{1.10}$$

Solitamente le funzioni η sono della forma:

$$\eta(\mathbf{w}_m) = \sum_{i=1}^{m} w_i^2 \tag{1.11}$$

$$\eta(\mathbf{w}_m) = \sum_{i=1}^{m} I(w_i \neq 0) \tag{1.12}$$

dove I è la funzione indicatrice pari ad uno se la condizione è verificata e zero altrimenti.

Nel modo non parametrico si controlla la smoothness di una funzione direttamente utilizzando operatori differenziali.

Un modo per caratterizzare la complessità di una macchina d'apprendimento è utilizzare la dimensione di Vapnick-Chervonenkis (*VC dimension*) per il relativo set di funzioni approssimanti [9] [23]. La VC dimension viene introdotta per i problemi di dicotomizzazione come il massimo numero di esempi separabili in tutti i modi possibili dal set di funzioni implementato dal learner.

Un altro modo utilizzato per mostrare il compromesso fra complessità ed approssimazione è mediante la riscrittura del secondo membro dell'eq. 1.7:

$$\begin{aligned} E[(f(\mathbf{x},w)-g(\mathbf{x}))^2] &= E[(f(\mathbf{x},w)-E[f(\mathbf{x},w)])^2]+ \\ &+ (g(\mathbf{x})-E[f(\mathbf{x},w)])^2 \end{aligned} \tag{1.13}$$

Per la media globale su $\mathbf{x}$, l'errore quadratico medio, il *variance* e il *bias* vengono definiti come:

$$mse(f(\mathbf{x},w)) = \int E[(g(\mathbf{x})-f(\mathbf{x},w))^2]p(\mathbf{x})d\mathbf{x} \tag{1.14}$$

$$bias^2(f(\mathbf{x},w)) = \int (g(\mathbf{x})-E[f(\mathbf{x},w)])^2 p(\mathbf{x})d\mathbf{x} \tag{1.15}$$

$$var(f(\mathbf{x},w)) = \int E[(f(\mathbf{x},w)-E[f(\mathbf{x},w)])^2]p(\mathbf{x})d\mathbf{x} \tag{1.16}$$

Quindi si ha la scrittura:

$$mse(f(\mathbf{x},w)) = bias^2(f(\mathbf{x},w)) + var(f(\mathbf{x},w)) \tag{1.17}$$

Un valore di λ piccolo nell'eq. 1.8, fa sì che la funzione approssimi molto bene la distribuzione dei punti a scapito della smoothness, portando ad avere un alto variance (fig. 1.1). Con λ grande si prediligono funzioni di complessità minore, peggiorando l'approssimazione.

Nei problemi reali resta la difficoltà di stimare bias e variance poiché non si può conoscere la distribuzione di probabilità senza essere in possesso di

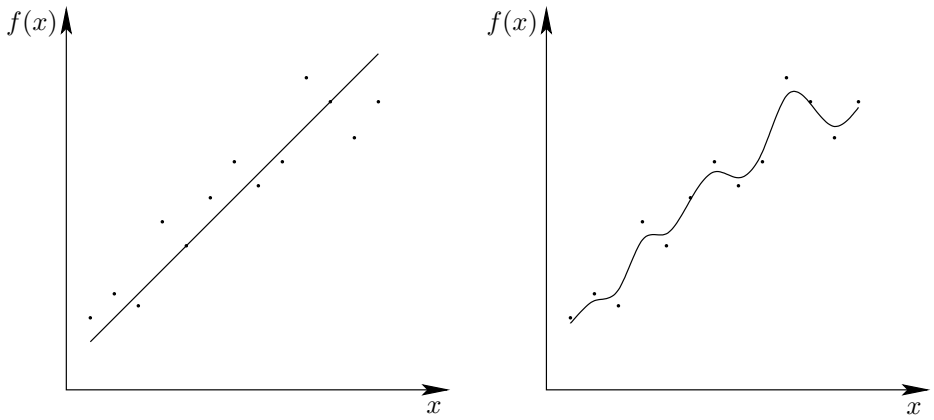


Figura 1.1: Due funzioni $f(x)$ con complessità diversa approssimanti la stessa distribuzione di punti. La prima ha complessità piccola mentre la seconda maggiore (variance più alto).

una grande quantità di dati. In problemi artificiali si possono determinare empiricamente bias e variance mediando su un numero molto grande di training set.

Capitolo 2

Previsione di serie storiche

Questo capitolo presenta una descrizione generale delle caratteristiche dei sistemi dinamici per poi introdurre i sistemi caotici e alcuni metodi per analizzare le serie storiche. Il problema della predizione senza la conoscenza a priori di un modello del sistema, infatti, necessita di opportuni metodi di indagine in modo da individuare le strutture e le caratteristiche sottostanti alle serie storiche.
Un elemento fondamentale per la trattazione è il teorema di Takens-Mañé [28] [25] che garantisce, sotto opportune condizioni, la possibilità di ricostruire la dinamica del sistema a partire dall'osservazione di una osservabile relativa al sistema stesso. Questi consente di trasformare il problema di previsione in uno di regressione.

Nella parte finale del capitolo viene introdotta l'SSA come strumento per l'estrazione di trend dalle serie storiche.

2.1 Sistemi dinamici

Con *sistema dinamico* [8] si definisce una struttura caratterizzata da:

- uno spazio degli stati $\mathcal{S}$;
- un parametro temporale che assume valori in $\mathbb{R}$ o $\mathbb{Z}$, solitamente indicato con t;
- una legge di evoluzione, cioè un'applicazione $(t, x) \rightarrow \varphi_t(x) \in \mathcal{S}$, $x \in \mathcal{S}$, tale che $\varphi_0(x) = x \ \forall x \in \mathcal{S}$.

La struttura dello spazio $\mathcal{S}$ è legata alla natura del sistema che si intende descrivere, dovendo i suoi elementi rappresentare in maniera univoca i possibili stati del sistema reale. Un processo di questo tipo può inoltre essere descritto in maniera naturale con un tempo continuo o discreto.
Ciò che caratterizza i sistemi in studio è:

- determinismo: il presente determina univocamente passato e futuro;
- dimensione finita dello spazio $\mathcal{S}$;
- differenziabilità: $\mathcal{S}$ possiede la struttura di una varietà differenziabile che è conservata dalla legge di evoluzione.

Il determinismo implica che $\varphi_t(\varphi_s(x)) = \varphi_{t+s}(x)$ per ogni t, s ed ogni $x \in \mathcal{S}$. Seguono inoltre alcune proprietà fra le quali il fatto che la famiglia $\{\varphi_t\}$ di endomorfismi di $\mathcal{S}$ costituisce un gruppo commutativo ad un parametro di trasformazioni di $\mathcal{S}$.
Nel caso di tempo discreto, assegnando una mappa g, applicazione biunivoca di $\mathcal{S}$ in se stesso, la legge di evoluzione si esprime iterando g:

$$g_0(x) = x \quad g_1(x) = g(x) \quad g_2(x) = g(g(x)) \cdots g_k(x) = g(g_{k-1}(x)) \tag{2.1}$$

Tornando al caso continuo quando si parla di sistema dinamico è dunque necessario assegnare una coppia $(\mathcal{S}, \{\varphi_t\})$. La funzione $t \rightarrow \varphi_t$ è detta *legge*

oraria di x, mentre la sua immagine in $\mathcal{S}$ viene chiamata *orbita* di x.
Utilizzando le ipotesi di differenziabilità e di finitezza della dimensione di $\mathcal{S}$, segue che $\{\varphi_t\}$ è un gruppo ad un parametro di diffeomorfismi di $\mathcal{S}$.
Per gli scopi di questa trattazione non è restrittivo considerare applicazioni $\mathbf{\Phi}_t : D \to \mathbb{R}^n$ a tempo continuo con $D \subseteq \mathbb{R}^n$ aperto.
Definiamo:

$$\mathbf{v}(\mathbf{x}) = \frac{d}{dt}\mathbf{\Phi}_t(\mathbf{x})\bigg|_{t=0} \tag{2.2}$$

Notiamo che per la proprietà di gruppo, per ogni $t \in \mathbb{R}$:

$$\frac{d}{dt}\mathbf{\Phi}_t(\mathbf{x}) = \lim_{h\to 0}\frac{\mathbf{\Phi}_{t+h}(\mathbf{x}) - \mathbf{\Phi}_t(\mathbf{x})}{h} = \lim_{h\to 0}\frac{\mathbf{\Phi}_h(\mathbf{\Phi}_t(\mathbf{x})) - \mathbf{\Phi}_t(\mathbf{x})}{h} = \mathbf{v}(\mathbf{\Phi}_t(\mathbf{x})) \tag{2.3}$$

In altri termini ogni processo evolutorio è soluzione dell'equazione:

$$\dot{\mathbf{x}} = \mathbf{v}(\mathbf{x}) \tag{2.4}$$

In questa formulazione il campo vettoriale $\mathbf{v}$ non dipende esplicitamente dal tempo, avendo assunto che la legge di evoluzione non dipenda dall'istante iniziale. Aggiungendo questa ulteriore condizione bisogerà assegnare una famiglia di applicazioni $\mathbf{\Phi}_{t,t_0} : \mathcal{S} \to \mathcal{S}$ al variare di $t, t_0 \in \mathbb{R}$.
Ripetendo le stesse considerazioni precedenti, ricordando che l'identità ora sarà $\mathbf{\Phi}_{t_0,t_0}$, si giunge alla conclusione che i moti sono soluzioni dell'equazione differenziale:

$$\dot{\mathbf{x}} = \mathbf{v}(t, \mathbf{x}) \tag{2.5}$$

Le soluzioni di questa equazione sono caratteristiche del sistema in studio. In particolare risultano interessanti gli studi sulla presenza di punti di equilibrio e di cicli limite che si configurano come un sottoinsieme di $\mathcal{S}$, chiamato *attrattore*, verso i quali tende lo stato del sistema.

2.2 Sistemi caotici

Con *sistema caotico* [1] si intende una classe di sistemi dinamici caratterizzata da due aspetti fondamentali:

- le orbite nello spazio $\mathcal{S}$ presentano una struttura complessa. La struttura in $\mathcal{S}$ dell'attrattore per i sistemi caotici è quella che viene chiamata frattale ed è costituita da un sottoinsieme $A \subset \mathcal{S}$ in generale di dimensione non intera. Il frattale geometrico A è caratterizzato dalla proprietà di autosimiglianza su diverse scale ovvero l'orbita ripete dei particolari andamenti su diversi ingrandimenti in $\mathcal{S}$.

- sensibilità dalle condizioni inziali. Tale proprietà consiste nel fatto che l'evoluzione del sistema a partire da due stati molto vicini, segue traiettorie molto diverse. Questa estrema sensibilità dalle condizioni iniziali è nota anche come *effetto farfalla*.

La spiegazione della complessità delle soluzioni è che la traiettoria dello stato del sistema viene continuamente deviata dalla presenza di un'infinità di orbite periodiche instabili e punti di equilibrio instabili. Siccome la misura dell'insieme di questi oggetti è nulla, la probabilità che lo stato cada in uno di essi è nulla e nel contempo basta una piccola perturbazione per fare allontanare il sistema da essi.

L'insorgere della caoticità nelle soluzioni avviene anche per un numero relativamente piccolo di gradi di libertà. In particolare, per poter avere una dinamica caotica bastano:

- tre equazioni differenziali autonome di primo ordine per sistemi descritti da equazioni differenziali (es. sistema di Lorenz [1]);

- due gradi di libertà per sistemi a tempo discreto descritti da mappe invertibili;

- un grado di libertà per sistemi a tempo discreto descritti da mappe non invertibili (es. mappa logistica [3]).

Per gradi di libertà si intende, nel caso di sistemi a tempo discreto, il numero di componenti del vettore di stato $\mathbf{x}(t)$.

2.3 Teorema di embedding

Supponendo di non possedere un modello per descrivere l'evoluzione di un sistema dinamico ci si chiede se sia possibile, a partire dall'osservazione di una o poche variabili dinamiche relative ad esso, ottenere informazioni sufficienti sulla sua dinamica per poter impostare il problema della previsione. Infatti per un qualsiasi sistema dinamico è tipico osservare, attraverso un processo di campionamento, solo una o poche variabili dinamiche relative ad esso. Nel caso sia possibile bisogna chiedersi anche sotto quali condizioni questo possa avvenire.
La risposta a queste domande viene data dal teorema di Takens-Mañé, chiamato *teorema di embedding* [28] [25]. Pensando di campionare un segnale scalare $s(t)$ con tempo di campionamento τ_s a partire dall'istante t_0, si otterrà una serie $\{s_k\}_{k=1\cdots N}$ con $s_k = s(t_0 + k\tau_\mathrm{s})$ che chiamiamo *serie storica*.

Si suppone di avere a che fare con un sistema dinamico descritto da una mappa $\mathbf{F}$ che fa evolvere $\mathbf{x}(n) \rightarrow \mathbf{F}(\mathbf{x}(n)) = \mathbf{x}(n+1)$, con $\mathbf{x}(n) \in \mathbb{R}^n$. Il teorema assicura che a partire dall'osservazione di una singola quantità scalare $h(\bullet)$ di una funzione vettoriale $\mathbf{g}(\mathbf{x}(n))$, la struttura della dinamica in $\mathcal{S}$ può essere spiegata[1] nello spazio Y_d dei vettori:

$$\mathbf{y}(n) = (h(\mathbf{x}(n)), h(\mathbf{g}^1(\mathbf{x}(n))), \cdots, h(\mathbf{g}^{d-1}(\mathbf{x}(n)))) \tag{2.6}$$

Questi vettori definiscono una traiettoria nello spazio euclideo $d-$dimensionale Y_d.

[1] Con questo termine si indica che la rappresentazione della traiettoria in Y_d è legata a quella in $\mathcal{S}$ da una relazione di diffeomorfismo.

Quello che viene richiesto è che d sia abbastanza grande e che le funzioni h e $\mathbf{g}$ siano almeno di classe C^2. Se questo avviene le caratteristiche delle orbite nello spazio delle fasi vengono riprodotte senza ambiguità nello spazio degli $\mathbf{y}(n)$. In particolare è necessario dimensionare i vettori $\mathbf{y}(n)$ in modo da evitare errori dovuti alla proiezione dell'evoluzione di $\mathbf{x}(n)$ in uno spazio scalare. Questi errori si hanno quando due punti lontani della dinamica vengono proiettati vicini lungo l'asse scalare delle ossevazioni.

Per il teorema di embedding, la scelta di h e $\mathbf{g}$ è arbitraria. Risulta conveniente utilizzare applicazioni semplici e che si riferiscono alla grandezza osservabile. Per h scegliamo la variabile scalare osservata $s(n)$:

$$h(\mathbf{x}(n)) = s(n) \tag{2.7}$$

e come $\mathbf{g}(\mathbf{x})$ l'operatore che, dato un vettore $\mathbf{x}(n)$, restituisce $\mathbf{x}(n+p)$ quello cioè relativo allo stato del sistema dopo un tempo p. Le potenze di questo operatore, $\mathbf{g}^k$ sono:

$$\mathbf{g}^k(\mathbf{x}(n)) = \mathbf{x}(n + pk) \tag{2.8}$$

Si ottiene la seguente scrittura per i vettori $\mathbf{y}(n)$:

$$\mathbf{y}(n) = (s(n), s(n+T), s(n+2T), \cdots, s(n+(d-1)T)) \tag{2.9}$$

dove T rappresenta nel dominio temporale un ritardo pari a un multiplo del τ_{s}. Questi vettori si possono dunque ottenere molto semplicemente campionando con tempo di campionamento τ_{s} una grandezza fisica del sistema in studio.

La rappresentazione nel nuovo spazio è legata a quella nello spazio $\mathcal{S}$ da una relazione differenziabile che permette la ricostruzione dell'attrattore senza perdita di informazione. Finora non si è ancora posto alcuna ipotesi riguardo la scelta di d e T (e quindi in qualche modo di τ_{s}).
Dalla teoria sulle equazioni differenziali sappiamo che le orbite non presentano incroci in $\mathcal{S}$. Nel passggio allo spazio Y_d risulta dunque necessario

raggiungere una dimensione d sufficiente per non avere incroci della traiettoria. Il teorema di embedding afferma che se l'attrattore nello spazio $\mathcal{S}$ è di dimensione d_{A}, la quale può essere non intera, è sufficiente scegliere $d > 2d_{\mathrm{A}}$ per non avere intersezioni dell'orbita con se stessa in Y_d.
È necessario porre l'attenzione sul fatto che esistono una serie di difficoltà nella stima di d_{A} e che la scelta di T è importante per ricostrutire bene l'attrattore.

Riassumendo il teorema di embedding consente di osservare la dinamica del sistema nello spazio Y_d (che si costruisce in maniera piuttosto semplice) assicurando che non si perde nessuna informazione posseduta dall'evoluzione in $\mathcal{S}$.

2.4 Ricostruzione della dinamica

Il teorema di embedding di Takens-Mañé non fornisce un criterio per la scelta di T. Bisogna cercare un modo per sceglierlo tenendo presente che solitamente si ha a disposizione un segnale il cui campionamento è legato alle caratteristiche del processo fisico in studio e che si è in possesso di una serie di lunghezza finita (più o meno lunga). In letteratura non esiste un teorema che assicuri una scelta ottimale di T. Bisogna quindi ricercare un criterio ragionevole per ottenere almeno una stima di dove possa cadere il valore ottimale e successivamente criticare la scelta di un particolare T sulla base dei risultati ottenuti o dal confronto con altri metodi. Innanzitutto si possono fare alcune considerazioni di carattere generale:

- T dovrà essere un multiplo di τ_{s}, siccome si è in possesso di una serie $\{s_i\}_{i=1,\cdots,N}$ ottenuta campionando con questo particolare tempo di campionamento.

- Una scelta di T troppo piccolo fa sì che due coordinate $s(n)$ e $s(n+T)$ da utilizare in un vettore $\mathbf{y}(n)$ siano troppo dipendenti fra loro e non

portano sostanziali nuove informazioni sulla dinamica nello spazio Y_d e quindi su quella in $\mathcal{S}$.

- Visto che i sistemi caotici sono intrinsecamente dotati di grande variabilità, se T è troppo grande ogni correlazione fra due coordinate svanisce totalmente, facendole sembrare legate da una relazione casuale.

Quindi quello che si richiede è una prescrizione che permetta di identificare un ritardo T che sia largo abbastanza perché $s(n)$ e $s(n+T)$ siano sufficientemente indipendenti ma non troppo, in modo che non siano completamente indipendenti in senso statistico. Nelle prossime sezioni seguiamo due approcci, di cui uno è basato sulla generazione di informazione di un sistema caotico, mentre un altro è legato alle carateristiche di memoria di un segnale [1].

2.4.1 Mutua informazione

Fondamentale per la nozione di informazione fra misure è l'idea di Shannon di mutua informazione fra due misure a_i e b_j estratte rispettivamente dai set A e B di possibili realizzazioni. La mutua informazione è la quantità di informazione acquisita quando si è verificata l'estrazione di a_i seguita dall'estrazione di b_j [1]. In bit:

$$\mu_{ij} = \log_2 \left(\frac{P_{AB}(a_i, b_j)}{P_A(a_i) P_B(b_j)} \right) \tag{2.10}$$

$P_{AB}(a, b)$ è la densità di probabilità congiunta per le realizzazioni a e b. $P_A(a)$ e $P_B(b)$ sono le densità di probabilità relative ai set A e B. Se l'estrazione di a è completamente indipendente dall'estrazione di b allora $P_{AB}(a, b)$ si può fattorizzare nel prodotto $P_{AB}(a, b) = P_A(a) P_B(b)$ e la mutua informazione risulta zero. Si definisce quindi la mutua informazione media, che si ottiene facendo la media delle mutue informazioni su tutte le possibili estrazioni dai set A e B, pesata da $P_{AB}(a, b)$. In simboli:

$$I_{AB} \;=\; \sum_{a_i, b_j} P_{AB}(a_i, b_j) \mu_{ij} =$$

$$= \sum_{a_i,b_j} P_{AB}(a_i,b_j)\log_2\left(\frac{P_{AB}(a_i,b_j)}{P_A(a_i)P_B(b_j)}\right) \tag{2.11}$$

Ricordando di essere in possesso della serie $\{s_i\}_{i=1,\cdots,N}$, utilizziamo questa formulazione nel nostro caso in cui si vuole valutare la mutua informazione fra s_n e s_{n+T}. Ora $A \equiv B$ rappresenta l'insieme delle possibili realizzazioni di s_n e in generale sarà un sottinsieme di $\mathbb{R}$. Si ottiene la mutua informazione media in bits:

$$I(T) = \sum_{s_n,s_{n+T}} P(s_n,s_{n+T})\log_2\left(\frac{P(s_n,s_{n+T})}{P(s_n)P(s_{n+T})}\right) \tag{2.12}$$

che chiaramente è una funzione di T. In particolare all'aumentare di questi, s_n e s_{n+T} diventano sempre più scorrelati fra loro e $I(T)$ tenderà a zero. Si osserva che le distribuzioni $P(s_n)$ e $P(s_{n+T})$ coincidono.

Seguendo il suggerimento di Fraser [1], la scelta del valore di T da utilizzare nella costruzione del vettore delle coordinate ritardate dovrebbe cadere intorno al primo minimo della mutua informazione media. È solo una prescrizione ma si è visto essere ragionevole nello studio degli attrattori. Questo tipo di scelta, infatti, dovrebbe fare in modo che T possieda le caratteristiche richieste.

Una precisazione è necessaria per spiegare in che modo vengono calcolate le varie distribuzioni di probabilità. Infatti si ha a che fare con distribuzioni di probabilità su sottoinsiemi di $\mathbb{R}$ che in generale saranno della forma di aperti $A = (u,v)$, con $u,v \in \mathbb{R}$. Senza nessuna precisazione, N realizzazioni per la variabile s_n, porterebbero ad avere in generale ad avere una $P(s_n)$ zero quasi ovunque e $\frac{1}{N}$ dove si è verificata una realizzazione. Si possono ripetere le stesse considerazioni per la probabilità congiunta $P(s_n,s_{n+T})$. In questo modo il calcolo della $I(T)$ non verrebbe fatto in maniera corretta.
Quello che si fa allora, è dividere l'intervallo (a,b) in k sottointervalli (tipicamente nell'ordine del centinaio) contigui:

$$A \;=\; (u,v) = \Delta_1 \cup \Delta_2 \cup \Delta_3 \cup \cdots \cup \Delta_k \qquad \text{con:}$$

$$\begin{aligned} \Delta_1 &= \left(u, u + \frac{v-u}{k}\right) \\ \Delta_j &= \left[u + (j-1)\frac{v-u}{k}, u + j\frac{v-u}{k}\right) \qquad j = 2, \cdots, k \end{aligned} \tag{2.13}$$

Il calcolo della $P(s_n)$ si effettua contando quanti s_n appartengono ad ogni sottointervallo e dividendo per N. La $P(s_n)$ diventa così un oggetto del tipo $\{P_i\}_{i=1,k}$.

Per la probabilità congiunta si costruisce una matrice J del tipo $k \times k$. L'elemento J_{ij} si riempe con il numero di volte che si conta una coppia (s_n, s_{n+T}) in cui $s_n \in \Delta_i$ e $s_{n+T} \in \Delta_j$ diviso per il numero di coppie $N-1$. Con queste premesse segue la formula per il calcolo della mutua informazione:

$$I(T) = \sum_{i=1}^{k}\sum_{j=1}^{k} J_{ij} \log_2\left(\frac{J_{ij}}{P_i P_j}\right) \tag{2.14}$$

sommando solo quando $P_i P_j \neq 0$.

2.4.2 Autocorrelazione

Un altro strumento importante per valutare le caratteristiche di memoria di una serie è l'autocorrelazione:

$$C(T) = \sum_{n}(s_n - \bar{s})(s_{n+T} - \bar{s}) \tag{2.15}$$

dove:

$$\bar{s} = \frac{1}{N}\sum_{n=1}^{N} s_n \tag{2.16}$$

è la media di s. Sappiamo che la perdita di memoria della serie è legata al primo attraversamento dello zero (*zero crossing*) della funzione di autocorrelazione, quindi si potrebbe scegliere T nelle vicinanze dell'attraversamento dello zero di $C(T)$. Nonostante anche questo sia un criterio assolutamente ragionevole, in alcuni esempi, non sembra funzionare bene come la scelta basata sul primo minimo della mutua informazione [1].

2.4.3 Falsi vicini

Si vuole determinare la dimensione intera globale del vettore delle coordinate ritardate in modo da spiegare completamente la struttura dell'attrattore. In altri termini si cerca per quali dimensioni non si verificano più incroci dell'orbita nello spazio Y_d. La dimensione minima che garantisce questo spiegamento viene detta dimensione di embedding e si indica con d_{E}. È intuitivo che una volta raggiunta una dimensione d_{E} per cui si riesce a spiegare completamente l'attrattore, le dimensioni $d > d_{\mathrm{E}}$ si comporteranno ancora così. In effetti la disuguaglianza vale per dati privi di rumore.
Dal teorema di embedding sappiamo che è sufficiente considerare $d_{\mathrm{E}} > 2d_{\mathrm{A}}$. Riguardo alla necessità di un valore di d_{E} non si può fare alcuna deduzione, se non osservare che in generale può bastare una dimensione minore.

Un aspetto cruciale da considerare è il fatto che in generale non si è a conoscenza della dimensione dell'attrattore d_{A} e i metodi per calcolarla richiedono una grande quantità di dati di cui solitamente non si è in possesso. Un metodo applicabile in una vasta casistica è quello dei falsi vicini [1]. L'algoritmo consiste nell'analisi dello spazio dei vettori $\mathbf{y}$ in modo da scoprire se due punti vicini rimangono tali passando ad uno spazio aumentato di una coordinata. Se accade ciò significa che i due punti sono dei vicini autentici anche nello spazio degli stati $\mathcal{S}$, altrimenti erano vicini solo perchè proiettati in uno spazio di dimensione troppo piccola. Procedendo iterativamente si dovrebbe raggiungere una dimensione per la quale non si trovano più falsi vicini e quindi nella quale si può spiegare completamente l'attrattore.

Formalmente, si costruisce il vettore $\mathbf{y}(k)$ nello spazio Y_d (d generico):

$$\mathbf{y}(k) = (s_k, s_{k+T}, s_{k+2T}, \cdots, s_{k+(d-1)T}) \tag{2.17}$$

con T suggerito dal criterio della mutua informazione.
Si cerca il vettore $\mathbf{y}^{NN}(k)$ più vicino ad $\mathbf{y}(k)$:

$$\mathbf{y}^{NN}(k) = \mathbf{y}(u) = (s_u, s_{u+T}, s_{u+2T}, \cdots, s_{u+(d-1)T}) \tag{2.18}$$

Serve un criterio per stabilire se i vettori $\mathbf{y}(k)$ e il suo più vicino in Y_d $\mathbf{y}^{NN}(k)$ sono vicini o lontani nel passaggio allo spazio Y_{d+1} aggiungendo una coordinata. Le componenti che vengono aggiunte ai vettori $\mathbf{y}(k)$ e $\mathbf{y}^{NN}(k)$ sono rispettivamente s_{k+dT} ed s_{u+dT}.
Risulta quindi utile il confronto fra $|s_{k+dT} - s_{u+dT}|$ e la distanza euclidea fra $\mathbf{y}(k)$ e $\mathbf{y}^{NN}(k)$ in Y_d. Se la distanza addizionale è grande rispetto a quella nello spazio Y_d, allora si è di fronte a due falsi vicini.
Traducendo in simboli, il quadrato della distanza euclidea fra $\mathbf{y}(k)$ e $\mathbf{y}^{NN}(k)$ in Y_d è:

$$R_d(k)^2 = \sum_{m=1}^{d} \left(s_{k+(m-1)T} - s_{u+(m-1)T}\right)^2 \tag{2.19}$$

mentre nello spazio Y_{d+1}:

$$\begin{aligned} R_{(d+1)}(k)^2 &= \sum_{m=1}^{d+1} \left(s_{k+(m-1)T} - s_{u+(m-1)T}\right)^2 \\ &= R_d(k)^2 + |s_{k+dT} - s_{u+dT}|^2 \end{aligned} \tag{2.20}$$

Si valuta se il rapporto:

$$\sqrt{\frac{R_{(d+1)}(k)^2 - R_d(k)^2}{R_d(k)^2}} = \frac{|s_{k+dT} - s_{u+dT}|}{R_d(k)} \tag{2.21}$$

è maggiore di una certa soglia per etichettare il vettore più vicino a $\mathbf{y}(k)$ come falso.
Nel caso in cui l'attrattore sia sufficientemente popolato la scelta della soglia non è critica, nel senso che non cambia in maniera sostanziale la classificazione come falso vicino. In letteratura si consiglia un valore della soglia intorno a 15 [1].

Si ripete la procedura per tutti i vettori $\mathbf{y}(k)$ e si valuta la percentuale di falsi vicini. Effettuando il calcolo a partire da $d = 1$ si giunge ad una dimensione per cui la percentuale è zero nel caso di dati privi di rumore e sufficientemente campionati.

Nell'effettuare l'analisi in dimensioni elevate ci si scontra con il fatto che il volume occupato dall'attrattore va ad addensarsi nella periferia dello spazio Y_d, visto che i volumi crescono con una potenza di d. Accade così che due punti considerati vicini siano lontani per motivi topologici relativi allo spazio di osservazione.
A questo prosito si aggiunge un ulteriore condizione alla precedente.
Definendo:

$$R_{\mathrm{A}} = \frac{1}{N}\sum_{k=1}^{N}(s_k - \bar{s})^2 \tag{2.22}$$

il raggio nominale dell'attrattore, se la quantità:

$$\frac{|s_{k+dT} - s_{u+dT}|}{R_{\mathrm{A}}} \tag{2.23}$$

è maggiore di un valore di circa due si etichetta $\mathbf{y}^{NN}(k)$ come falso vicino. In altre parole si vuole evitare che $|s_{k+dT} - s_{u+dT}|$ sia piccolo rispetto alla distanza in dimensione d ma sia dell'ordine del diametro dell'attrattore e ciò può accadere quando d è grande.

2.5 Previsione e regressione

Dal nostro punto di vista il teorema di embedding permette di trasformare il problema della predizione in un problema di regressione. Più precisamente il compito che verrà assegnato ai learners sarà quello di approssimazione di una funzione $\mathbb{R}^d \rightarrow \mathbb{R}$. Grazie al teorema di embedding, si può pensare di ricostruire la struttura seguita dalla dinamica del sistema. L'utilità di effettuare questa ricostruzione risiede nel fatto che, una volta noto come si sviluppa la dinamica, è possibile, almeno in linea di principio, poter prevedere cosa accadrà nel futuro.
In generale si è in possesso di una serie di N valori $\{s_k\}_{k=1,\cdots,N}$ relativa ad una grandezza del sistema in studio. Il problema della predizione è quello di

associare ai vettori:

$$\mathbf{y}(n) = (s_n, s_{n+1}, s_{n+2}, \cdots, s_{n+d-1}) \tag{2.24}$$

i relativi s_{n+d}. Il valore di d è quello suggerito dall'analisi dei falsi vicini, mentre la scelta di costruire i vettori $\mathbf{y}(n)$ con osservazioni contigue di s, risiede nel fatto che si vuole sfruttare tutta l'informazione posseduta dalla dinamica fino al valore da prevedere. D'ora in avanti le $l = N - d + 1$ coppie $\{(\mathbf{y}(k), s_{k+d})\}_{k=1,\cdots,l}$ verranno indicate per comodità notazionale $\{(\mathbf{x}_k, y_k)\}_{k=1,\cdots,l} \in \mathbb{R}^d \times \mathbb{R}$.
Il compito di regressione viene assegnato alle macchine ad apprendimento automatico.

2.6 Serie storiche fisiche

Nei problemi reali ci si scontra con alcuni problemi. Il primo aspetto da tenere presente è che la serie storica sarà ottenuta attraverso un processo di misura che in qualche modo aggiungerà un termine di rumore ad essa. Il fatto che non sia possibile conoscere il valore di una grandezza con precisione arbitraria, si riflette nel fatto che, per i sistemi caotici, non sia possibile effettuare una predizione più avanti di un certo intervallo temporale. Infatti l'evoluzione a partire da due stati del sistema molto vicini, porta dopo poco tempo ad evoluzioni completamente differenti. Volendo ricostruire la dinamica del sistema a partire dalla serie storica $\{s_k\}_{k=1\cdots N}$, la conoscenza dei valori reali con precisione arbitraria consentirebbe di conoscere con precisione arbitraria lo stato del sistema. Nel caso reale dunque si possiede lo stato del sistema con una indeterminazione rappresentabile attraverso un'ipersfera in $\mathcal{S}$, le cui dimensioni crescono in maniera esponenziale. Quantitativamente l'espandersi dell'ipersfera è descritta dagli esponenti di Lyapunov, e pone un vincolo alla possibilità di prevedere cosa accadrà molto avanti nel tempo [24].

Pensando anche a serie generate sinteticamente attraverso calcolatori, si va incontro ad "errori di misura" dovuti all'allocazione in memoria dei dati in un numero finito di bit. Questo troncamento su cifre molto poco significative porta comunque ad una incertezza nella conoscenza dello stato del sistema che viene amplificata nel tempo secondo gli esponenti di Lyapunov.

Per quanto riguarda i disturbi esterni al sistema, anch'essi per sistemi caotici, possono provocare un cambiamento radicale nell'evoluzione non imputabile al sistema stesso.

Infine un altro aspetto fondamentale è *l'ergodicità*, ovvero il fatto che le caratteristiche statistiche di diversi spezzoni temporali della serie siano le stesse (i.i.d. pag. 4). All'interno di una serie molto lunga infatti è possibile che l'evoluzione del passato non possa dare alcuna forma di esperienza per la previsione di ciò che accadrà nel futuro proprio perché generata da un processo non ergodico. È dunque necessario che durante il processo di misura, il sistema possa considerarsi ergodico.

Siccome le serie fisiche presentano delle caratteristiche di complessità che possono rendere il problema della predizione difficile, può risultare utile poter prevedere l'andamento di un loro trend. Il metodo che si è utilizzato in questa tesi è basato sull'SSA e viene illustrato nella prossima sezione.

2.7 Analisi di spettro singolare - SSA

L'*analisi di spettro singolare* (*Singular Spectrum Analysis SSA*) viene utilizzata da anni come tecnica di analisi ed elaborazione di segnali digitali (per esempio in campo geofisico [30] [19]). Essa si basa sull'*analisi delle componenti principali* [15] (*Principal Component Analysis PCA*) nello spazio dei vettori delle coordinate ritardate. La PCA è una tecnica che permette di ottenere, a partire da una distribuzione di punti in uno spazio ad m dimensioni,

le m direzioni principali[2] lungo cui si dispongono i dati. In questo modo è possibile descrivere la distribuzione in termini delle direzioni principali, le quali danno informazioni utili sulle caratteristiche dei dati in studio. Per una trattazione esaustiva consultare [15].

A partire dalla serie storica $\{s_i\}_{i=1,\cdots,n}$ a media nulla si costruiscono $N = n - m + 1$ vettori delle coordinate ritardate $\mathbf{v}_j$:

$$\begin{aligned} \mathbf{v}_1 &= (s_1, s_2, \cdots, s_m) \\ \mathbf{v}_2 &= (s_2, s_3, \cdots, s_{m+1}) \\ &\cdots \\ \mathbf{v}_N &= (s_{n-m+1}, s_{n-m+2}, \cdots, s_n) \end{aligned} \tag{2.25}$$

Si costruisce la matrice di covarianza, avente la struttura di una matrice di Toeplitz:

$$T = \begin{pmatrix} c_0 & c_1 & c_2 & \cdots & c_{m-1} \\ c_1 & c_0 & c_1 & \cdots & c_{m-2} \\ c_2 & c_1 & c_0 & \cdots & c_{m-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{m-1} & c_{m-2} & c_{m-3} & \cdots & c_0 \end{pmatrix} \tag{2.26}$$

dove c_j è la covarianza di s con se stessa ritardata di j. Esistono diversi modi per stimare i vari c_j e fra questi uno dei più usati è quello di Yule-Walker [15]:

$$c_j = \frac{1}{n-j} \sum_{i=1}^{n-j} s_i s_{i+j} \tag{2.27}$$

Per T si calcolano gli autovalori e gli autovettori, i quali si ordinano sulla base dell'ordinamento decrescente dei relativi autovalori. La matrice T è simmetrica, quindi i sui autovalori λ_k sono reali e la base in $\mathbb{R}^m$ costituita dagli m autovettori $\mathbf{e}^k$ normalizzati è ortonormale. Inoltre, per sua costruzione,

[2]Solitamente si è interessati ad una parte di esse tale da descrivere i dati in maniera compatta.

risulta semidefinita positiva, ovvero $\lambda_k \geq 0 \ \forall k = 1, \cdots, m$ [15].
Ognuno dei λ_k esprime la varianza della distribuzione nella direzione individuata dal relativo $\mathbf{e}^k$ e solitamente si normalizza rispetto alla somma di tutti gli autovalori:

$$\nu_k = \frac{\lambda_k}{\sum_{k=1}^{m} \lambda_k}, \qquad k = 1, \cdots, m \tag{2.28}$$

Una certa ν_k contiene la percentuale di varianza spiegata dal relativo autovalore λ_k e per questo viene detta *varianza spiegata*.

Si possono riscrivere i vettori $\mathbf{v}_j$ in termini degli autovettori normalizzati $\mathbf{e}^k$:

$$\mathbf{v}_j = \sum_{k=1}^{m} \left(\mathbf{v}_j \cdot \mathbf{e}^k\right) \mathbf{e}^k \tag{2.29}$$

Il prodotto scalare fra $\mathbf{v}_j$ e $\mathbf{e}^k$ è noto come componente principale a_j^k:

$$\mathbf{v}_j = \sum_{k=1}^{m} a_j^k \mathbf{e}^k \tag{2.30}$$

Questa scrittura permette chiaramente di riottenere la serie originale. Per convincersi di questo basta osservare che ogni elemento s_i della serie è contenuto in uno o più vettori $\mathbf{v}_j$. In particolare per ogni s_i con $i \geq m$ lo stesso s_i è esattamente in m vettori $\mathbf{v}_j$. Il primo elemento s_1 sarà solo in $\mathbf{v}_1$, s_2 in $\mathbf{v}_1$ e in $\mathbf{v}_2$ e così via. Dunque per riottenere un certo s_i basterà estrarlo da uno dei vari $\mathbf{v}_j$ ai quali appartiene oppure, come si fa in filtraggio digitale, mediando su tutti i vettori di appartenenza.

Per poter utilizzare in predizione l'SSA bisogna fare attenzione a come ricostruire le varie onde. Siccome non è lecito utilizzare l'informazione degli elementi della serie successivi al valore da ricostruire, la costruzione può avvenire in un solo modo. Nella ricostruzione del valore s_m si potrà utilizzare il vettore $\mathbf{v}_1$, poiché questi ha come ultima coordinata proprio s_m, e così via fino ad s_n.
Quindi:

$$s_{m+i-1} = \sum_{k=1}^{m} \left(\mathbf{v}_i \cdot \mathbf{e}^k\right) e_m^k \tag{2.31}$$

dove e_m^k è la m-esima componente dell'autovettore $\mathbf{e}^k$. In questo modo le componenti principali $\mathbf{v}_i \cdot \mathbf{e}^k$ contengono solo informazione del passato. Per quanto riguarda i primi valori di s, quelli cioè che vanno da 1 ad $m-1$, non c'è modo di ricostruirli senza mettere in gioco dell'informazione del futuro. Per questo motivo si è deciso di scartare questi valori, riottenendo solo la serie $\{s_i\}_{i=m,\cdots,n}$.
Riassumendo, l'SSA consente di scomporre ogni elemento della serie originale e quindi della sua totalità, nella somma di m componenti indipendenti fra loro[3].

La scelta di m assume importanza nella possibilità di risolvere oscillazioni di vario periodo. Oscillazioni di bassa frequenza possono essere risolte solo scegliendo m elevato ma d'altra parte ciò non consente di apprezzare sufficientemente quelle di alta frequenza. Quando la finestra temporale è molto più piccola di n i risultati dell'SSA dipendono debolmente da m [15] e solitamente si sceglie $m \simeq n/4$. Volendo risolvere oscillazioni di frequenza f e larghezze spettrali di $2\delta f$ Vautard suggerisce una scelta di m che rispetti la seguente condizione [31]:

$$\frac{1}{f} \leq m \leq \frac{1}{2\delta f} \tag{2.32}$$

Sperimentalmente si evidenzia che una finestra di lunghezza m permette di risolvere oscillazioni con periodi nel range che va da $m/5$ ad m [15].

L'equazione 2.31 permette di ottenere m onde indipendenti che sommate consentono di riottenere la serie originale. Un altro punto di vista è quello di considerare le varie onde come versioni filtrate della serie originale, ottenute applicando diversi filtri non lineari[4] ad essa.

[3]Esse derivano da autovettori ortogonali a due a due quindi indipendenti.

[4]Il procedimento che porta al calcolo delle componenti principali è non lineare.

2.7.1 Estrazione di trend con SSA

L'indicazione fornita dalla varianza spiegata permette di analizzare quale sia la quantità di informazione portata dalla descrizione in termini di ogni autovettore. In altre parole è possibile capire quanta informazione si perde ricostruendo la serie originale utilizzando una parte soltanto degli autovettori. Ciò è molto utile qualora si volesse valutare l'andamento generale seguito da una serie. Si può infatti ragionare sul fatto che considerando una parte degli autovettori si spiega gran parte della varianza totale associata alla distribuzione dei dati, ovvero si perde una piccola parte dell'informazione posseduta dalla serie originale. Questo corrisponde ad estrarre dalla serie un trend. Inoltre le onde con maggiore contenuto informativo saranno meno intaccate, in termini di potenza, dal rumore presente.
Il vantaggio di estrarre il trend di una serie è che esso possiede un andamento molto più regolare della serie originale, poiché ne è una versione filtrata e quindi in linea di principio più semplice da prevedere. D'altra parte, questo procedimento fa sì che si perda una parte di informazione contenuta nella serie originale. Quale sia il numero di onde da sommare si può dedurre direttamente dall'analisi della varianza spiegata. Le onde per cui è alta sono buone candidate ad essere scelte per far parte della composizione del trend.

Capitolo 3

Base learner

In questo capitolo vengono descritte le macchine d'apprendimento base utilizzate nella tesi: le reti neurali e le macchine a supporto vettoriale (SVM). Nonostante le molte somiglianze formali, i metodi di apprendimento seguono idee differenti. La differenza sostanziale sta nella scelta del tipo di loss function che per le reti neurali è quadratica, mentre per le SVM è di tipo modulo con una zona di insensitività e verrà descritta nella sezione dedicata. Questi due base learner appartengono alla categoria dei metodi globali, nel senso che vengono addestrati su tutto il data base. Verrà inoltre fatto cenno ai metodi locali che vengono addestrati su un sottoinsieme di interesse del data base.

3.1 Concetti preliminari

Grazie alle nozioni descritte nel secondo capitolo, possiamo affermare che i metodi predittivi consistono nella ricerca di un'applicazione $F : \mathbb{R}^d \to \mathbb{R}$ che rappresenta una curva γ in $\mathbb{R}^{d+1}$ in grado di approssimare la distribuzione dei punti $(\mathbf{x}_k, y_k)$. Bisogna dunque risolvere un problema di regressione e ciò può essere fatto attraverso macchine ad apprendimento automatico. Per ottenere un buon modello di predittore, si divide il database totale delle coppie $(\mathbf{x}_k, y_k)$ in due parti:

- training set, costituito dalle associazioni che si assume di conoscere;
- test set, composto da coppie di cui si conosce l'associazione, ma su cui si vuole valutare la capacità del predittore di aver appreso e generalizzato la regola che descrive la corrispondenza;

Formalmente si effettua una predizione $\hat{y}_u$ per tutti i vettori $\mathbf{x}_u$ del test set sulla base dagli esempi forniti dal training set e si quantifica la bontà della generalizzazione valutando il valore $RMSE$ (root mean square error):

$$RMSE = \sqrt{\frac{1}{N_{test}} \sum_{u=1}^{N_{test}} (y_u - \hat{y}_u)^2} \tag{3.1}$$

Nel caso in cui fosse $\hat{y}_u = y_u \ \forall u = 1, \cdots, N_{test}$ risulterebbe $RMSE = 0$.
Per quanto riguarda il test set si è scelto di costruirlo in maniera che fosse costituito di associazioni cronologicamente successive al training set.
Un altro modo di valutare la bontà di generalizzazione, utilizzata diffusamente, è il valore $NMSE$ (normalized mean square error):

$$NMSE = \frac{\varepsilon^2}{\sigma^2} = \frac{\frac{1}{N_{test}} \sum_{u=1}^{N_{test}} (y_u - \hat{y}_u)^2}{\frac{1}{N_{test}} \sum_{u=1}^{N_{test}} (y_u - \bar{y})^2} \tag{3.2}$$

dove $\bar{y}$ è:

$$\bar{y} = \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} y_i \tag{3.3}$$

Sostanzialmente si normalizza l'errore $RMSE$ rispetto alla varianza di y per il test set.
In questo lavoro si è preferito utilizzare la valutazione secondo il $RMSE$. Riassumendo si costruiscono vari modelli sulla base del training set e si sceglie il migliore valutando l'$RMSE$ sul test set. Nella tesi si è anche considerato l'ME, definito come:

$$ME = \max_{u=1,\cdots,Ntest} |y_u - \hat{y}_u| \tag{3.4}$$

ovvero il massimo del valore assoluto dell'errore sul test set.

3.2 Reti neurali

Con rete neurale [9] si intende una rete composta da unità fondamentali dette neuroni, le quali sono collegate fra loro e vengono caratterizzate tramite un valore numerico che ne identifica lo stato. Formalmente lo stato ξ_i di ogni neurone si esprime come combinazione lineare dello stato degli altri:

$$\xi_i = h_i \left(\sum_j w_{ij}\xi_j - \vartheta_i \right) \tag{3.5}$$

Successivamente a questa combinazione viene applicata una funzione h_i, chiamata *funzione di attivazione*. Le forme tipiche delle h_i sono:

$$\begin{aligned} h(x) &= \frac{e^x}{1+e^x} && \text{Funzione logistica} \\ h(x) &= \tanh(x) && \text{Tangente iperbolica} \\ h(x) &= \Theta(x) = \begin{cases} 0 & \text{se} \quad x \leq 0 \\ 1 & \text{se} \quad x > 0 \end{cases} && \text{Funzione di Heavyside} \end{aligned} \tag{3.6}$$

I pesi secondo cui ogni neurone contribuisce allo stato degli altri sono contenuti nella *matrice delle connessioni* $\{w_{ij}\}_{i,j}$. In particolare una non connessione fra il neurone p e quello q si traduce nel fatto che $w_{pq} = 0$

Riguardo a ϑ_i, comunemente denominata soglia o termine di bias, si può inglobare direttamente nella matrice delle connessioni. Supponendo che nell'eq. 3.5 l'indice j inizi da 1, si pone $w_{i0} = \vartheta_i$ e $\xi_0 = -1$. In questo modo si riscrive l'eq. 3.5 con una notazione più compatta:

$$\xi_i = h_i \left(\sum_{j=0,1,\cdots} w_{ij} \xi_j \right) \tag{3.7}$$

Esiste una classe di reti, di cui ci si è occupati in questo lavoro, chiamati *percettroni* o *reti feed-forward* (*Multi Layer Perceptron MLP*). Esse sono composte da strati di neuroni all'interno dei quali non vi sono collegamenti e in cui l'informazione viaggia in un solo senso. Lo strato al quale vengono forniti gli input si chiama strato di input e non viene contato come strato. Quello che si conta è infatti il numero di matrici di connessione fra strati. Gli strati che separano lo strato di ingresso da quello di uscita si chiamano strati nascosti.

I percettroni ad uno strato con un neurone di uscita, anche noti come *adaline* (3.1) sono descritti dall'equazione:

$$O = h \left(\sum_j w_{ij} I_j \right) \tag{3.8}$$

Per i percettroni a due strati con un neurone di uscita ad attivazione lineare e con gli altri ad attivazione logistica tutti uguali (3.2), la forma assunta dallo stato del neurone di uscita sarà:

$$O = \sum_i w_i^{(h \to o)} h_i \left(\sum_j w_{ij}^{(i \to h)} I_j \right) \tag{3.9}$$

dove h_i è appunto la funzione logistica (eq. 3.6).

Sussiste un utile teorema, chiamato di approssimazione universale [9], che garantisce che reti descritte dall'eq. 3.9 possono approssimare qualsiasi funzione continua uniformemente su compatti. Non esiste un teorema che

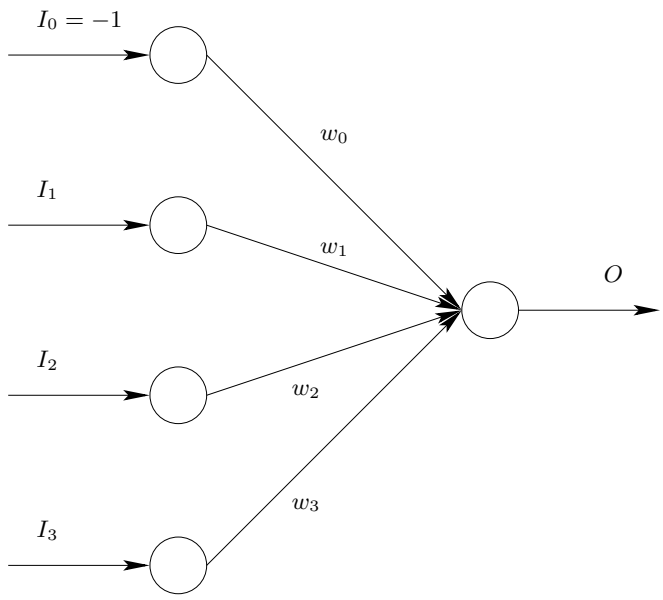


Figura 3.1: Adaline con tre neuroni di ingresso. La soglia è contenuta in w_0.

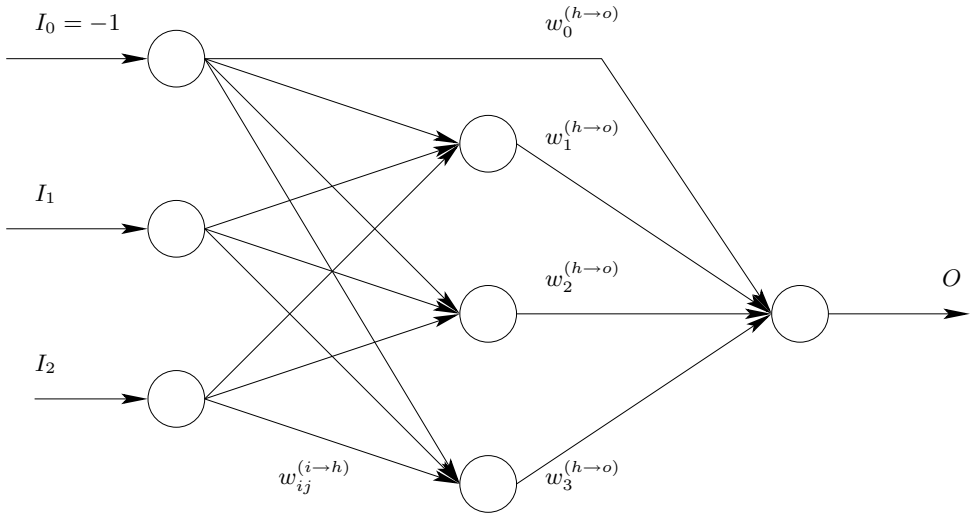


Figura 3.2: Percettrone a due strati con due neuroni di ingresso, tre neuroni nello strato nascosto ed uno di uscita.

possa garantire quale sia la scelta migliore del numero di neuroni nello strato nascosto.

L'utilità del teorema di approssimazione sarebbe vana se non si conoscesse un modo per costruire il funzionale nell'eq. 3.9. Risulta infatti necessario ricavare in qualche modo quali debbano essere i pesi da assegnare alle varie connessioni.

Questo è proprio lo scopo dell'addestramento, cioè trovare la matrice delle connessioni a partire dagli esempi contenuti nel training set. A questo proposito l'algoritmo di apprendimento venne scoperto più volte ma si affermò definitivamente nel 1986 con il lavoro di Rumelhart, Hinton e Williams (Nature, vol:323, pagg: 533–536). L'algoritmo, per le sue caratteristiche, viene chiamato di *backpropagation* e successivamente sono state proposte varianti tese a migliorarne le prestazioni.

L'idea centrale è quella di minimizzare nello spazio dei pesi, il funzionale:

$$E(\mathbf{w}) = \sum_{i=1}^{N_{training}} (y_i - O_i)^2 \tag{3.10}$$

il quale altri non è che il rischio empirico di eq. 1.2 nel caso di funzione di perdita quadratica. L'algoritmo di backpropagation consiste nel fornire i vari esempi in ingresso, calcolare il valore di uscita e confrontarlo con il valore obiettivo, ovvero ciò che si vuole fare apprendere alla rete quando in input ha quel particolare ingresso. In base all'errore commesso si effettua una variazione dei pesi fra lo strato di uscita e quello nascosto e successivamente fra quest'ultimo e quello di ingresso; per questo motivo prende il nome di backpropagation (per una derivazione dell'algoritmo di può vedere [9]).

Esistono due modi di utilizzare backpropagation, chiamati *by pattern* e *by epoch* e si riferiscono alla fatto che i pesi possono essere aggiornati alla presentazione di ogni singolo esempio oppure alla presentazione di tutto il training set. La fase di training della rete consiste nel presentare tutti gli esempi del training set molte volte, in modo da minimizzare E (eq. 3.10).

L'inizializzazione dei valori delle matrici delle connessioni va fatta con numeri piccoli e random. Con piccoli si intende intorno alla sensibilità messa a disposizione dall'allocazione di essi nella memoria di un calcolatore.

Le varianti che sono state proposte negli anni sono tese a migliorare le prestazioni in termini di velocità di apprendimento del problema e di generalizzazione. Infatti spingere l'algoritmo alla convergenza significa apprendere molto bene gli esempi del training set senza curarsi della capacità di generalizzare. Serve dunque un criterio per decidere quando interrompere l'apprendimento (criterio di *early stopping*). L'idea è quella di non addestrare più la rete appena l'errore sul test set inizia ad aumentare. Infatti nella fase iniziale di apprendimento, sia l'errore sul training set che sul test set diminuisce, mentre da un certo punto in poi la rete inizierà ad imparare troppo bene gli esempi del training set e peggiorerà la generalizzazione sul test set[1].

Un altro problema a cui si è fatto fronte è il fatto che le unità logistiche, qualora raggiungessero zone di lavoro lontane dalla zona quasi lineare, possono difficilmente cambiare il loro stato verso la zona quasi lineare (*saturazione*). Questo segue direttamente dal fatto che la derivata della funzione logistica è molto piccola all'allontanarsi dall'origine. Per ovviare a questo problema si cerca di minimizzare l'eq. 3.10 sommata ad un termine di regolarizzazione o meglio si aggiunge un termine di decadimento dei pesi:

$$E(\mathbf{w}) = \sum_{i=1}^{N_{training}} (y_i - O_i)^2 + \lambda \sum_{ij} w_{ij}^2 \tag{3.11}$$

In questo modo si vuole evitare che nella fase di addestramento si arrivi ad avere pesi grandi che possono provocare valori di input tali da mandare il neurone in una zona di lavoro poco utile. La scelta di λ si riflette in maniera naturale nel compromesso fra complessità e generalizzazione del MLP. Infatti l'introduzione di questo decadimento dei pesi equivale a diminuire la complessità della rete.

[1] Questo in letteratura è noto come problema di *overfitting*.

Per concludere, i due aspetti che hanno convinto nella scelta dei percettroni sono:

- il teorema di approssimazione universale;
- la disponibilità di un algoritmo per l'apprendimento efficiente.

3.3 Macchine a supporto vettoriale - SVM

Inizialmente sviluppate per la risoluzione di problemi di classificazione, le *macchine a supporto vettoriale* (*SVM*), si possono applicare anche a problemi di regressione [23].
Le SVM permettono di costruire funzioni della forma:

$$f(\mathbf{x}, \mathbf{w}) = \sum_{i=1}^{N} w_i \varphi_i(\mathbf{x}) \tag{3.12}$$

dove le $\varphi_i(\mathbf{x})$ vengono chiamate *basis functions o kernels* [23]. In altri termini le SVM sono adaline in cui all'ingresso vengono applicate le funzioni di kernel, con un diverso algoritmo di apprendimento rispetto alle reti neurali. Anche qui il termine di bias è contenuto all'interno del vettore $\mathbf{w}$ e il problema dell'apprendimento si configura come la ricerca del vettore dei pesi $\mathbf{w}$. L'equazione è un modello di regressione non lineare poiché in generale la risultante ipersuperficie nello spazio n-dimensionale degli $\mathbf{x}$ è non lineare.

Come primo obiettivo si introduce il problema di regressione lineare per poi utilizzare i concetti acquisiti per quella nonlineare. Esplicitando il termine di bias, si ha a che fare con funzioni del tipo:

$$f(\mathbf{x}, \mathbf{w}) = \mathbf{w} \cdot \mathbf{x} + b \tag{3.13}$$

Vapnik ha introdotto un tipo di funzione di perdita generale, avente una zona di insensibilità larga ε (fig. 3.3):

$$|y - f(\mathbf{x}, \mathbf{w})|_\varepsilon = \begin{cases} 0 \quad \text{se} \quad |y - f(\mathbf{x}, \mathbf{w})| \leq \varepsilon \\ |y - f(\mathbf{x}, \mathbf{w})| - \varepsilon \quad \text{altrimenti} \end{cases} \tag{3.14}$$

L'eq. 3.14 fa sì che l'errore sia zero quando la differenza fra la predizione $f(\mathbf{x}, \mathbf{w})$ e il valore osservato è minore di ε e pari alla distanza fra valore misurato e la circonferenza di raggio ε centrata sulla predizione altrimenti. La funzione di perdita così costruita è come se crease un ε-tubo intorno alla $f(\mathbf{x}, \mathbf{w})$ (fig. 3.4). Si introduce il rischio empirico sulla base della funzione

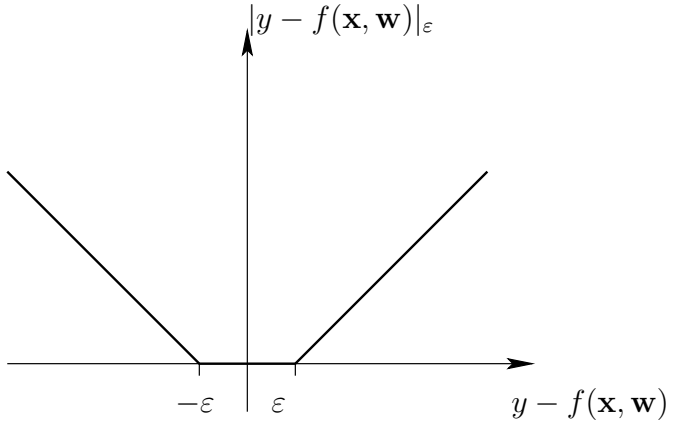


Figura 3.3: Funzione di perdita di Vapnick (eq. 3.14)

di perdita di Vapnick:

$$R^{\varepsilon}_{\text{emp}} = \frac{1}{l} \sum_{i=1}^{l} |y_i - \mathbf{w} \cdot \mathbf{x}_i - b|_\varepsilon \tag{3.15}$$

Secondo i concetti legati al compromesso fra complessità e approssimazione, l'obiettivo dell'apprendimento è minimizzare una combinazione lineare di $R^{\varepsilon}_{\text{emp}}$ e $\| \mathbf{w} \|^2$, quindi:

$$R = \frac{1}{2} \| \mathbf{w} \|^2 + C \sum_{i=1}^{l} |y_i - \mathbf{w} \cdot \mathbf{x}_i - b|_\varepsilon \tag{3.16}$$

Da notare nuovamente la somiglianza con il MLP (in particolare eq. 3.11) nel caso di funzione costo con il decadimento dei pesi. Un alto valore della costante C penalizza maggiormente valori del training set al di fuori del tubo,

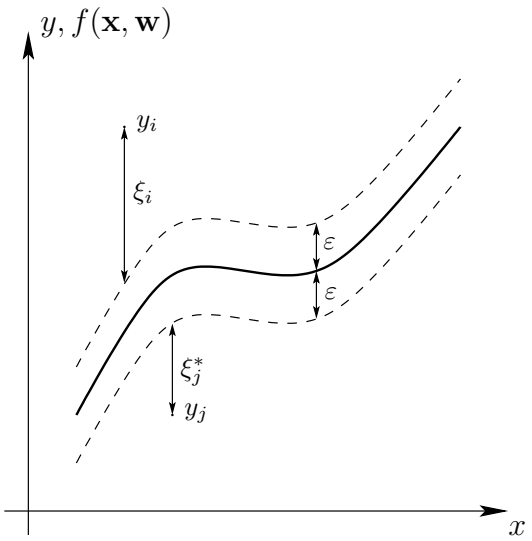


Figura 3.4: Attorno alla funzione $f(\mathbf{x}, \mathbf{w})$ (linea continua) viene creato un ε-tubo (linee tratteggiate)

privilegiando la capacità di approssimazione rispetto a quella di generalizzazione. Quindi C assume il significato di parametro di regolarizzazione.
Per i dati del training set all'esterno dell'ε tubo si definiscono:

- $\xi_i = |y_i - f(\mathbf{x}, \mathbf{w})| - \varepsilon$ per dati al di sopra dell'ε tubo;
- $\xi_i^* = |y_i - f(\mathbf{x}, \mathbf{w})| - \varepsilon$ per dati al di sotto dell'ε tubo.

Quindi minimizzare R equivale a minimizzare:

$$R_{\mathbf{w},\xi_i,\xi_i^*} = \frac{1}{2} \parallel \mathbf{w} \parallel^2 + C \left(\sum_{i=1}^{l} \xi_i + \sum_{i=1}^{l} \xi_i^* \right) \tag{3.17}$$

sottoposto ai vincoli (fig. 3.4):

$$\begin{aligned} & y_i - \mathbf{w} \cdot \mathbf{x}_i - b \leq \varepsilon + \xi \quad && i = 1, \cdots, l \\ & \mathbf{w} \cdot \mathbf{x}_i + b - y_i \leq \varepsilon + \xi^* \quad && i = 1, \cdots, l \\ & \xi \geq 0 && i = 1, \cdots, l \\ & \xi^* \geq 0 && i = 1, \cdots, l \end{aligned} \tag{3.18}$$

La risoluzione avviene tramite la tecnica dei moltiplicatori di Lagrange, che conduce ad un problema di minimizzazione quadratica [23].
Si introduce la funzione lagrangiana:

$$\begin{aligned} L_{\mathrm{p}}(\mathbf{w}, b, \xi_i, \xi_i^*, \alpha_i, \alpha_i^*, \beta_i, \beta_i^*) &= R_{\mathbf{w},\xi,\xi^*} - \sum_{i=1}^{l} \alpha_i(\mathbf{w}\cdot\mathbf{x}_i + b - y_i + \varepsilon + \xi_i) + \\ &- \sum_{i=1}^{l}(\beta_i^*\xi_i^* + \beta_i\xi_i) \end{aligned} \tag{3.19}$$

la quale va minimizzata rispetto a $\mathbf{w}$, b, ξ_i e ξ_i^* e massimizzata rispetto ai moltiplicatori α_i, α_i^*, β_i, β_i^*. Il problema può essere risolto nello spazio duale [23]. Applicando la condizione di Karush-Kuhn-Tucker il problema si traduce nella massimizzazione di:

$$\begin{aligned} L_{\mathrm{d}}(\alpha_i, \alpha_i^*) &= -\varepsilon\sum_{i=1}^{l}(\alpha_i^* + \alpha_i) + \sum_{i=1}^{l}(\alpha_i^* - \alpha_i)y_i \\ &- \frac{1}{2}\sum_{i,j=1}^{l}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)\mathbf{x}_i\cdot\mathbf{x}_j \end{aligned} \tag{3.20}$$

soggetto ai vincoli:

$$\sum_{i=1}^{l}\alpha_i^* = \sum_{i=1}^{l}\alpha_i, \qquad i = 1, \cdots, l \tag{3.21}$$

$$0 \le \alpha_i^* \le C, \qquad i = 1, \cdots, l \tag{3.22}$$

$$0 \le \alpha_i \le C, \qquad i = 1, \cdots, l \tag{3.23}$$

Rispetto al compito di classificazione, il numero di moltiplicatori è doppio poiché ci sono sia gli l α_i che gli l α_i^*. Risolvendo il problema di minimizzazione quadratica si ottiene il miglior iperpiano di regressione.

$$\mathbf{z} = f(\mathbf{x}, \mathbf{w}) = \mathbf{w}\cdot\mathbf{x} + b \tag{3.24}$$

Tornando al problema di regressione non lineare si risolve applicando la regressione lineare nello spazio dei features. Si costruisce cioè un'applicazione

$\mathbf{\Phi} : \mathbb{R}^n \to \mathbb{R}^f$ che associa ad ogni vettore $\mathbf{x} \in \mathbb{R}^n$ un vettore $\mathbf{z}$ nello spazio dei features F di dimensioni maggiori ($f > n$) e in questo nuovo spazio si tenta di applicare l'algoritmo di regressione lineare. La speranza è che nello spazio F sia possibile ricavare un iperpiano di regressione; questi, riportato nello spazio degli $\mathbf{x}$, sarà in generale una ipersuperficie non lineare.

Formalmente, dunque si ha:

$$\mathbf{x} \in \mathbb{R}^n \to \mathbf{z} = (a_1\phi_1, a_1\phi_1, \cdots, a_n\phi_n) \in \mathbb{R}^f \tag{3.25}$$

L'applicazione $\mathbf{\Phi}$, nella funzione lagrangiana L_d, fa nascere il prodotto scalare:

$$\mathbf{z}_i \cdot \mathbf{z}_j = \mathbf{\Phi}(\mathbf{x}_i) \cdot \mathbf{\Phi}(\mathbf{x}_j) \tag{3.26}$$

Questi viene chiamata kernel ed è una funzione dei vettori di input $\mathbf{x}_i$.

$$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{\Phi}(\mathbf{x}_i) \cdot \mathbf{\Phi}(\mathbf{x}_j) \tag{3.27}$$

L'introduzione della funzione kernel porta a vantaggi di calcolo, poiché consente di non dover effettuare prodotti scalari fra vettori ad elevate dimensioni e permette il passaggio ad uno spazio F di dimensione infinita [23]. In più non ci si deve preoccupare di conoscere la forma esplicita di $\mathbf{\Phi}$.
Le funzioni K che vengono utilizzate sono di tipo polinomiale di grado p e gaussiano:

$$\begin{aligned} K(\mathbf{x}_i, \mathbf{x}_j) &= (\mathbf{x}_i \cdot \mathbf{x}_j + b_0)^p \\ K(\mathbf{x}_i, \mathbf{x}_j) &= e^{-\gamma|\mathbf{x}_i - \mathbf{x}_j|^2} \end{aligned} \tag{3.28}$$

L'applicazione di questi kernel introduce ulteriori parametri come il grado p e il termine noto b_0 nel caso polinomiale o la larghezza della gaussiana legata all'inverso di γ nel caso gaussiano[2].

Per un numero di coppie del training set maggiore del migliaio, il problema di ottimizzazione quadratica richiede una capacità di calcolo non ancora

[2]Un valore alto di γ corrisponde ad una gaussiana stretta.

disponibile [23]. Sono stati così proposti alcuni metodi per scomporre il problema o per raggiungere soluzioni approssimate.

Il confronto delle prestazioni per piccoli database fra reti neurali e SVM pone in prima posizione SVM. Per database grandi hanno prestazioni confrontabili [23].

3.4 Metodi locali

A differenza dei metodi globali, in cui si cerca una funzione in grado di approssimare i punti del training set, in quelli locali si osserva lo sviluppo della dinamica localmente. In altre parole, quando si vuole effettuare una predizione a partire da un certo vettore $\mathbf{x}_u$, si analizza cosa sia successo all'evoluzione dei suoi k più vicini, traendone un'informazione sul suo possibile sviluppo.
In questo caso, dunque, per ogni vettore del test set si cercano all'interno del training set i k più vicini e si memorizza come si sono evoluti. Indicando con $\mathbf{x}_u^N(v_i)$ con $i = 1, \cdots, k$ e $\mathbf{v}$ il vettore degli indici dei k vicini l'i-esimo vicino a $\mathbf{x}_u$, essi evolveranno nei vettori $\mathbf{x}_u^N(v_i + 1)$.
In predizione si è interessati all'ultima coordinata degli $\mathbf{x}_u^N(v_i + 1)$, ovvero $y(v_i+1)$ che è possibile combinare in diversi modi. Si citano i due che si sono utilizzati:

- si mediano i k valori $y(v_i + 1)$ (metodo KNN1);
- utilizzando le coppie $(\mathbf{x}_u^N(v_i), y(v_i + 1))_{i=1,\cdots,k}$ si costruisce un'applicazione lineare $\mathbb{R}^d \to \mathbb{R}$ (metodo KNN2);

Il metodo KNN1 consiste nel generare una predizione per ogni vettore $\mathbf{x}_u$ del test set:

$$\hat{y}_u = \frac{1}{k} \sum_{i=1}^{k} y(v_i + 1) \tag{3.29}$$

ovvero mediando l'evoluzione dei k vicini.

Il metodo KNN2 si basa sull'idea che localmente la dinamica può essere pensata descritta da un'applicazione $\mathbf{H} : \mathbb{R}^d \to \mathbb{R}^d$ che associa gli $\mathbf{x}_u^N(v_i)$ ai relativi $\mathbf{x}_u^N(v_i + 1)$.
Sviluppando al primo ordine si ottiene una applicazione $\mathbf{G} : \mathbb{R}^d \to \mathbb{R}^d$:

$$\mathbf{x}_u^N(v_i + 1) = \mathbf{G}(\mathbf{x}_u^N(v_i)) = A \cdot \mathbf{x}_u^N(v_i) + B \tag{3.30}$$

dove:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1d} \\ a_{21} & a_{22} & \cdots & a_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ a_{d1} & a_{d2} & \cdots & a_{dd} \end{pmatrix} \quad B = (b_1, b_2, \cdots, b_d) \tag{3.31}$$

Si osserva che G, essendo lineare, consente di ridurre il problema alla ricerca di un funzionale, ovvero di un'applicazione $L : \mathbb{R}^d \to \mathbb{R}$ del tipo:

$$y(v_i + 1) = L(\mathbf{x}_u^N(v_i)) = \mathbf{a} \cdot \mathbf{x}_u^N(v_i) + b \tag{3.32}$$

con $\mathbf{a} = (a_1, a_2, \cdots, a_d)$ e b scalare.
La ricerca del vettore $\mathbf{a}$ e di b si effettua minimizzando la quantità:

$$Q = \sum_{m=1}^{k} \left| y(v_i + 1) - \mathbf{a} \cdot \mathbf{x}_u^N(v_i) - b \right|^2 \tag{3.33}$$

Per minimizzare Q si possono utilizzare gli algoritmi di discesa gradiente nello spazio dei $d + 1$ coefficienti. Una volta ottenute le stime $\hat{\mathbf{a}}$ e $\hat{b}$ per $\mathbf{a}$ e b minimizzando la quantita nell'eq. 3.33, la predizione relativa ad $\mathbf{x}_u$ sarà:

$$\hat{y}_u = \hat{\mathbf{a}} \cdot \mathbf{x}_u + \hat{b} \tag{3.34}$$

Bisogna sottolineare che i metodi locali richiedono che l'attrattore sia ben popolato, in modo che i k vicini distino poco fra loro. Tenuto conto di ciò, la scelta di k dovrebbe essere commisurata alle dimensioni del database a disposizione.

Capitolo 4

Metodi di ensemble

Questo capitolo tratta i metodi di ensemble. Dopo una introduzione generale su cosa si intenda per ensemble, vengono descritti i due metodi che sono stati implementati, Bagging e Adaboost.
In particolare Freund e Schapire hanno vinto nel 2003 il premio Gödel[1] con l'articolo [17] su Adaboost.

[1] I particolari sono disponibili al sito sigact.acm.org/prizes/godel/

4.1 Introduzione ai metodi di ensemble

Con *ensemble* si intende un insieme di macchine d'apprendimento base le cui predizioni vengono combinate per migliorare le prestazioni globali. La varietà di termini con cui in letteratura vengono chiamate le varie macchine, riflette l'assenza di una teoria unificata sui metodi di ensemble e il fatto che sia un campo di ricerca ancora da esplorare sotto molti aspetti [29]. Le ragioni per cui indagare sull'efficacia di questi metodi sta nei risultati sperimentali fino ad oggi ottenuti [11] [16] [6] [27] [12] [13], supportati da argomentazioni teoriche [10] [2] [6] [13] [7].

I metodi di ensemble si possono dividere in due categorie: generativi e non generativi. Quelli non generativi cercano di combinare nel miglior modo possibile le predizioni fatte dalle macchine, mentre quelli generativi generano nuovi set di learner, in modo da generare fra essi delle diversità che possano migliorare le prestazioni globali.

Per quanto riguarda le tecniche non generative, in classificazione, per esempio, si utilizza la tecnica della votazione a maggioranza [29] (*major voting*), eventualmente raffinata pesando i voti proposti dalle macchine. Si possono combinare le predizioni attraverso operazioni di media (eventualmente pesata), mediana, prodotto, somma, oppure scegliendo il minimo o il massimo [29].

I metodi generativi tentano di migliorare le performance del sistema tentando di utilizzare le diversità fra i learner. Per fare questo vengono generati diversi set con cui addestrare le macchine (tecnica di *resampling*), o vengono manipolate diversamente l'aggregazione delle classi (tecnica di *feature selection*) o ancora si possono addestrare dei learner che si specializzano su parti specifiche del set di apprendimento (*mixture of experts*). Le tecniche di resampling, come il *bootstrapping* [29], consentono di generare nuovi set a partire da quello originale e vengono utilizzate nei due metodi generativi che

sono stati utilizzati nella tesi: Bagging e Adaboost.

4.2 Bagging

Il bagging è un metodo di ensemble che prende il nome dall'unione delle parole Bootstrap AGGregatING [6]. La tecnica di bootstrapping su un database consiste nell'estrazione con rimpiazzo dei suoi elementi in modo da creare nuovi diversi training set. La probabilità di estrazione di ogni esempio, nel bagging, è uguale a quella degli altri. L'algoritmo base prevede la creazione di modelli per ogni training set e successivamente la combinazione delle varie predizioni sul test set attraverso un'operazione di media.

Avendo a disposizione il training set iniziale $\mathcal{L} = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_l, y_l)\}$, si costruiscono p nuovi training set $\mathcal{L}_k$ con $k = 1, \cdots, p$ estraendo da $\mathcal{L}$ con rimpiazzo. A partire da ognuno di questi nuovi training set si effettua un modello $f_k(\mathbf{x}, \mathcal{L}_k)$ e successivamente si considera come modello predittivo quello ottenuto mediando ls f_k:

$$f(\mathbf{x}, \mathcal{L}) = \frac{1}{p} \sum_{i=1}^{p} f_k(\mathbf{x}, \mathcal{L}_k) \tag{4.1}$$

La ragione per cui l'algoritmo può portare a dei miglioramenti sta nella diversità dei vari modelli f_k. Per questo motivo vengono consigliati modelli base meno stabili, ovvero in grado di produrre diversità consistenti anche a partire da set di addestramento simili. Questo non significa che le macchine base debbano essere necessariamente diverse.

Ricordando la scomposizione dell'errore in bias e variance (eq. 1.14), la ragione per cui il Bagging può migliorare le prestazioni globali in termini di predizione è che esso riduce il variance [14]. D'altra parte è auspicabile che le macchine che ne fanno parte abbiano basso bias.

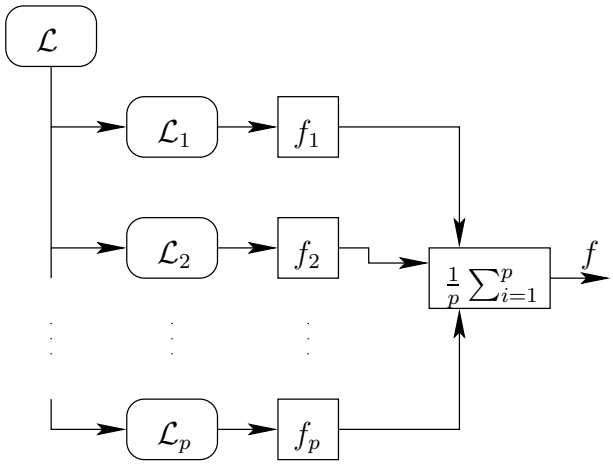


Figura 4.1: Schema a blocchi dell'algoritmo Bagging.

Il criterio con cui è stato scelto il tipo di architettura da utilizzare nel bagging, si basa sulla valutazione del migliore in termini di prestazioni sul test set. Per giustificare questa scelta si può effettuare una valutazione dell'errore delle macchine base in termini di bias e variance (eq. 1.14). Mediare modelli a basso bias permette di ottenere ancora un modello a basso bias riducendo il variance. Nonostante non si sia affrontata l'analisi bias-variance delle macchine, la scelta di considerare il modello con migliori prestazioni sul test set si è rivelata ragionevole (ved. parte 2).

Si può criticare ulteriormente la scelta pensando che le macchine migliori sono ottimali per il training set originale e non si sa se possano esserlo per i vari set generati dall'estrazione casuale. La speranza è che le migliori possiedano caratteristiche di complessità adatte alle caratteristiche dei dati da approssimare.

Riguardo alle dimensioni degli $\mathcal{L}_k$, non esiste una teoria che indichi quale debba essere il valore ottimale [5]. Queste dipendono dalla quantità e dalla tipologia dei dati a disposizione.

Infine il bagging è un algoritmo parallelizzabile poiché l'addestramento di

una singola macchina non influenza in alcun modo quello delle altre.

4.3 Adaboost

Il nome Adaboost [12] [4] deriva dal fatto che l'ensemble prevede il bootstrap adattivo nel senso che possiede la capacità di adattarsi alle caratteristiche di difficoltà del training set. L'idea centrale è quella di estrarre a caso un certo numero di esempi dal training set assegnando successivamente una probabilità maggiore di estrazione per gli esempi più difficili da apprendere. Inizialmente si addestra un primo predittore con un training set costruito mediante estrazione casuale con probabilità uguale per tutti gli esempi. Fatto ciò si aggiornano le probabilità di estrazione per il training set successivo aumentando quella degli esempi del set originale appresi peggio. Si genera un nuovo training set e si addestra un nuovo predittore e così via.

Sperimentalmente si è evidenziato che Adaboost può ridurre sia bias che variance (eq. 1.14) [14].

Nell'introdurre Adaboost, Freund e Schapire introducono la nozione di *weak learner* [17] nei problemi di classificazione, come macchina d'apprendimento le cui prestazioni sono migliori del predittore casuale. In questo lavoro si è seguito l'algoritmo presentato da Drucker [12] per la sua chiarezza. In questi il base learner utilizzato è l'*albero di regressione* (*regression tree*), in quanto week learner.

Il punto di vista seguito nella tesi è invece qullo di effettuare un'analisi delle caratteristiche degli ensemble utilizzando i base learner introdotti nel capitolo precedente.

Formalmente l'algoritmo [12] inizia assegnando una probabilità $p_i^{(1)} = 1/l$ di estrazione ad ognuno degli l esempi appartenenti al set:

$$\mathcal{L} = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_l, y_l)\} \tag{4.2}$$

Si genera un training set $\mathcal{L}_1$ estraendo con rimpiazzo dal set iniziale $l^{(1)}$ esempi e si addestra la prima macchina ottenendo un modello $f^{(1)}$ con il quale si può effettuare una previsione $\hat{y}_i^{(1)}$ per ogni $\mathbf{x}_i \in \mathcal{L}$. Si calcola dunque:

$$L_i^{(1)} = L\left(|\hat{y}_i^{(1)} - y_i|\right) \tag{4.3}$$

dove L è una delle funzioni di perdita:

$$L_i^{(1)} = \frac{|\hat{y}_i^{(1)} - y_i|}{D} \tag{4.4}$$

$$L_i^{(1)} = \left(\frac{|\hat{y}_i^{(1)} - y_i|}{D}\right)^2 \tag{4.5}$$

$$L_i^{(1)} = 1 - e^{-\frac{|\hat{y}_i^{(1)} - y_i|}{D}} \tag{4.6}$$

e D è un opportuna costante di normalizzazione per fare in modo che $L_i \in [0, 1]$:

$$D = \max_{i=1,\cdots,N} |\hat{y}_i^{(1)} - y_i| \tag{4.7}$$

Si calcola la media pesata degli $L_i^{(1)}$:

$$\bar{L}^{(1)} = \sum_{i=1}^{l^{(1)}} L_i^{(1)} p_i^{(1)} \tag{4.8}$$

e $\beta^{(1)}$, una quantità che dà un valore tanto più piccolo quanto maggiore è la bontà dell'apprendimento rispetto ad $\mathcal{L}$:

$$\beta^{(1)} = \frac{\bar{L}^{(1)}}{1 - \bar{L}^{(1)}} \tag{4.9}$$

A questo punto si aggiornano le probabilità di estrazione secondo l'espressione:

$$p_i^{(1)} = p_i^{(1)} \left(\beta^{(1)}\right)^{1 - L_i^{(1)}} \tag{4.10}$$

e si normalizzano a 1. Questo procedimento consente appunto di assegnare una probabilità di estrazione maggiore agli esempi di $\mathcal{L}$ predetti peggio. Si itera il procedimento finché non si raggiunge un valore di T per cui $\bar{L}^{(T)}$

risulta maggiore di 0.5. Arrivati infatti ad un tale valore si è raggiunta una condizione per cui l'ultima macchina non ha prodotto una buona previsione sul training set originale $\mathcal{L}$.
Per calcolare la previsione su un certo $\mathbf{x}_i$ da parte dell'ensemble così composto si effettua un'operazione di mediana degli $\hat{y}_i^{(t)}$ pesata dai relativi $\beta^{(t)}$. Per effettuare questa operazione si considerano le predizioni $\hat{y}_i^{(t)}$ e i relativi $\beta^{(t)}$ di tutte le T macchine che hanno partecipato all'algoritmo. Esse vengono rinominate in modo che sia $\hat{y}_i^{(1)} < \hat{y}_i^{(2)} < \cdots < \hat{y}_i^{(T)}$ mantenendo sempre l'associazione fra un $\hat{y}_i^{(t)}$ e $\beta^{(t)}$. Quindi si somma il $\log \frac{1}{\beta^{(t)}}$ su t finché non viene soddisfatta la disuguaglianza:

$$\sum_t \log \frac{1}{\beta^{(t)}} \geq \frac{1}{2} \sum_t \log \frac{1}{\beta^{(t)}} \tag{4.11}$$

Chiamando con t^* il valore minimo di t per cui vale la 4.11, si considera come predizione $\hat{y}_i$ quella della macchina t^*, cioè $\hat{y}_i = \hat{y}_i^{(t^*)}$. L'utilizzo della mediana rispetto per esempio alla media risiede nella sua maggiore robustezza.

Anche nell'algoritmo di Adaboost, come per il Bagging, si è scelto di costruire l'ensemble con macchine tutte uguali, con l'architettura in grado di garantire il minimo dell'errore sul test set.

Parte II

Implementazione ed applicazioni

Capitolo 5

Ambiente software

A partire da questo capitolo elenchiamo i principali risultati sperimentali ottenuti. La prima parte è dedicata alla descrizione dell'ambiente utilizzato per l'implementazione degli algoritmi illustrati nei precedenti capitoli.

5.1 Il linguaggio R

Gli esperimenti sono stati condotti in un ambiente software chiamato R[1], uno strumento per l'analisi statistica e l'apprendimento automatico molto potente. Nacque come estensione ad S e ed è un progetto GNU[2] orientato alla diffusione del software libero. R è un linguaggio interpretato che mette a disposizione una vasta serie di toolbox ottimizzati per la risoluzione di una vastissima gamma di problemi.
In particolare sono presenti elementi quali vettori, matrici e oggetti con strutture più articolate come i data frames e le liste. È stata sviluppata una serie di operatori per questi oggetti che rendono veloce e compatto il calcolo di praticamente tutto ciò di cui si necessita. Tutto questo, assieme alla sua somiglianza con Matlab, ha favorito la sua diffusione nella comunità scentifica.

Per quanto riguarda i toolbox a disposizione si citano alcuni di quelli che sono stati utilizzati per questo lavoro, come il pacchetto di implementazione del percettrone multistrato (*nnet*), delle macchine a supporto vettoriale (*svm*) e di funzioni di ricerca di minimi di funzionali di più variabili (*optim* oppure *optimize*).
Ancora, la presenza di tutte le strutture di controllo possedute da un linguaggio di programmazione (espressioni condizionali e loops) e la possibilità di costruire grafici 2D e 3D in maniera molto naturale fanno di R uno strumento di lavoro veramente efficace.

Il fatto che il linguaggio sia interpretato ha il vantaggio di risultare compatto e di rendere facile l'implementazione di un algoritmo poiché si scrive il codice ad alto livello. Questo ha però lo svantaggio di poter essere lento

[1] R è disponibile al sito www.r-project.org/ sia per sistemi operativi Windows che Linux.

[2] Esso nacque nel 1984 per lo sviluppo di un sistema operativo tipo UNIX completamente libero. Per una descrizione completa del progetto GNU si rimanda a www.gnu.org, in cui è possibile capire cosa si intende per software libero.

(cosa che accade quando si fanno molte operazioni elementari). Come già detto i tools sviluppati sono ottimizzati; quello che rallenta l'esecuzione è il fatto che ogni istruzione deve essere interpretata. Per esempio, l'esecuzione di 10^8 cicli che non compiono nessuna operazione, in C impiega un tempo che è dell'ordine di cinquanta volte minore della corrispondente in R.
Per questo motivo R mette a disposizione la possibilità di chiamare all'interno di un suo programma routines esterne scritte in C, C++ e Fortran precedentemente compilate. Ciò è utile qualora si debbano sviluppare delle parti di codice che richiedono una complessità difficilmente gestibile in R dal punto di vista dell'ottimizzazione temporale.

Per esempio, la ricerca nello spazio Y_d del vettore più vicino ad $\mathbf{y}(k)$, è un problema che richiede la costruzione di una struttura dati da esplorare attraverso una serie di cicli e di operazioni di distanza. In R un'algoritmo del genere per database grossi non è pensabile. In C, invece, l'algoritmo risulta efficiente e veloce anche per database molto grandi.

Si è lavorato sotto un sistema operativo Linux con la distribuzione RedHat 9 con kernel *OpenMosix* che ha permesso di sfruttare le potenzialità offerte da un cluster di sette work station Pentium 1GHz. La particolarità di OpenMosix è la possibilità di smistare i processi sulle varie macchine, bilanciando in maniera automatica il carico di lavoro fra esse. Questo è risultato molto utile quando si è voluto provare alcuni algoritmi con diversi parametri. Ognuno infatti veniva eseguito da una diversa macchina senza rallentamenti dovuti alla presenza di altri processi e in maniera assolutamente trasparente.

OpenMosix non consente di parallelizzare un singolo processo ma è compatibile con librerie come MPI[3] o PVM[4] (per esempio) e scrivendo del codice parallelo dedicato al problema da risolvere.

[3]Message Passing Interface per cui si può vedere per esempio: www.mpi-forum.org.
[4]Parallel Virtual Machine. Informazioni più dettagliate al sito www.netlib.org/pvm3.

5.2 Percettroni multistrato - NNET

In R è già implementato, attraverso la funzione *nnet*, il percettrone a due strati ed è contenuto nel pacchetto *nnet*. La funzione nnet consente di scegliere percettrone ad uno strato o Adaline semplicemente tramite un controllo booleano, e risolve il problema dell'apprendimento minimizzando la funzione costo regolarizzata (eq. 3.11). La minimizzazione avviene attraverso una discesa gradiente nello spazio dei pesi, che si interrompe quando l'algoritmo di minimizzazione non riesce a ridurre il costo al di sopra di una soglia selezionabile dall'utente. A questo punto si considera che la minimizzazione sia giunta alla convergenza.
La funzione nnet effettua l'addestramento sulla base di un data base di esempi fornito dall'utente e consente di scegliere alcuni parametri, di cui citiamo quelli utilizzati per la regressione:

- scelta del tipo di architettura fra percettrone a due strati e Adaline;
- numero di neuroni nello strato nascosto *nnhl*;
- termine di decadimento dei pesi λ nel costo;
- soglia per l'interruzione dell'algoritmo di minimizzazione del costo;
- numero massimo di iterazioni dopo le quali interrompere l'algoritmo di apprendimento;
- valori iniziali con cui inizializzare la matrice delle connessioni;
- funzione di attivazione per i neuroni di uscita.

La riduzione della complessità della rete si può ottenere impostando un valore per il numero massimo di iterazioni da parte dell'algoritmo di minimizzazione oppure impostando un valore di $\lambda \neq 0$.
Nella tesi si è scelta la seconda soluzione.

5.3 Macchine a supporto vettoriale - SVM

Le SVM in R sono implementate attraverso la funzione *svm* nel pacchetto *e1071*. La funzione svm segue l'algoritmo descritto in dettaglio in: http://www.csie.ntu.edu.tw/cjlin/papers/libsvm.ps.gz e dà all'utente la possibilità di variare diversi parametri:

- tipo di kernel (lineare, polinomiale, gaussiano);
- parametri del kernel (γ per kernel gaussiano, p e b_0 per kernel polinomiale eq. 3.28);
- valore del parametro di regolarizzazione C nell'eq. 3.16;
- valore di ε nella loss function di Vapnik (eq. 3.14).

5.4 Software implementato

Tutti i metodi illustrati nel capitolo 2 sono stati implementati in R durante lo svolgimento di questo lavoro. Riguardo alle parti di codice più complesse si è scelto di scrivere queste parti in C e successivamente importarlo nell'ambiente R per l'analisi e l'elaborazione dei risultati. In particolare sono stati scritti in linguaggio C:

- l'algoritmo di calcolo della mutua informazione;
- l'algoritmo di calcolo dei falsi vicini ottimizzato [26];
- l'algoritmo di ricerca dei k vicini nei metodi locali.

Il resto è stato scritto in R, e citiamo gli algoritmi principali:

- analisi di spettro singolare (SSA);
- algoritmo di addestramento per i MLP e le SVM con diversi parametri per scegliere quelli che forniscono prestazioni migliori;

- Bagging;
- Adaboost;
- metodi locali.

Bagging e Adaboost utilizzano al loro interno le funzioni che implementano i base learners.
In particolare è stato sviluppato un programma in grado di generare, a partire da una serie storica, il set di apprendimento e quello di test. In più è possibile scegliere se effettuare la previsione ad uno o più istanti successivi e si può aggiungere l'informazione proveniente da altre serie storiche relative allo stesso sistema (per esempio per prevedere la temperatura si può utilizzare la serie di temperatura e quella di pressione). La gestione di questi casi avviene in maniera molto semplice.
I dettagli tecnici di implementazione vengono dati in appendice A.

Capitolo 6

Applicazione ad un problema di regressione

In questo capitolo si sono voluti mettere alla prova i metodi di ensemble in confronto con i base learners in un problema di regressione consistente nell'approssimazione di una funzione sinusoidale contaminata da una grande quantità di rumore. In particolare si tratta di prevedere il valore della funzione in alcuni punti scelti a caso all'interno del suo dominio.

6.1 Generazione della serie

Il problema di regressione proposto è quello di approssimare una funzione sinusoidale alla quale viene sommato del rumore gaussiano bianco[1] a media nulla. La serie viene generata in questo modo:

$$s_k = \sin(0.02k) + \varepsilon_k, \qquad k = 1, \cdots, 1000 \tag{6.1}$$

I parametri di media e deviazione standard con cui viene generato il rumore sono stati scelti in:

$$\bar{\varepsilon} = 0 \tag{6.2}$$

$$\sigma_\varepsilon = 1 \tag{6.3}$$

La serie così ottenuta risulta molto contaminata dal rumore.

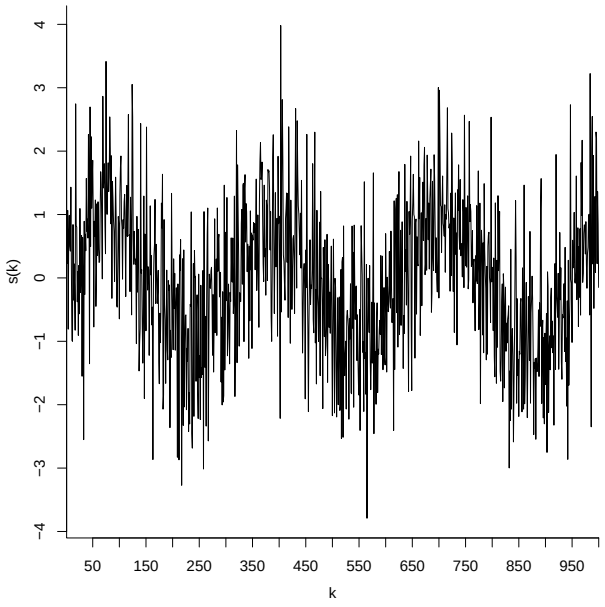


Figura 6.1: Rapresentazione grafica della serie generata attraverso l'eq. 6.1

[1]Si intende rumore bianco (quindi con caratteristiche di memoria nulla) con distribuzione delle ampiezze gaussiana.

Si costruisce il database totale delle associazioni $\{(x_k, y_k)\}_{k=1,\cdots,1000}$ attraverso le coppie $\{(k, s_k)\}_{k=1,\cdots,1000}$. Da esso ne vengono estratti a caso senza rimpiazzo l'80% per formare il training set mentre il restante 20% va a costituire il test set.

6.2 Criteri di confronto e scelta dei parametri

Segue l'elenco dei metodi messi a confronto:

- rete neurale;
- SVM;
- bagging con reti neurali;
- bagging con SVM;
- adaboost con reti neurali;
- adaboost SVM.

Il criterio con il quale si giudica la bontà di un modello di regressione è la valutazione sul test set del valore $RMSE$ (eq. 3.1) della previsione sia sul test set che sul test set senza rumore (che chiameremo $RMSE_2$). Quest'ultima è utile poiché permette di valutare la capacità di avere approssimato la funzione sinusoidale. Per quanto riguarda le macchine base, cioé la rete neurale e la SVM, si procede alla scelta di quelle ottimali addestrandone sul training set diverse, variando di volta in volta i parametri e calcolando gli $RMSE$.
Per la rete neurale i parametri liberi sono:

- il numero di neuroni dello strato nascosto ($nnhl$);
- il termine di decadimento (λ nell'eq. 3.11) per il compromesso fra approssimazione e complessità della macchina.

Per le SVM:

- la funzione kernel con i suoi parametri;
- il valore di ε nella funzione di perdita di Vapnick eq. 3.14;
- il valore di C nell'eq. 3.16 che rappresenta il compromesso fra approssimazione e complessità della macchina.

Si è scelto di utilizzare un kernel di tipo gaussiano, quindi il parametro da variare risulta solo quello relativo alla ampiezza della gaussiana γ (eq. 3.28). Infatti nel caso polinomiale si avrebbero due parametri da variare (grado del polinomio e termine noto), appesantendo ulteriormente la ricerca dei parametri ottimali.

6.3 Addestramento delle macchine base

Il range in cui si sono variati i paramentri dei MLP sono stati: $nnhl = 2, \cdots, 40$ e $\lambda = 0, \cdots, 1$.
Siccome l'inizializzazione dei pesi delle reti è casuale, ogni set di parametri è stato provato dieci volte per poi mediare l'errore commesso sulle dieci prove. Questo procedimento è inutile per le SVM, poiché il risultato dell'apprendimento a partire da parametri uguali è lo stesso. Il minimo $RMSE = 1.004$ sul test set è stato ottenuto con un numero di neuroni nello strato nascosto $nnhl = 36$ e con un termine di decadimento dei pesi $\lambda = 0.2$. Per quanto riguarda questa configurazione l'errore di approssimazione sulla funzione seno senza rumore è stato di $RMSE_2 = 0.196$

Il range in cui si sono variati i paramentri delle SVM sono stati: $\gamma = 1, \cdots, 20$, $C = 1, \cdots, 20$ e $\varepsilon = 0, \cdots, 0.5$; la superficie di errore per $\varepsilon = 0.4$ è mostrata in figura 6.3. Quelli per i quali si è ottenuto il minimo valore $RMSE = 0.976$ sul test set sono $\gamma = 2$, $C = 12$ e $\varepsilon = 0.4$. Con

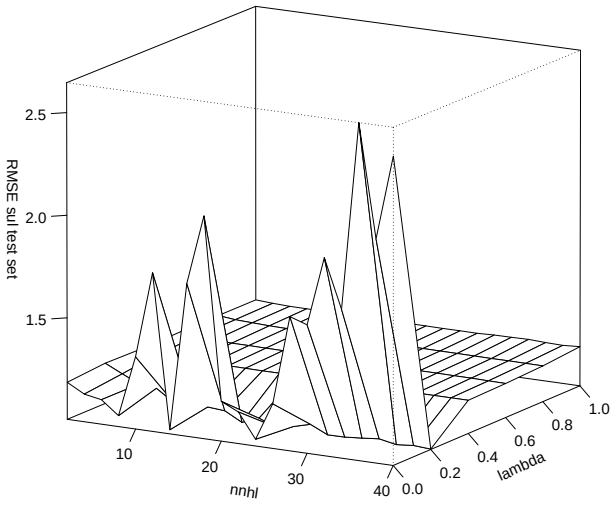


Figura 6.2: Superficie di errore in funzione di *nnhl* e λ per il MLP.

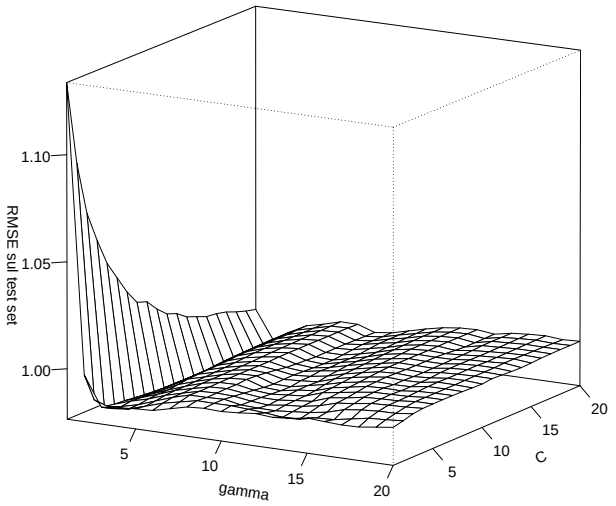


Figura 6.3: Superficie di errore in funzione dei parametri per le SVM con $\varepsilon = 0.4$.

questi parametri l'errore di approssimazione sulla funzione senza rumore è risultato $RMSE_2 = 0.168$. Il grafico 6.4 mostra la capacità della macchina di aver appreso la funzione senza rumore. Nel caso di $RMSE = 0$ i punti si disporrebbero lungo la bisettrice (linea grigia).

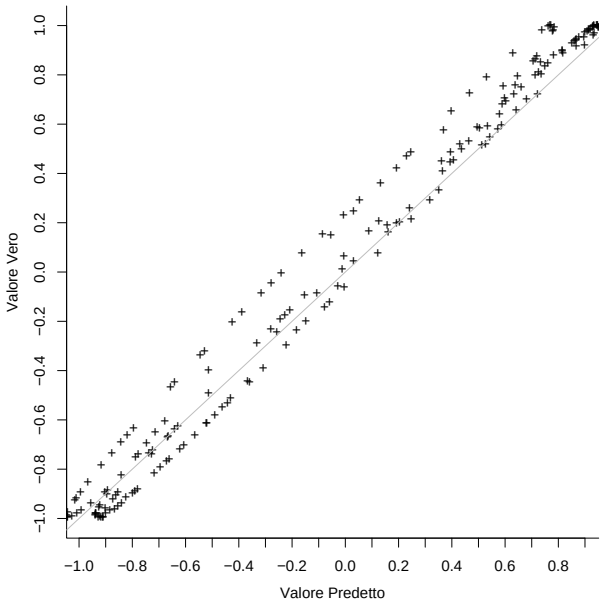


Figura 6.4: Valore vero contro valore predetto.

6.4 Addestramento degli ensemble

Costruendo gli ensemble utilizzando 50 macchine base[2] dello stesso tipo con i parametri scelti sulla base del migliore $RMSE$ sul training set originale, si ottengono i risultati in tabella 6.1.

Gli algoritmi di Adaboost utilizzavano la funzione di perdita tipo modulo nell'eq. 4.3. In figura 6.5 si può osservare come la distribuzione dei punti nel

[2]Per Adaboost questo significa porre un ulteriore criterio di fine per l'algoritmo, oltre a quello dettato dalla condizione $\bar{L}^{(k)} > 0.5$.

	MLP	SVM	Bagging MLP	Bagging SVM	Adaboost MLP	Adaboost SVM
$RMSE$	1.004	0.976	1.036	0.981	1.041	1.016
$RMSE_2$	0.196	0.168	0.281	**0.087**	0.244	0.204

Tabella 6.1: Tabella riassuntiva dei risultati per il problema di regressione.

grafico del valore predetto contro valore vero della funzione senza rumore sia più vicina alla bisettrice rispetto a quella di figura 6.4.

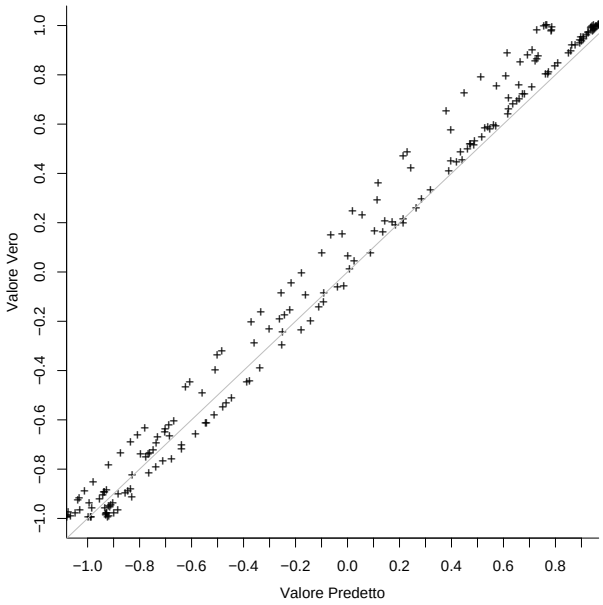


Figura 6.5: Retta di regressione.

Il bagging di reti neurali ha dato cattivi risultati, addirittura peggiorando l'errore rispetto alla macchina base. Questo si può ricondurre al fatto che le caratteristiche in termini di bias della macchina non fossero adatte all'interno dell'ensemble.

Riguardo ad Adaboost, le cose non vanno bene poiché dare più probabilità di estrazione agli esempi difficili non è di alcun aiuto nel migliorare le

prestazioni dell'ensemble. Infatti, in un caso simile, la complessità della serie deriva dall'aggiunta di rumore. Quindi l'algoritmo in qualche modo tenderà ad apprendere di più gli esempi più complessi per via del rumore. Questo comporta una sorta di overfitting dei dati difficili che non è di aiuto in un problema di questo tipo.

Capitolo 7

Serie di Lorenz

A partire da questo capitolo vengono affrontati problemi di previsione di serie storiche. La serie proposta in questo capitolo è quella generata dal sistema di Lorenz ed è stata utilizzata diffusamente come banco di prova per il test di metodi predittivi [1] [32]. Nella tesi si è effettuato un confronto fra le tecniche predittive mediante base learners e ensemble di essi sulla serie di Lorenz resa più complicata attraverso sottocampionamento.

I parametri che si valutano per una previsione sono il $RMSE$ (eq. 3.1) e il ME (eq. 3.4). D'ora in poi si aggiungerà un nuovo criterio per valutare se davvero i learners riescano ad apprendere e generalizzare il problema. Metteremo a confronto sempre i $RMSE$ con quelli di due predittori elementari, che chiameremo $Naive_0$ e $Naive_1$, per i quali le previsioni risultano:

$$\hat{s}_i = s_{i-1} \qquad Naive_0 \tag{7.1}$$

$$\hat{s}_i = s_{i-1} + (s_{i-1} - s_{i-2}) \qquad Naive_1 \tag{7.2}$$

Essi possono essere considerati predittori di grado zero e uno.

Inoltre, per la valutazione degli errori per le reti neurali, si sono ottenuti i loro valori mediando su dieci prove per ogni set di parametri.

7.1 Serie storica

Il sistema di Lorenz è descritto analiticamente dal set di equazioni differenziali:

$$\begin{cases} \dot{x} = \sigma(y - x) \\ \dot{y} = -xz + rx - y \\ \dot{z} = xy - bz \end{cases} \tag{7.3}$$

Il sistema si comporta in maniera caotica per esempio per $\sigma = 16$, $b = 4$ e $r = 45.92$. Le varie funzioni possono essere generate sinteticamente sostituendo all'operazione di derivata quella di rapporto incrementale[1] con intervallo di tempo piccolo[2], intorno a $10^{-3}s$. La figura 7.1 si riferisce all'andamento della

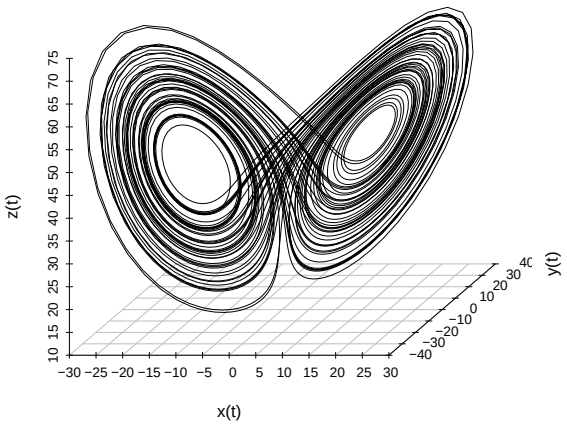


Figura 7.1: L'attrattore di Lorenz.

soluizione a partire da $x(0) = y(0) = z(0) = 0.001$ nello spazio costituito dalle terne $(x(t), y(t), z(t))$.

[1] Anziché la derivata $\frac{dx}{dt}$, il rapporto incrementale $\frac{\Delta x}{\Delta t}$, con Δt finito.

[2] Un intervallo grande porterebbe le soluzioni del sistema a comportamenti assolutamente incontrollabili

Il problema della predizione di una delle serie ottenute campionando per esempio con $\tau_s = 10^{-2}s$ una delle tre variabili è piuttosto semplice ed è stato studiato diffusamente [1].
Per rendere il problema più difficile si è pensato di sottocampionare il segnale proveniente da una delle variabili ottenendo una serie complicata. In particolare si è campionato il segnale $x(t)$ con $\tau_s = 0.2s$, ottenendo una serie di 1000 campioni $\{s_i\}_{i=1,\cdots,1000}$, mostrata in figura 7.2.

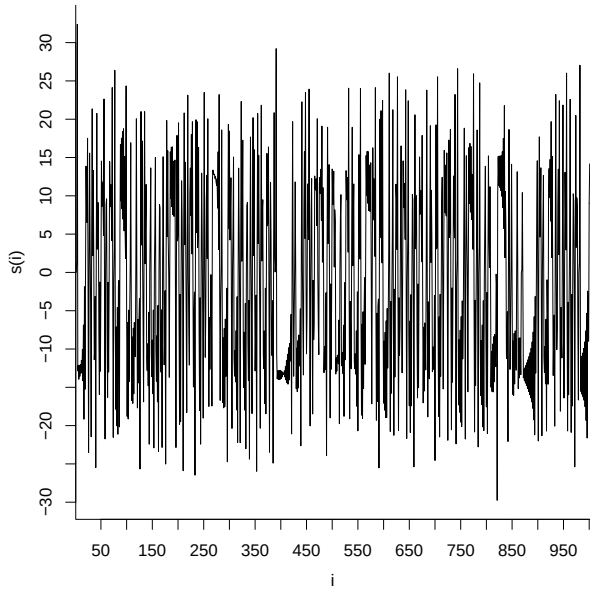


Figura 7.2: La serie di 1000 valori di $\{s_i\}$, ottenuta campionando $x(t)$ con $\tau_s = 0.2s$.

7.2 Ricostruzione della dinamica

Per poter ricostruire la dinamica a partire dalla serie $\{s_i\}$ applichiamo gli algoritmi di mutua informazione e falsi vicini (da pagina 18). L'analisi della mutua informazione evidenzia una perdita di memoria molto rapida, dovuta proprio al sottocampionamento. In questi casi viene consigliata una scelta di T pari ad uno [1]. Infatti il grafico della mutua informazione $I(T)$

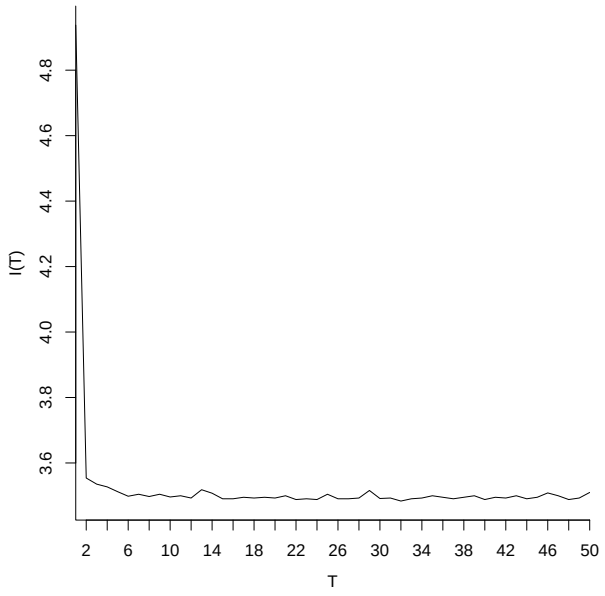


Figura 7.3: Mutua informazione in funzione di T.

(fig. 7.3), non presenta un vero e proprio minimo, bensì una zona quasi piatta per $T \geq 2$. Posto $T = 1$, l'andamento dei falsi vicini in funzione della dimensione di embedding presenta un minimo e una successiva risalita. Questo comportamento è tipico di serie casuali [1], e in questo caso deriva dal forte sottocampionamento di $x(t)$. Infatti ci si dovrebbe aspettare che la percentuale scenda per arrivare eventualmente a zero.
Il valore di d per la costruzione del vettore delle coordinate ritardate, viene scelto in base al minimo della percentuale di falsi vicini, quindi $d = 3$.

7.3 Risultati sulla serie storica di Lorenz

Per l'addestramento delle reti neurali si sono variati i parametri $nnhl = 2, \cdots, 40$ e $\lambda = 0, \cdots, 0.5$. Ciò ha portato alla scelta della migliore avente numero di neuroni nello strato nascosto $nnhl = 36$ e termine di decadimento

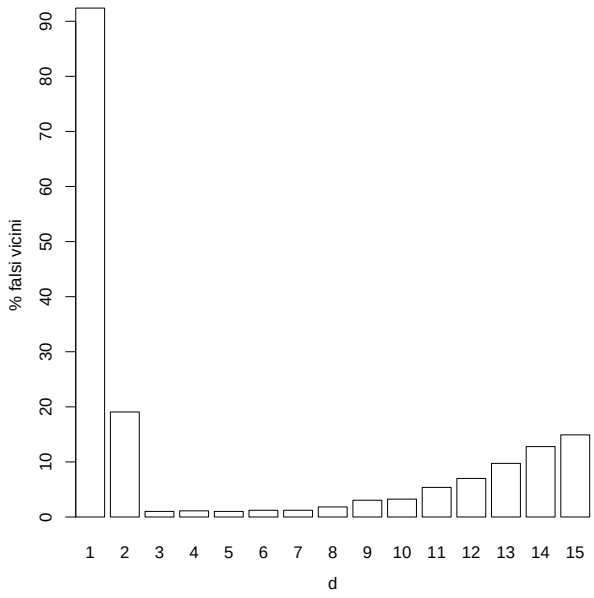


Figura 7.4: Percentuale di falsi vicini in funzione di d per $T = 1$.

dei pesi $\lambda = 0.1$. Essa ha commesso un errore sul test set pari a $RMSE = 2.13$.

Anche qui per le SVM si è scelto un kernel di tipo gaussiano in modo che vi fosse un solo parametro da variare, γ. Il miglior $RMSE = 2.76$ si è ottenuto con $\gamma = 5$, $C = 5$ e $\varepsilon = 0.005$. Nonostante si siano considerati range di parametri molto vasti ($\gamma = 10^{-5}, \cdots, 10^4$, $C = 10^{-2}, \cdots, 10^3$ e $\varepsilon = 10^{-5}, \cdots, 5$) per le SVM, quella migliore commette un errore maggiore rispetto alla migliore rete neurale.

Per i metodi locali il parametro libero è il numero di k vicini che è stato variato da 2 a 50. Si è valutato il minimo $RMSE$ attraverso i metodi KNN1 e KNN2 (vedere sezione 3.4).
Il minimo $RMSE = 3.38$ si ha per l'algoritmo KNN2 con $k = 6$.

Infine il confronto dei vari metodi in tabella 7.1, in cui si elenca anche il massimo del valore assoluto dell'errore ME (eq. 3.4). Per gli algoritmi di Adaboost e Bagging si sono utilizzati 100 predittori e per Adaboost la

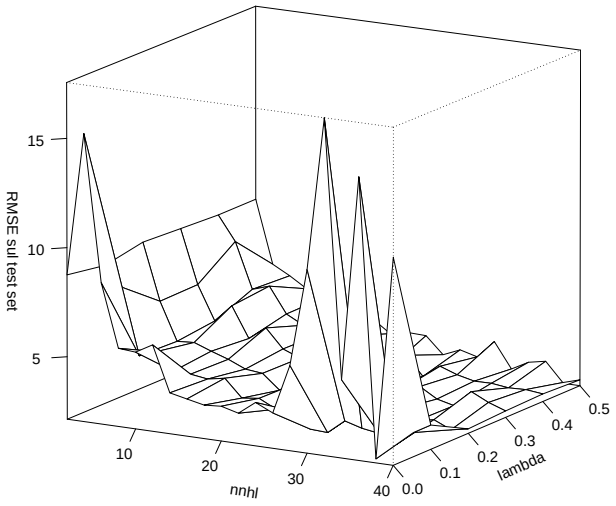
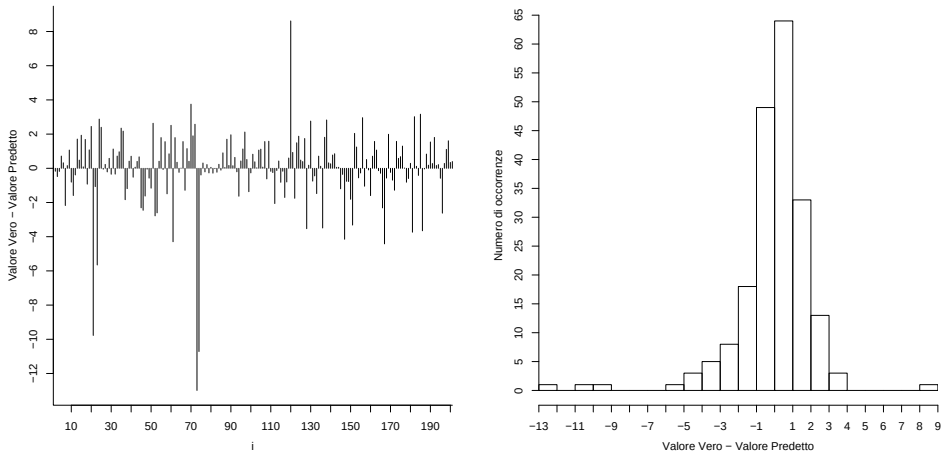


Figura 7.5: Errore sul test in funzione di *nnhl* e λ.



(a) (b)

Figura 7.6: (a) Andamento temporale dell'errore di predizione commesso dalla rete neurale sulla serie di Lorenz. (b) Istogramma delle occorrenze dell'errore commesso dalla rete neurale sulla serie di Lorenz.

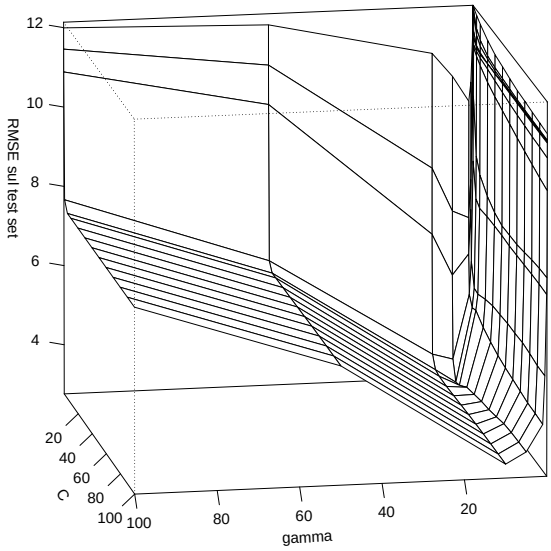
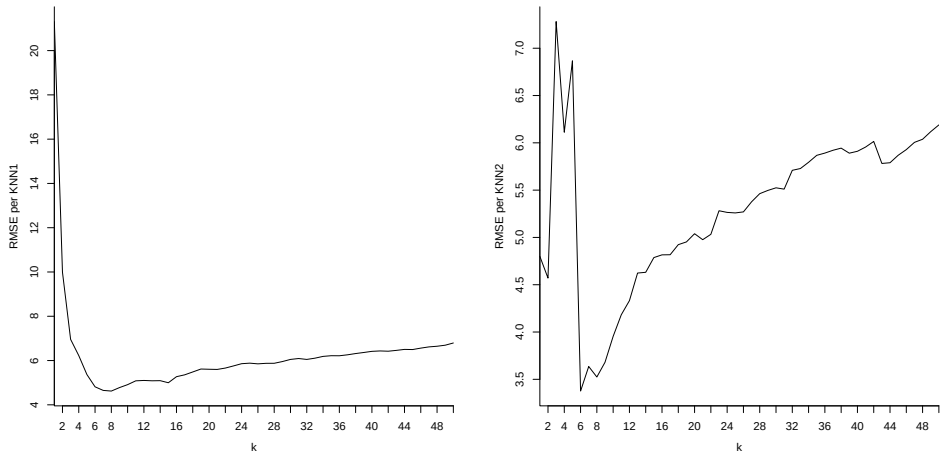


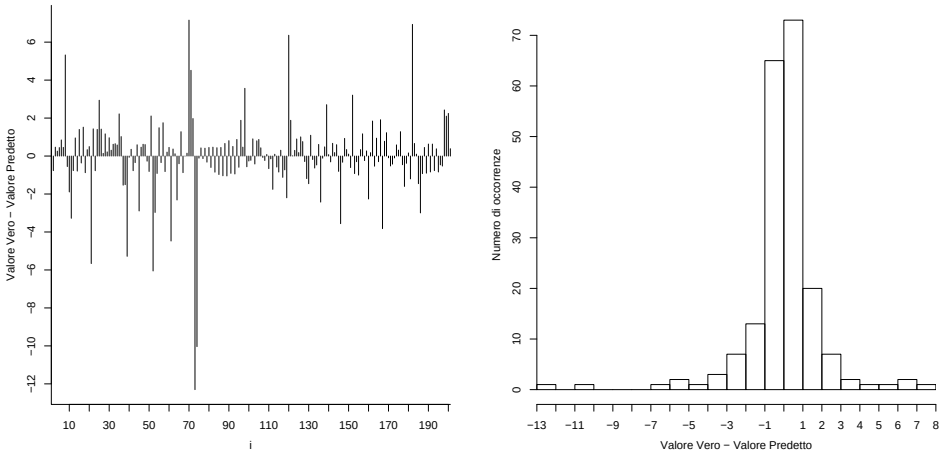
Figura 7.7: Errore sul test per $\varepsilon = 0.005$.



(a) (b)

Figura 7.8: (a) Errore sul test per KNN1 in funzione di k per la serie di lorenz. (b) Errore sul test per KNN2 in funzione k per la serie di lorenz.

funzione di perdita di tipo modulo nell'eq. 4.3.



(a) (b)

Figura 7.9: (a) Andamento temporale dell'errore di predizione commesso dal bagging con reti neurali. (b) Istogramma delle occorrenze dell'errore commesso dal bagging di reti neurali.

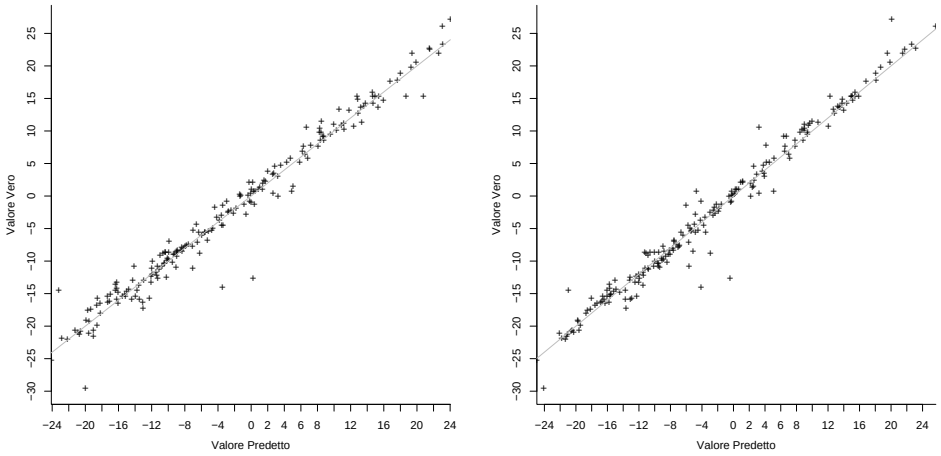
	MLP	SVM	Bagging MLP	Bagging SVM	Adaboost MLP	Adaboost SVM	KNN
$RMSE$	2.13	2.76	2.42	2.89	**2.02**	3.02	3.38
ME	13.0	21.3	12.6	23.7	**12.3**	22.4	24.8

Tabella 7.1: Risultati sulla serie di lorenz.

KNN in tabella 7.1 si riferisce all'algoritmo KNN2, migliore rispetto al KNN1. In questo caso l'algoritmo di Adaboost con reti neurali ha fornito i risultati migliori. Questo significa che le caratteristiche della rete erano adatte all'ensemble.

Il fatto che i Bagging peggiorino le predizioni fa intuire che il bias della rete neurale e della SVM fosse elevato. In figura 7.10 viene fatto un confronto fra

i grafici di valore vero contro valore predetto per la rete neurale migliore e per Adaboost di reti neurali.



(a) (b)

Figura 7.10: (a) Valore vero contro valore predetto per la rete neurale migliore sulla serie di Lorenz. (b) Valore vero contro valore predetto per adaboost con reti neurali sulla serie di Lorenz.

I risultati superano di gran lunga le prestazioni dei $Naive_0$ e $Naive_1$ per cui si ha rispettivamente $RMSE = 13.17$ e $RMSE = 21.11$.

Capitolo 8

Laser ad anello

La serie considerata in questa sezione è reperibile sul web[1] e venne utilizzata in una competizione di metodi predittivi [32]. Sempre in [32] è possibile ottenere una descrizione dettagliata dell'esperimento che ha condotto alla serie s e delle analogie con il sistema di Lorenz. In questo capitolo viene presentata la serie per poi procedere alla ricostruzione della dinamica, fondamentale per l'applicazione dei metodi predittivi.
Anche per questa serie storica verranno valutati i metodi predittivi tramite $RMSE$ (eq. 3.1) e ME (eq. 3.4).

[1] http://www-psych.stanford.edu/ andreas/Time-Series/SantaFe.html

8.1 Serie storica

L'esperimento segue quelli descritti in [21] e [33] e [22] e riguarda un laser ad anello portato in una regione di lavoro caotica. La serie è stata costruita campionando con $\tau_s = 80ns$ l'intensità di un laser ad anello di tipo FIR (far-infrared) ed è stata successivamente resa digitale convertendo ogni misura in una stringa binaria ad 8 bit. Il database totale è di 10000 valori circa, ma per questo lavoro si è deciso di utilizzarne solo 1200, di cui 1000 per l'addestramento, per rendere il problema più complicato.

8.2 Ricostruzione della dinamica

La serie ottenuta tramite l'esperimento descritto in [32] viene mostrata in figura 8.1.

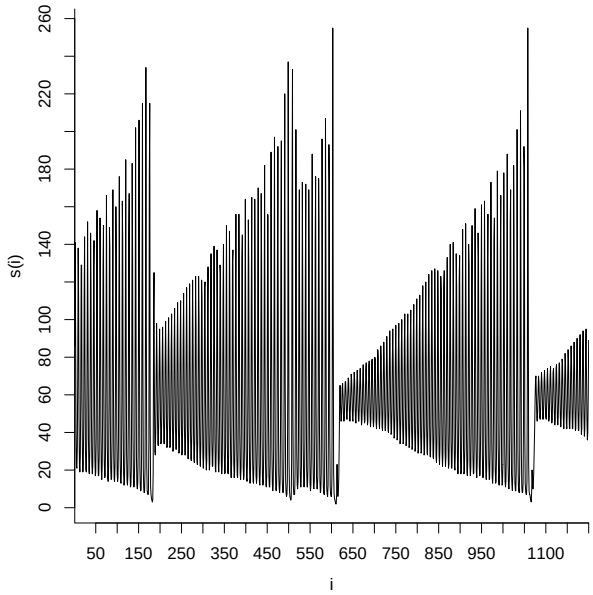


Figura 8.1: La serie di 1200 valori di $\{s_k\}$ del laser ad anello tipo FIR.

Per stimare il valore di T e successivamente di d da utilizzare nella costruzione del vettore delle coordinate ritardate, applichiamo gli algroritmi di mutua informazione e falsi vicini. Il primo minimo della mutua informa-

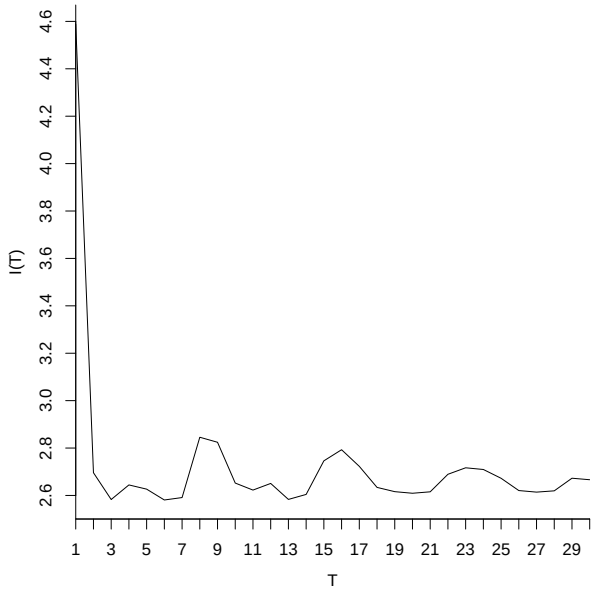


Figura 8.2: Mutua informazione in funzione di T.

zione si ha per $T = 3$, come anche il primo zero crossing dell'autocorrelazione. Con questa scelta la percentuale di falsi vicini indica un valore di $d = 4$.

8.3 Risultati sulla serie storica del laser

Per l'addestramento delle reti neurali si sono variati i parametri $nnhl = 2, \cdots, 50$ e $\lambda = 0, \cdots, 2$. La migliore rete neurale con $nnhl = 30$ e termine di decadimento dei pesi $\lambda = 1.4$ ha commesso un errore sul test set pari a $RMSE = 5.60$. In figura 8.5 viene mostrata la superficie di errore in funzione dei parametri.

Per le SVM i parametri sono stati variati nei range: $\gamma = 10^{-2}, \cdots, 10$, $C = 5, \cdots, 10^3$ e $\varepsilon = 10^{-4}, \cdots, 1$. La SVM con $\gamma = 1$, $C = 1000$ e $\varepsilon = 0.01$

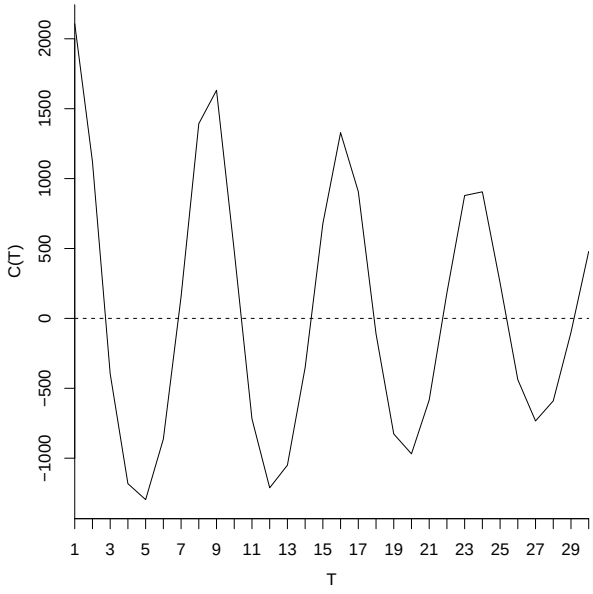


Figura 8.3: Autocorrelazione in funzione di T.

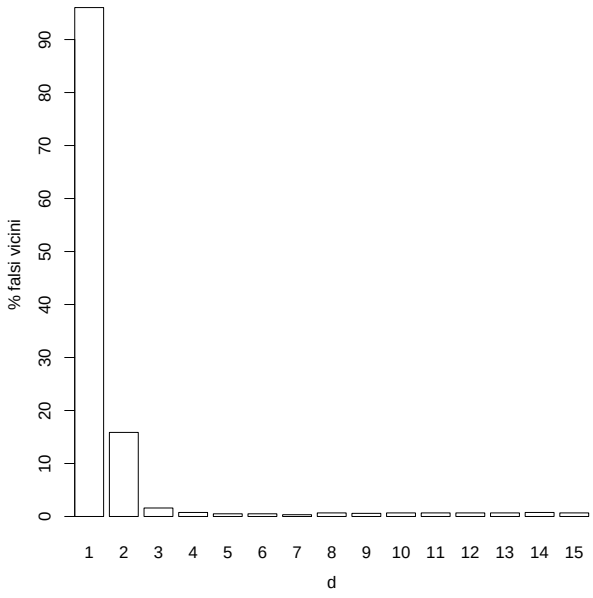


Figura 8.4: Percentuale di falsi vicini in funzione di d per $T = 3$.

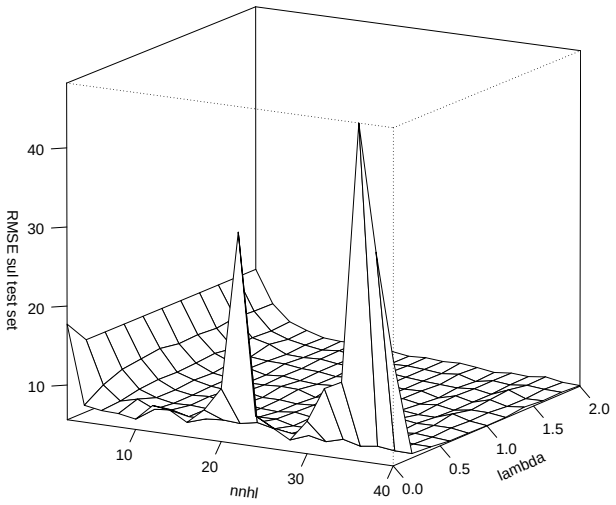


Figura 8.5: Errore sul test in funzione di *nnhl* e λ per la serie del laser.

ha prodotto un $RMSE = 6.12$. Utilizzando questi modelli di macchine d'apprendimento base, si è proceduto con l'addestramento degli ensemble.

Per quanto riguarda i metodi KNN1 e KNN2, $RMSE = 5.07$ è stato il risultato migliore, ottenuto per KNN2 con $k = 41$ (fig. 8.7). Il metodo KNN1 ha fornito un $RMSE = 12.0$ con $k = 9$.

Per l'algoritmo KNN2 l'$RMSE$ risulta minore rispetto a rete neurale e SVM.

Gli algoritmi di bagging e adaboost sono stati lanciati con un numero di mecchine pari a 50. Per quanto riguarda gli ensemble, il risultato migliore è stato ottenuto da adaboost con reti neurali con $RMSE = 4.24$. Riassumendo, i risultati migliori a confronto sono quelli mostrati in tabella 8.1. Gli $RMSE$ per i $Naive_0$ e $Naive_1$ sono rispettivamente $RMSE = 39.6$ e $RMSE = 49.8$, molto peggiori dei risultati in tabella 8.1.

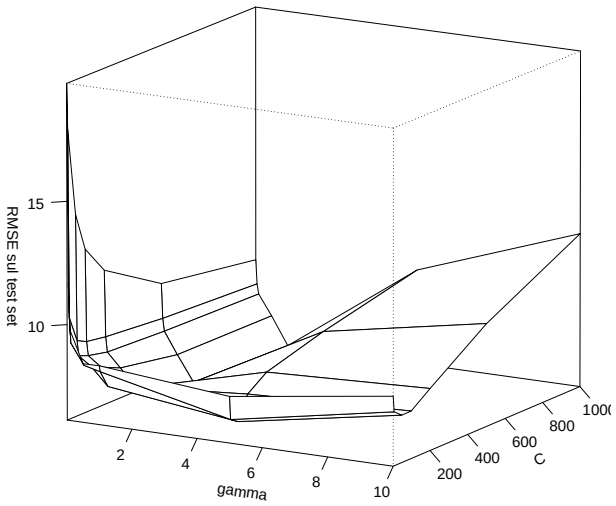
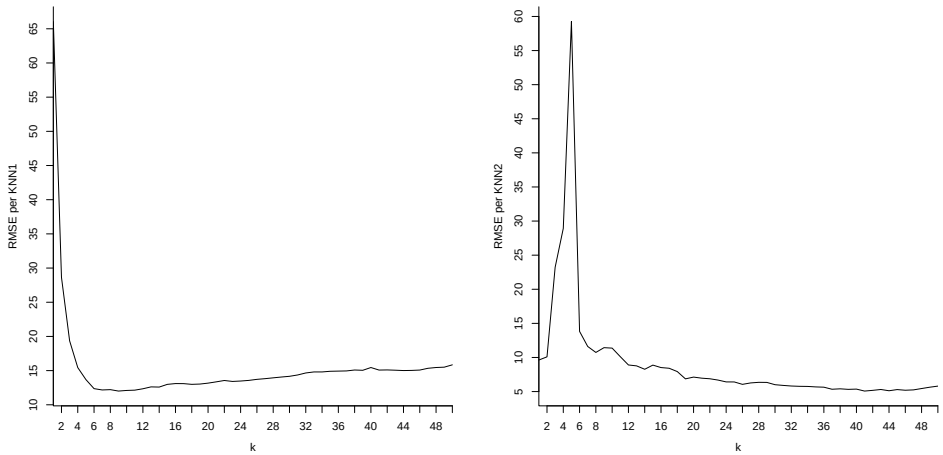
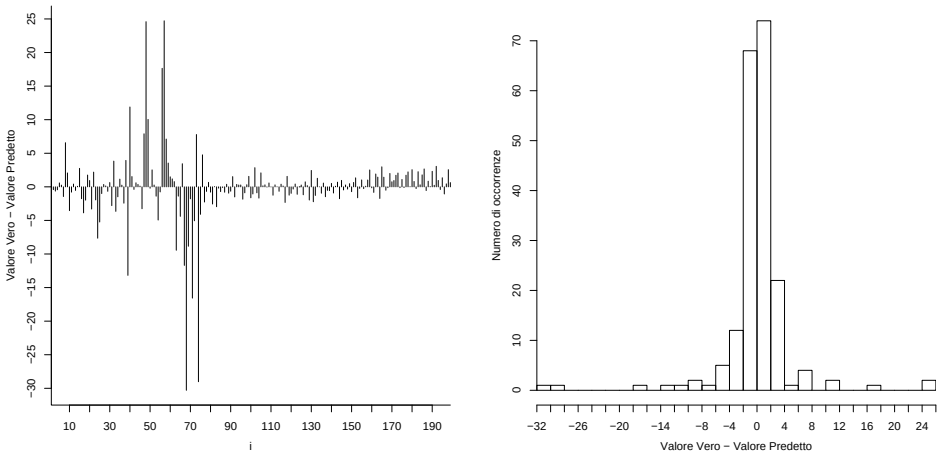


Figura 8.6: Errore sul test commesso dalle SVM per $\varepsilon = 0.01$.



(a) (b)

Figura 8.7: (a) Errore sul test per KNN1 in funzione di k per la serie del laser. (b) Errore sul test per KNN2 in funzione k per la serie del laser.



(a) (b)

Figura 8.8: (a) Andamento temporale dell'errore di predizione commesso dall'algoritmo KNN2. (b) Istogramma delle occorrenze dell'errore commesso dall'algoritmo KNN2.

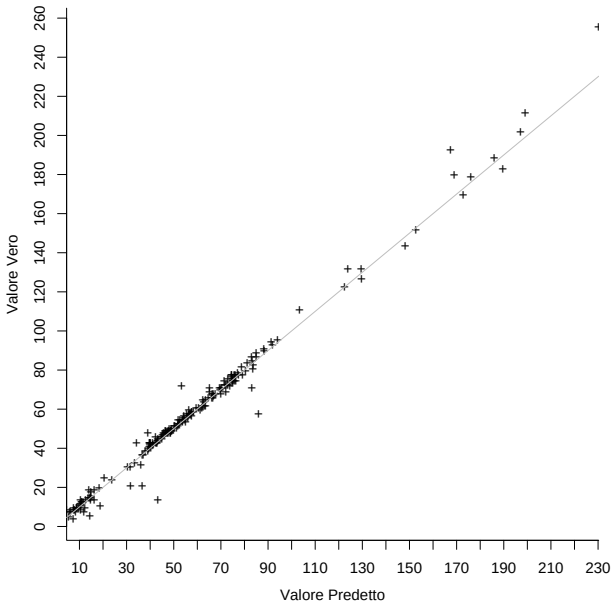
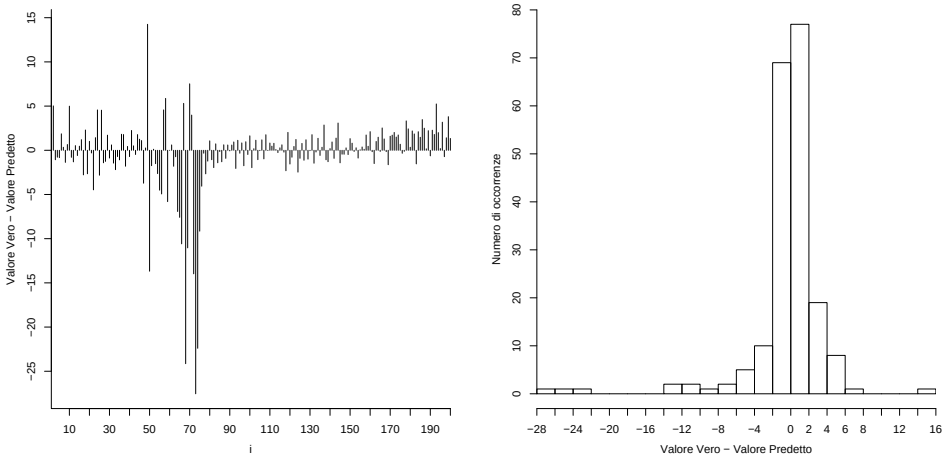


Figura 8.9: Valore vero contro valore predetto per l'algoritmo KNN2.



(a) (b)

Figura 8.10: (a) Andamento temporale dell'errore di predizione commesso dal bagging di reti neurali. (b) Istogramma delle occorrenze dell'errore commesso dal bagging di reti neurali.

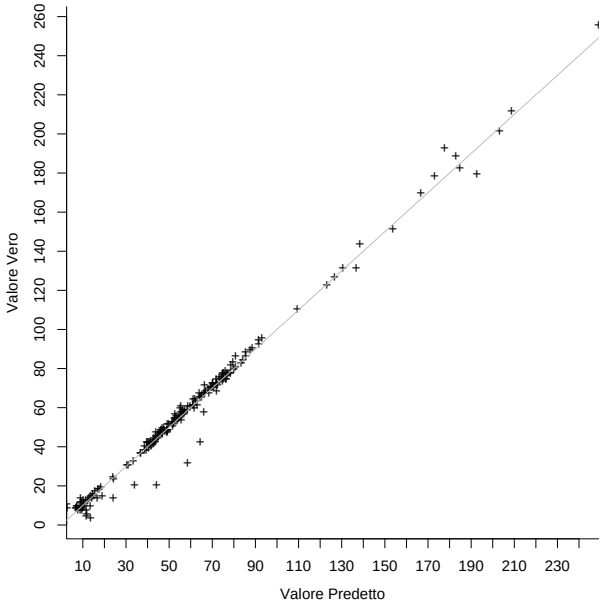


Figura 8.11: Valore vero contro valore predetto per il bagging di reti neurali.

	MLP	SVM	Bagging MLP	Bagging SVM	Adaboost MLP	Adaboost SVM	KNN
$RMSE$	5.60	6.12	5.51	6.87	**4.24**	8.98	5.07
ME	39.3	42.0	30.8	63.0	**27.5**	74.8	30.3

Tabella 8.1: Tabella rissuntiva degli errori delle previsioni sulla serie del laser.

Capitolo 9

Temperatura media giornaliera

In questo capitolo si affronta il problema della predizione di una serie storica di temperatura media giornaliera e successivamente di un suo trend, estratto utilizzando l'SSA (cap. 3). L'estrazione del trend permette di ottenere una serie con caratteristiche tali da permettere una buona previsione su di esso.

In questo capitolo la ricerca dei parametri per le reti neurali e per le SVM non ha seguito lo schema dei capitoli precedenti. Infatti le dimensioni del data base avrebbero richiesto un tempo per l'addestramento troppo lungo. Si è preferito quindi procedere ad una ricerca dei parametri attraverso un tuning manuale.

Anche per la serie di temperatura media giornaliera il confronto dei metodi viene fatto valutando $RMSE$ (eq. 3.1) e ME (eq. 3.4).

9.1 Serie di temperatura media giornaliera

La serie storica in studio è la temperatura media giornaliera a Parigi[1] espressa in decimi di grado.
Essa possiede la temperatura media giornaliera di oltre 15000 giorni, cioè di oltre 40 anni.

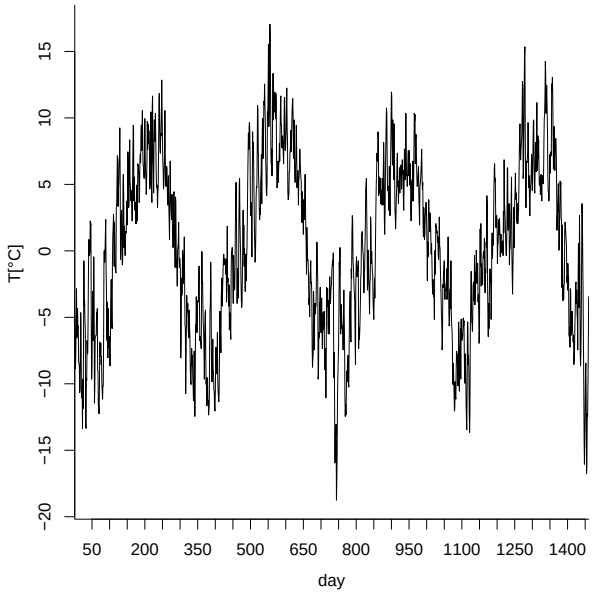


Figura 9.1: La serie corrispondente a 4 anni di temperatura media giornaliera a Parigi.

Si considera il problema della predizione degli ultimi 1000 valori della serie a partire da soli 4000 esempi precedenti per ridurre i tempi di apprendimento. Si applicano gli algoritmi di mutua informazione e falsi vicini per stimare la dimensione d del vettore delle coordinate ritardate. L'analisi della mutua informazione $I(T)$ porta alla scelta di $T = 90$ pari ad un quarto di anno. Con tale valore l'analisi dei falsi vicini indica un valore $d = 5$.

[1]La serie è reperibile al sito: http://www.knmi.nl/samenw/eca. Dati messi a disposizione da: Klein Tank, A.M.G. and Coauthors, 2002. Daily dataset of 20th-century surface air temperature and precipitation series for the European Climate Assessment. Int. J. of Climatol., 22, 1441-1453. La serie si chiama: Paris-14E Parc Montsouris.

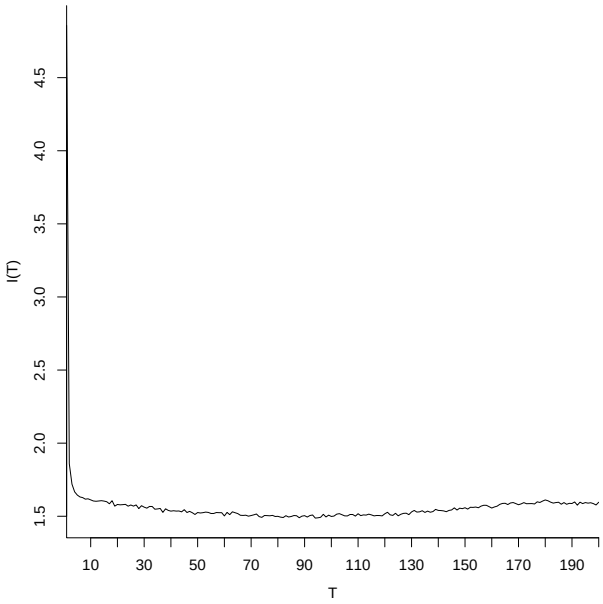


Figura 9.2: Mutua informazione per la serie di temperatura.

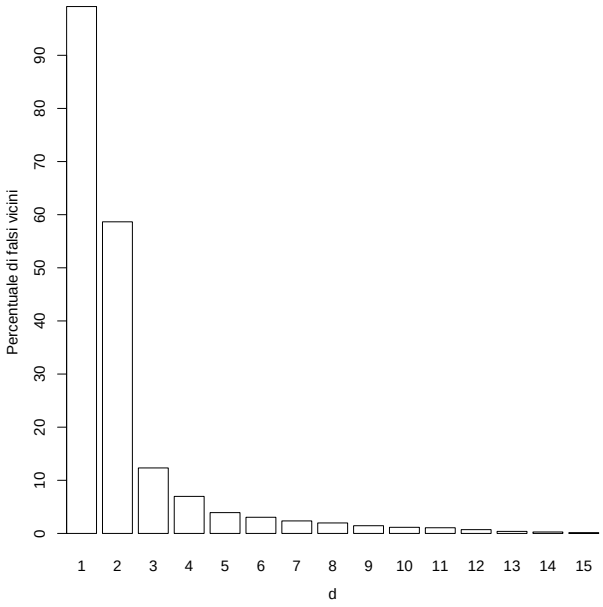
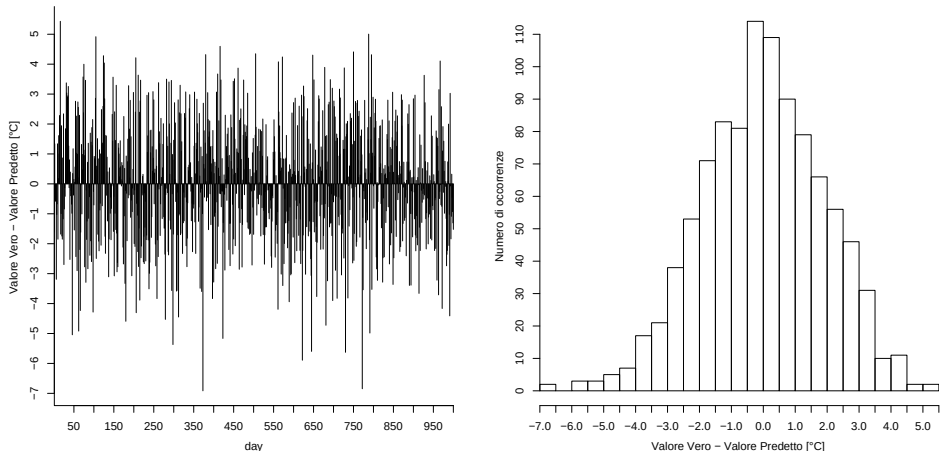


Figura 9.3: Falsi vicini per la parte finale del trend con $T = 90$.

9.1.1 Risultati sulla temperatura media giornaliera

Si è proceduto alla stima dei parametri per le reti neurali e per le SVM e per la rete neurale si è scelto $nnhl = 20$ e $\lambda = 0.2$ ottenendo un $RMSE = 1.93\,°\text{C}$, mentre per l'SVM con $\gamma = 0.05$, $C = 30$ e $\varepsilon = 0.08$ e si è ottenuto un $RMSE = 1.93\,°\text{C}$.



(a) (b)

Figura 9.4: (a) Andamento temporale dell'errore di predizione commesso dalla rete neurale. (b) Istogramma delle occorrenze dell'errore commesso dalla rete neurale.

Dopodiché si sono addestrati gli ensemble Bagging e Adaboost composti da 50 macchine con i parametri citati.

Per quanto riguarda i metodi locali KNN1 e KNN2 si è proceduto variando $k = 1, \cdots, 100$. Il minimo errore per KNN1 $RMSE = 2.08\,°\text{C}$ si è ottenuto con $k = 32$ mentre per KNN2 $RMSE = 1.94\,°\text{C}$ con $k = 99$. Il confronto dei vari risultati è riassunto in tabella 9.1.

Per quanto riguarda $Naive_0$ e $Naive_1$ gli errori $RMSE$ sono stati rispettivamente $RMSE = 2.01\,°\text{C}$ e $RMSE = 2.80\,°\text{C}$.

Gli errori per i base learner sono confrontabili anche se c'è un leggero miglioramento sia nel $RMSE$ che nel ME utilizzando il Bagging di reti

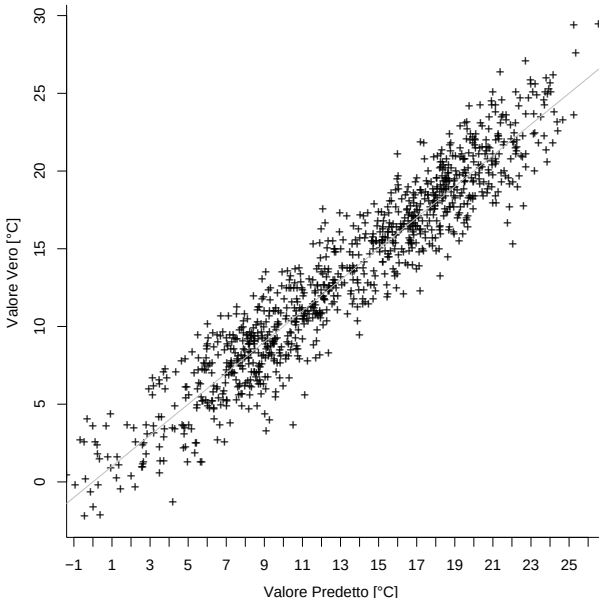
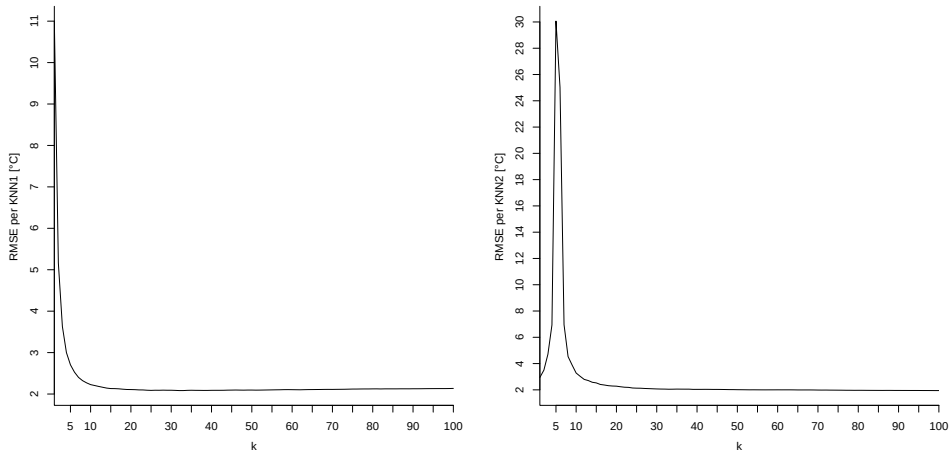


Figura 9.5: Valore vero contro valore predetto dalla rete neurale.



(a) (b)

Figura 9.6: (a) $RMSE$ in funzione di k per KNN1 sulla serie storica di temperatura media giornaliera. (b) $RMSE$ in funzione di k per KNN2 sulla serie storica di temperatura media giornaliera.

	MLP	SVM	Bagging MLP	Bagging SVM	Adaboost MLP	Adaboost SVM	KNN
$RMSE$	1.93	1.93	**1.91**	1.93	2.04	2.02	1.94
ME	6.92	6.98	**6.86**	6.98	8.27	7.00	6.93

Tabella 9.1: Tabella riassuntiva dei risultati sulla serie della temperatura media giornaliera.

neurali.

9.2 Trend di temperatura

Per estrarre il trend si applica l'SSA alla serie portata a media nulla e si effettua l'analisi della varianza spiegata 2.28 in modo da valutare quali onde siano in possesso della gran parte dell'informazione. Si sceglie una finestra temporale $m = 400$, in modo da poter apprezzare variazioni dell'ordine dell'anno. Come si può osservare, la maggior parte dell'informazione è contenuta nelle prime due onde (39.8 % e 33.8 %). Il grafico dell'andamento temporale della serie ottenuta utilizzando solo queste due onde viene confrontato con quello della serie originale in fig. 9.8. Si osserva che le prime due onde contengono l'andamento stagionale della serie. Per ottenere informazioni più dettagliate si aggiungono altre onde. Si è scelto di aggiungere tante onde fino ad arrivare a spiegare il 90 % della varianza totale. Questa condizione viene soddisfatta attraverso la somma di 50 onde. In figura 9.9 si può osservare il confronto fra la serie ottenuta e quella originale.

A questo punto si applicano gli algoritmi di mutua informazione e falsi vicini sul trend di temperatura. Ci si è posti il problema di prevedere gli ultimi 1000 valori del trend. Per fare questo si è provato con l'utilizzo di soli 4000 esempi, in modo da ridurre i tempi di addestramento delle macchine.

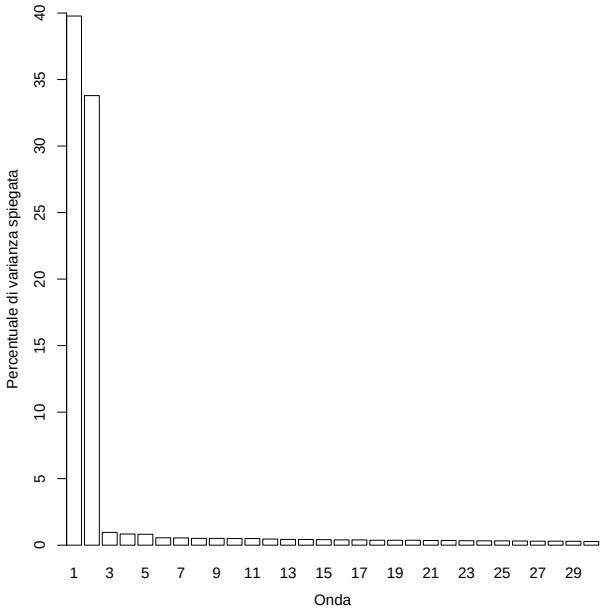


Figura 9.7: Percentuale di varianza spiegata delle onde relative ai primi 30 autovettori.

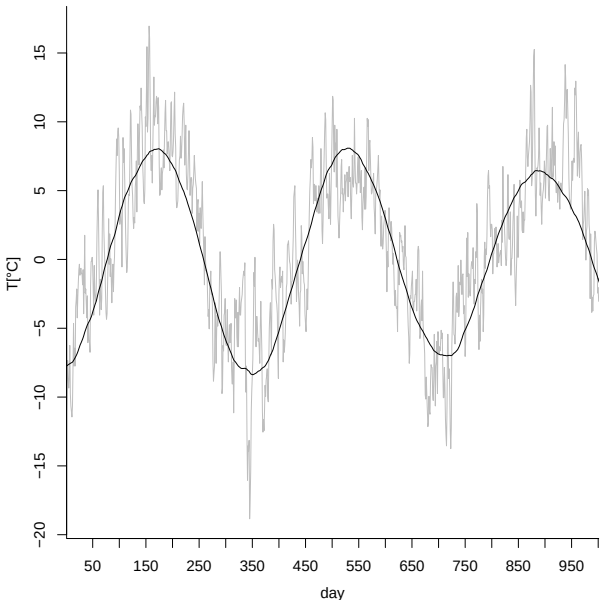


Figura 9.8: Confronto della serie originale (grigio) con quella ottenuta sommando due onde dell'SSA (nero).

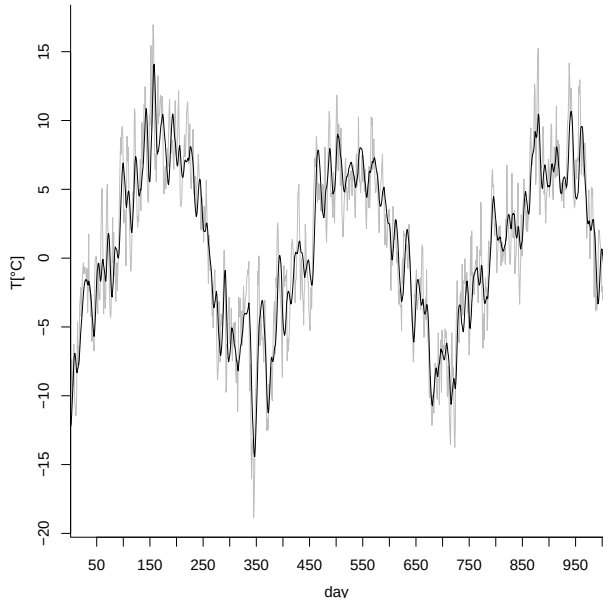


Figura 9.9: Confronto della serie originale (grigio) con quella ottenuta sommando 50 onde dell'SSA, raggiungendo un totale di varianza spiegata pari al 90 % del totale (nero).

Quindi gli algoritmi per la stima di d, vengono applicati alla parte relativa ai 4000 esempi. Nel grafico di figura 9.10 si ha un minimo per $T \simeq 90$, che corrisponde ad un quarto di anno, quindi si sceglie $T = 90$. Con questa scelta, la percentuale di falsi vicini in funzione di d viene mostrata in figura 9.11. Dall'analisi di essa si sceglie $d = 5$.

9.2.1 Risultati sul trend di temperatura

Viste le dimensioni del data base, la stima dei parametri ottimali per SVM e reti neurali avrebbe richiesto un tempo di addestramento molto lungo. Quindi si è provveduto alla scelta dei parametri variandoli, fino a raggiungere dei parametri che garantissero buoni $RMSE$. Per le reti neurali essi sono stati scelti in $nnhl = 20$ e $\lambda = 0.2$, ottenendo un $RMSE = 0.235\,°\text{C}$.

Per SVM, con $\gamma = 0.1$, $C = 40$ e $\varepsilon = 0.04$ si è ottenuto un valore $RMSE = 0.234\,°\text{C}$.

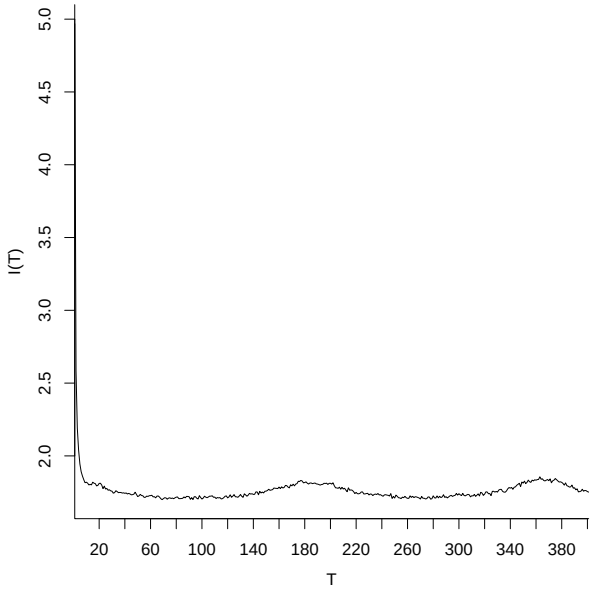


Figura 9.10: Mutua informazione per il trend di temperatura.

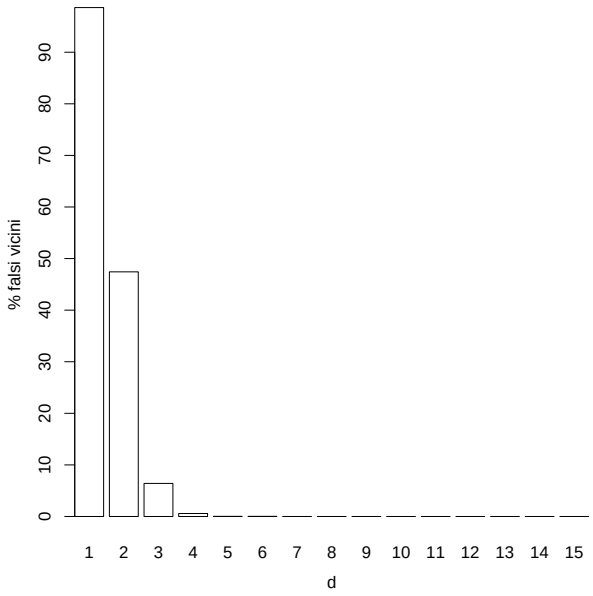
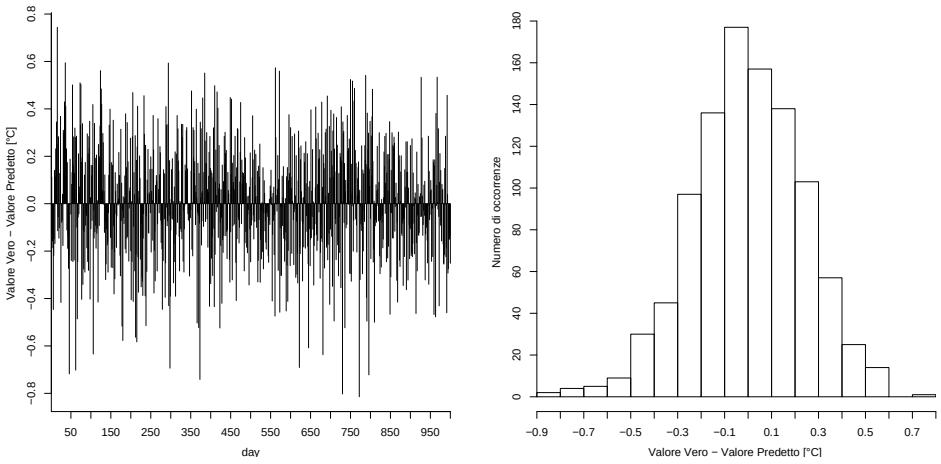


Figura 9.11: Percentuale di falsi vicini per il trend di temperatura con $T = 90$.



(a) (b)

Figura 9.12: (a) Andamento temporale dell'errore di predizione commesso dalla rete neurale sul trend. (b) Istogramma delle occorrenze dell'errore commesso dalla rete neurale sul trend.

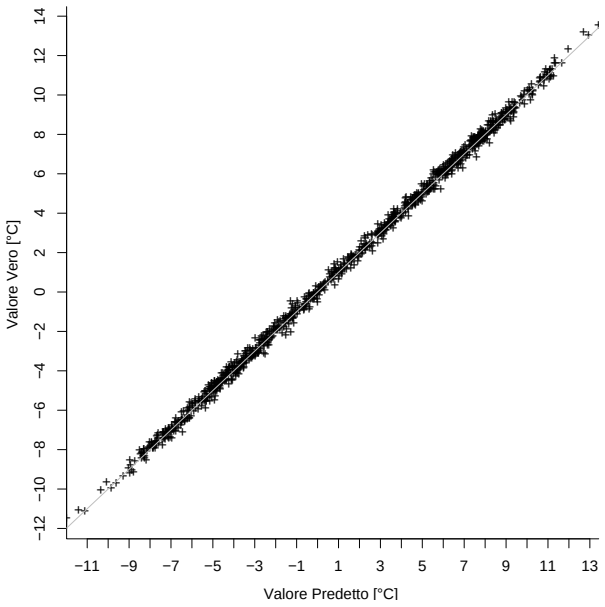
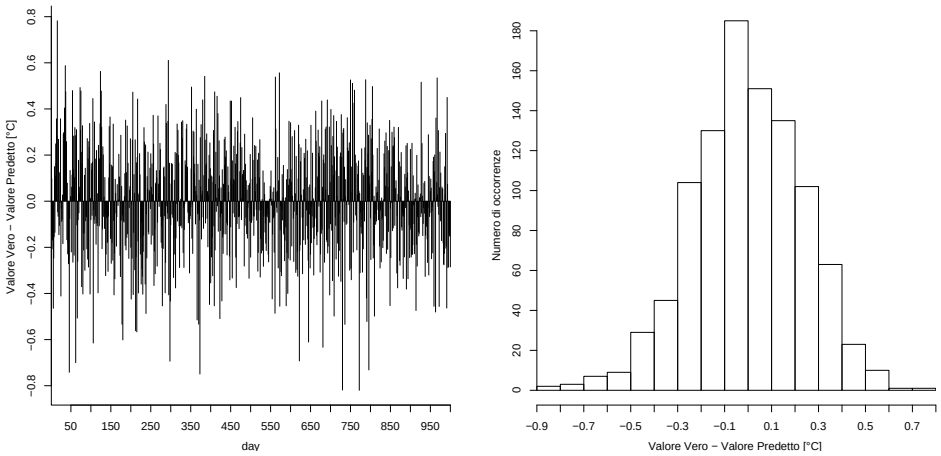


Figura 9.13: Valore vero contro valore predetto per la rete neurale sul trnd.



(a) (b)

Figura 9.14: (a) Andamento temporale dell'errore di predizione commesso dalla SVM sul trend. (b) Istogramma delle occorrenze dell'errore commesso dalla SVM sul trend.

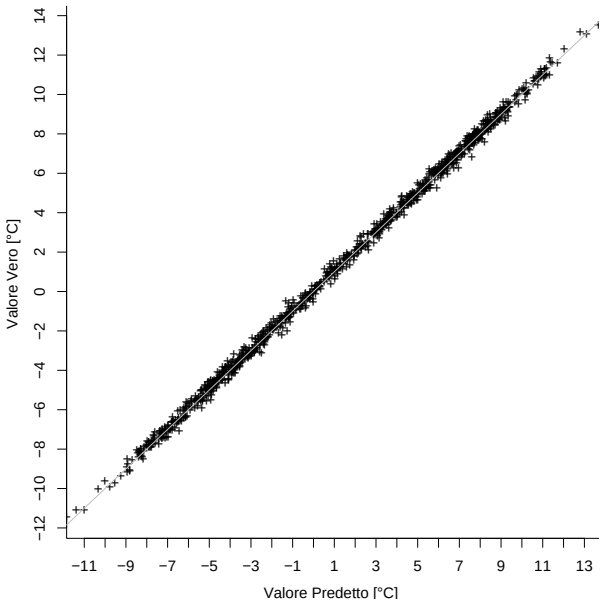
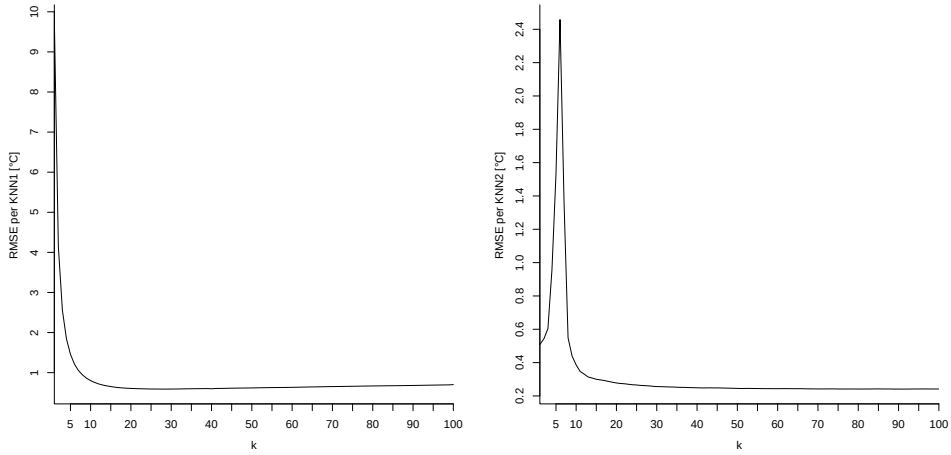


Figura 9.15: Valore vero contro valore predetto per la SVM sul trend.

Per KNN1 il migliore $RMSE = 0.589\,°\text{C}$ è stato ottenuto per $k = 29$, mentre per KNN2 $RMSE = 0.241\,°\text{C}$ per $k = 91$.



(a) (b)

Figura 9.16: (a) $RMSE$ in funzione di k per KNN1 sul trend di temperatura. (b) $RMSE$ in funzione di k per KNN2 sul trend di temperatura.

Gli ensemble sono stati addestrati con 50 macchine scelte sempre con i parametri sopra citati. Gli $RMSE$ per i $Naive_0$ e $Naive_1$ sono rispettivamente

	MLP	SVM	Bagging MLP	Bagging SVM	Adaboost MLP	Adaboost SVM	KNN
$RMSE$	0.235	0.234	**0.235**	0.234	0.252	0.250	0.241
ME	0.814	0.820	**0.812**	0.824	0.824	0.880	0.873

Tabella 9.2: Tabella riassuntivi di risultati sulle previsioni del trend di temperatura media giornaliera.

$RMSE = 0.544\,°\text{C}$ e $RMSE = 0.280\,°\text{C}$.

Anche per il trend di temperatura non vi sono sostanziali miglioramenti forniti da Bagging e Adaboost. Addirittura Adaboost tende a peggiorare la previsione, mentre Bagging lascia sostanzialmente inalterato l'errore.

Conclusioni

Uno dei più recenti e promettenti settori dell'apprendimento automatico è quello degli ensemble di macchine di apprendimento. Fino ad ora gli ensemble sono stati studiati principalmente nell'ambito dei problemi di classificazione. L'obiettivo di questa tesi è stato lo studio preliminare delle proprietà di generalizzazione dei metodi di ensemble nell'ambito della previsione delle serie storiche.

In questa tesi è stato affrontato il problema di previsione di serie storiche attraverso trasformazione in un problema di regressione. Sono stati applicati due metodi di ensemble, Bagging e Adaboost, che hanno dato buoni risultati in problemi di classificazione. L'implementazione è stata fatta in linguaggio R con routines ottimizate scritte in C. I metodi sono stati testati su un insieme di data base, di cui uno sintetico per regressione, uno sintetico per previsione e due fisici per previsione. I base learner utilizzati sono stati i percettroni multistrato e le macchine a supporto vettoriale (SVM). I risultati hanno evidenziato che nella maggior parte dei casi le prestazioni dei base learners sono confrontabili, anche se per la serie di Lorenz e del laser le SVM hanno fornito una prestazione leggermente peggiore. Questo si può ricondurre in parte al fatto che la SVM richiedono il tuning di tre parametri contro i due delle reti neurali. Comunque in tutti i casi i risultati sono stati migliori dei predittori $Naive_0$ e $Naive_1$.

Riguardo ai metodi di ensemble, i risultati ottenuti hanno una dipendenza dal tipo di data base. In tutti i casi, tranne nella serie storica di temperatura,

hanno portato a dei miglioramenti consistenti. In particolare nel problema di regressione, grazie al Bagging di SVM si è ottenuto un miglioramento del $RMSE_2$ del 48 % circa rispetto alla migliore SVM. Per la serie di Lorenz il miglioramento Adaboost con reti neurali è stato del 5 % circa del $RMSE$ rispetto alla migliore rete neurale. Per quella del laser si è ottenuto un miglioramento sul $RMSE$ della migliore rete neurale del 24 % circa grazie ad Adaboost con reti neurali. Nel caso della serie di temperatura media giornaliera il bagging con MLP ha fornito un leggero miglioramento sul ME. Per ottenere tutti questi risultati sono stati addestrati oltre trentamila learner.

La varietà dei risultati ottenuti suggerisce il passo successivo che sarà quello di effettuare un'analisi dettagliata delle caratteristiche dei singoli base learner, basata sulla scomposizione dell'errore di generalizzazione in bias e variance, per poter selezionare quelli più adatti a costituire l'ensemble, sulla base del lavoro in via di pubblicazione [18]. Un analisi di questo tipo richiederà un'estensione significativa del lavoro sperimentale per poter essere esaustiva.

Appendice A

Il linguaggio R

L'ambiente che ho utilizzato per lo sviluppo software si chiama R. Si può utilizzare R in modalità batch, o scrivendo dei programmi come i comuni linguaggi strutturati quali il C. La dichiarazione degli oggetti, come per esempio le variabili, avviene nel momento stesso in cui si effettua una assegnazione. Da quel momento, fino all'eliminazione esplicita, l'oggetto è disponibile e può essere utilizzato. L'assegnazione di una variabile `a` avviene attraverso l'operatore `<-`:

```
> a <- 3
```

Per le variabili sono disponibili tutte le operazioni possibili fra scalari.

R mette a disposizione anche i vettori, la cui assegnazione avviene tramite la funzione `c()`:

```
> v <- c(1, 2, 3, 4)  # Crea un vettore di 4 elementi
```

Alternativamente all'assegnazione precedente si può sostituire:

```
> v <- c(1:4)  # Crea un vettore con gli interi da 1 a 4
```

Oltre ai vettori si possono utilizzare le matrici e gli array:

```
> M <- matrix(c(1,2,1,2), nrow=2, ncol=2)
> A <- array(c(1:12), dim=c(2,2,3))
```

Per tutti questi oggetti c'è la possibilità di effettuare somme, prodotti scalari (`%*%`) fra vettori, prodotti matriciali (`%*%`) e operatori più complicati per i quali si rimanda alla documentazione sul web.

La grande potenza di R sta nella gestione degli indici per selezionare delle parti di interesse di questi oggetti. L'espressione:

```
> A[,,-1]
```

stampa il contenuto dell'array `A` eliminando la parte per cui la terza componente è uno. Questo, accanto alla possibilità di inserire nelle parentesi quadre dei vettori di indici, permette in maniera molto compatta e semplice, di selezionare delle parti anche in strutture complesse.

Le strutture più complesse sono le liste che consistono in una aggregazione di più oggetti anche di tipi diversi fra loro.

```
> l <- list(a, v, M, A)
```

La selezione di un elemento all'interno della lista avviene attraverso le doppie parentesi quadre `[[]]`.

```
> l[[1]]  # Stampa il contenuto di a
```

Un tipo speciale di lista è il dataframe.

Segue un breve esempio di codice scritto in linguaggio R che implementa gli algoritmi KNN1 e KNN2. La funzione da minimizzare si chiama `LocalMapping` e viene definita all'inizio. Viene creato il data base di esempi e viene diviso in training set e test set. Quindi viene creata una matrice in cui ogni colonna contiene gli indici dei k vicini ad ogni elemento del test set. Questa struttura dati viene riempita dalla routine `knn` scritta in linguaggio C che viene chiamata attraverso la funzione `.C`.

Al momento della chiamata della routine, viene creata una copia di tutti i parametri che le vengono passati. Di questa copia viene passato il puntatore. Da notare che è necessario specificare, al momento della chiamata, il tipo dei parametri. Il prototipo della funzione `knn` è:

```
void knn (double *x, int *nx_pointer, int *nexamples_pointer,
          int *de_pointer, int *knn, int *k_pointer);
```

Il programma scritto in C non ha bisogno di un "main" che viene rimpiazzato dalla funzione stessa. La compilazione avviene tramite l'istruzione:

```
R CMD SHLIB knn knn.c
```

che genera il file `knn.so` che viene caricato tramite l'istruzione:

```
> dyn.load("knn.so")
```

Ecco il codice:

```
# Definizione del funzionale da minimizzare per KNN2
LocalMapping <- function(AB) {
  A <- AB[1:de]
  B <- AB[de+1]
  res <- 0
  for (j in 1:k)
    res <- res + sum((A %*% vect1[j,] + B - vect2[j,de])^2)
}
# Carica la funzione scritta in C contenuta in knn.so
dyn.load("knn.so")

de <- 3  # Dimensione di embedding
k <- 50  # Numero di vicini da trovare per ogni vettore di input
# Costruzione della struttura dati contenente gli indici dei
# vicini di ogni input del test set
knn <- matrix(0, nrow = k, ncol = length(x)-nexamples-1)

# Riempimento della matrice knn
knn <- .C("knn", as.double(x), as.integer(length(x)), as.integer(nexamples-1),
                 as.integer(de), as.integer(knn), as.integer(k))[[5]]
dim(knn) <- c(k, length(x)-nexamples-1)

# Crea il training set e il test set
dataset <- list()
for (i in 1:de)
  dataset$input <- cbind(dataset$input,x[i:(length(x)-de+i-1)])
dataset$output <- x[(de+1):length(x)]

trainingset <- list()
trainingset$output <- dataset$output[1:(nexamples-de+1)]
```

```
testset <- list()
testset$input <- dataset$input[(nexamples-de+2):(length(x)-de),]
testset$output <- dataset$output[(nexamples-de+2):(length(x)-de)]

predtest1 <- rep(0,times = dim(knn)[2])
predtest2 <- predtest1

# KNN1
for (i in 1:k)
  predtest1 <- predtest1 + dataset$output[knn[i,]]
predtest1 <- predtest1 / k

# KNN2
params <- rep(0,times = de+1)
for (i in 1:length(predtest1)) {
  vect1 <- matrix(dataset$input[knn[1:k,i],], nrow = k)
  vect2 <- matrix(dataset$input[knn[1:k,i]+1,], nrow = k)

  model <- optim(params, LocalMapping, control = list(maxit = 10000),method = 'BFGS')
  A <- model$par[1:de]
  B <- model$par[de+1]
  predtest2[i] <- A %*% testset$input[i,] + B
}
```

Bibliografia

[1] H.D.I. Abarbanel. *Analysis of Observed Chaotic Data*. Springer, New York, 1996.

[2] E.L. Allwein, R.E. Schapire, and Y. Singer. Reducing multiclass to binary: a unifying approach for margin classifiers. *Journal of Machine Learning Research*, 1:113–141, 2000.

[3] D.K. Arrowsmith and C.M. Place. *An introduction to Dynamical Systems*. Cambridge University Press, 1990.

[4] R. Avnimelech and N. Intrator. Boosting regression estimators. *Neural Computation*, 11:491–513, 1999.

[5] Z. Baratçouğlu and E. Alpaydin. A comparison of model aggregation methods for regression. In Okyay Kaynak, Ethem Alpaydin, Erkki Oja, and Lei Xu, editors, *ICANN*, volume 2714 of *Lecture Notes in Computer Science*, pages 76–83. Springer, 2003.

[6] L. Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.

[7] P. Bülhmann and B. Yu. Boosting with the l_2-loss: Regression and classification. Technical Report 98, Eidgenössische Technische Hochschule, August 2001.

[8] J.L. Casti. *Dynamical Systems and Their Applications: Linear Theory*. Academic Press, New York, 1977.

[9] V. N. Cherkassky and F. Mulier. *Learning from data: Concepts, Theory and Methods*. Wiley & Sons, New York, 1998.

[10] T.G. Dietterich. Ensemble methods in machine learning. In *Multiple Classifier Systems. First International Workshop, MCS2000, Cagliari, Italy*, pages 1–15. Springer-Verlag, 2000.

[11] T.G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision tress: Bagging, boosting and randomization. *Machine Learning*, 40(2):139–158, 2001.

[12] H. Drucker. Improving regressors using boosting techniques. In Douglas H. Fisher, editor, *ICML*, pages 107–115. Morgan Kaufmann, 1997.

[13] N. Duffy and D. Helmbold. Boosting methods for regression. *Machine Learning*, 47:153–200, 2002.

[14] R. Kohavi E. Bauer. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36:105–142, 1999.

[15] J. B. Elsner and A. A. Tsonis. *Singular Spectrum Analysis: A New Tool in Time Series Analysis*. Plenum, New York, 1996.

[16] Y. Freund and R.E. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the 13th International Conference on Machine Learning*, pages 148–156. Morgan Kauffman, 1996.

[17] Y. Freund and R.E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55:119–139, 1997.

[18] T. G. Dietterich G. Valentini. Support vector machines for the development of SVM-based ensemble methods. accepted for publication Journal of Machine Learning Research, MIT Press.

[19] M. Ghil, M.R. Allen, M.D. Dettinger, K. Ide, D. Kondrashov, M.E. Mann, A.W. Robertson, A. Saunders, Y. Tian, F. Varadi, and P. Yiou. Advanced spectral methods for climatic time series. *Reviews of Geophysics*, 40:1–1–1–41, 2002.

[20] J. Hadamard. Sur les problemes aux derivees parielies et leur signification physique. Bull. Univ. of Princeton, pages 49-52, 1902.

[21] E.H.M. Hogenboom, W. Klische, C.O. Weiss, and Godone A. Instabilities of a homogeneously broadened laser. *Phisical Review Letters*, 55(23):2571–2574, 1985.

[22] U. Hübner, N.B. Abraham, and C.O. Weiss. Dimensions and entropies of chaotic intensity pulsations in a single-mode far-infrared NH_3 laser. *Phisical Review A*, 40(11):6354–6365, 1989.

[23] V. Kecman. *Learning and Soft Computing*. MIT Press, Cambridge, 2001.

[24] Y.A. Kravtsov. *Limits of Predictability*. Springer-Verlag, Berlin Heildeberg, 1993.

[25] R. Mañé. On the dimension of the compact invariant sets of certain non-linear maps. In D.A. Rand and L.-S. Young, editors, *Dynamical Systems and Turbulence*, volume 898 of *Lecture Notes in Mathematics*, pages 230–242, Warwick 1980, 1981. Springer-Verlag, Berlin.

[26] F. Montarsolo. Realizzazione di un toolbox per la modellazione di serie temporali non lineari. Laurea thesis in computer science, University of Genova, 1998.

[27] R.E. Schapire and Y. Singer. Boostexter: A boosting-based system for text categorization. *Machine Learning*, 39:135–168, 2000.

[28] F. Takens. Detecting strange attractors in turbulence. In D.A. Rand and L.-S. Young, editors, *Dynamical Systems and Turbulence*, volume 898 of *Lecture Notes in Mathematics*, pages 366–381, Warwick, 1981. Springer-Verlag, Berlin.

[29] G Valentini and F. Masulli. Ensembles of learning machines. In R. Tagliaferri and M. Marinaro, editors, *Neural Nets Wirn Vietri-02*, volume LNCS 2486, pages 3–19, Heidelberg (Germany), 2002. Series *Lecture Notes in Computer Sciences*, Springer-Verlag. *(invited review)* .

[30] R. Vautard and M. Ghil. Singular-spectrum analysis in nonlinear dynamics, with applications to paleoclimatic time series. *Physica D*, 35:395–424, 1989.

[31] R. Vautard, P. You, and M. Ghil. Singular-spectrum analysis: A toolkit for short, noisy chaotic signals. *Physica D*, 58:95–126, 1992.

[32] A.S. Weigend and N.A. Gershenfeld, editors. *Proceedings of the NATO Advanced Research Workshop on Comparative Time Serie Analysis held in Santa Fe, New Messico, May 14-17, 1992*, volume XV of *Proceedings Volume, Santa Fe Institute Studies in the Sciences of Complexity*. Addison Wesley Publishing Company, 1993.

[33] C.O. Weiss and J. Brock. Evidence for Lorenz-type chaos in a laser. *Phisical Review Letters*, 57(22):2804–2806, 1986.