Università degli Studi di Padova

Tesi di Laurea Magistrale

Classificazione tramite motion primitives di dati Cyberglove per applicazioni robotiche

Autore:
Giacomo Cacco - 1063269

Relatore:
Emanuele Menegatti

Correlatore:
Stefano Michieletto

6 luglio 2017

Sommario

Lo sviluppo e l'analisi di protesi robotiche comandabili tramite segnali biometrici è diventato un argomento di studio molto trattato. Una delle tecniche di analisi consiste nella creazione di classificatori di movimenti, tramite i quali si riesce a capire quale movimento una persona vorrebbe eseguire per farlo fare poi alla protesi robotica, tramite una serie di comandi appositamente pre-impostati che eseguano il movimento voluto.

Un classificatore necessita di movimenti di esempio di cui è nota la tipologia per la sua costruzione. Queste informazioni saranno prese dal *dataset* NinaPro, reperibile da internet nel quale sono presenti un numero ampio, ma limitato di tipologie di movimenti da poter utilizzare come esempi. A questi viene applicata la tecnica delle *Dynamic Motion Primitives* (DMP) per l'estrazione di *features* utilizzate per caratterizzare i movimenti e permetterne la classificazione.

Verranno testate diverse tipologie di classificatori, quali Classification tree, GMM e modelli ECOC con diverse configurazioni, al fine di trovare il migliore da utilizzare assieme alle *features* estratte con le DMP.
Durante lo sviluppo di questi classificatori sono state riscontrate diverse problematiche, che causavano bassi livelli di precisione per i classificatori creati. Tramite un lavoro di *preprocessing* sui movimenti forniti dal *dataset* queste problematiche sono state risolte in modo da raggiungere livelli di accuratezza elevati nel riconoscere movimenti di soggetti differenti da quelli presenti nel *dataset*.

Le *features* per rappresentare i movimenti derivano da segnali provenienti da un guanto *Cyberglove2* a cui vengono applicate le DMP, utilizzate poi per classificare i movimenti. Il nostro scopo sarà quello di comprendere le problematiche che possono sorgere nell'analizzare un *dataset* così ampio (17 movimenti che coinvolgono 22 giunti) e ricco di soggetti differenti (40 persone) attraverso l'uso di DMP.

Infatti, le DMP si sono dimostrate efficaci in diversi casi in letteratura nel modellare il movimento di robot, in particolare manipolatori. Solo pochi test sono stati effettuati su sistemi così complessi e articolati e siamo convinti che questo studio possa essere utilizzato come un'importante base di partenza per lavori futuri che vogliano ampliare i test effettuati finora.

Il codice realizzato durante il lavoro è stato scritto principalmente in linguaggio Matlab, data la sua facilità di utilizzo per l'elaborazione di grandi quantità di dati e la presenza di pacchetti per l'estrazione delle DMP e costruzione di alcune tipologie di classificatori. Per i modelli GMM sono stati utilizzati pacchetti di codice C++ in quanto la loro creazione richede un tempo di calcolo elevato, ed il linguaggio C++ risulta essere molto più veloce rispetto a quello Matlab.

Abstract

The development and analysis of robust prosthetic robots via biometric signals has become a highly studied subject. One of the techniques of analysis is creations of motion classifiers through which one can know out what movement a person would want to perform, to do the robotic prosthesis then.

A classifier needs sample movements whose type is known for its construction. This examples will be taken from the NinaPro dataset, available from the internet, which has a large but limited number of types of movements to be used as samples These are applied to the Dynamic Motion Primitives (DMP) technique for extracting features used to characterize movements and allow them to be classified.

Various types of classifiers will be tested, such as Classification tree, GMM and ECOC model with different configurations in order to find the best one to use with features extracted from DMP.
During the development of these classifiers several problems have been encountered hich caused low levels of precision for classifiers. Through a preprocessing work on the movements provided by the dataset, these issues have been resolved, to achieve high precision levels in recognizing motion of subjects other than those in the dataset

Features to represent movements are derived from signals from a *Cyberglove2* glove to which DMPs are applied, used to classify movements. Our aim will be to understand the issues that can arise in analyzing such a large dataset (17 movements involving 22 joints) and rich in different subjects (40 people) through the use of DMP. In fact, DMPs have proved effective in several cases in literature in modeling the movement of robots, in particular manipulators. Only a few tests have been carried out on such complex and articulated systems and we are convinced that this study can be used as an important starting point for future jobs.

The code written during the work was mainly in Matlab language because its ease of use for processing large amounts of data and presence of packages for the extraction of DMPs and the construction of some types of classifiers. C++ code packs have been used for GMM models since their creation has a high computation time and the C ++ language is much faster than Matlab.

Acronimi

DMP Dynamic Motion Primitive

ECOC Error-Correcting Output Codes

EM Expectation Maximization

EMG Electromyography

GMM Gaussian Mixture Model

k-NN k-Nearest Neighbors

LWPR locally weighted projection regression

LWR locally weighted regression

NinaPro Non-Invasive Adaptive Hand Prosthetics

PCA Principal Component Analysis

sEMG Surface Electromyography

SVM Support Vector Machine

xcorr Cross Correlation

xcov Cross Covariance

Indice

Capitolo 1

Introduzione

Lo scopo di questa tesi è quello di realizzare un classificatore per un insieme di movimenti della mano al fine di riuscire a capire, per un movimento sconosciuto, a quale tipologia di movimento appartiene tra quelli classificati.
Per la costruzione di un qualunque classificatore si rende necessario avere un insieme di movimenti di esempio, da utilizzare come riferimenti per la classificazione successiva di movimenti sconosciuti.

Una possibile applicazione in campo robotico del classificatore che andremo a costruire consiste nel caso in cui il movimento sconosciuto debba essere riprodotto da una mano robotica. Tramite l'utilizzo del classificatore si potrà capire a quale tipologia di movimento appartiene, per poi fornire alla mano robotica i comandi necessari alla simulazione del movimento voluto. In alternativa si potrà fornire il classificatore anche solo per capire la tipologia di un movimento sconosciuto fornito a terze parti, che potranno poi utilizzare tale informazione per i propri scopi specifici.

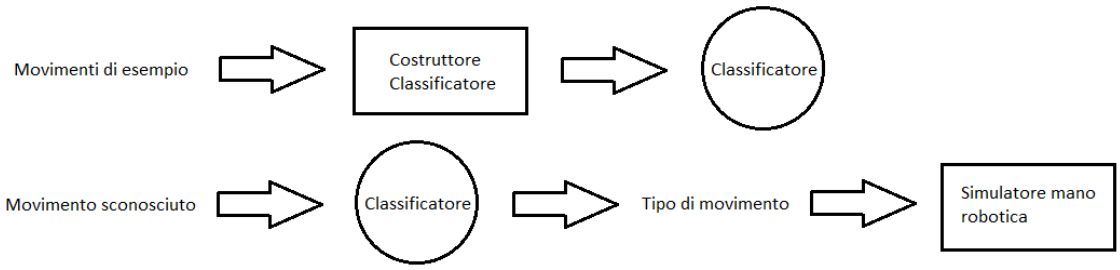

Figura 1.1: Schema a blocchi per costruzione ed utilizzo di un classificatore

I movimenti di esempio con cui costruire il classificatore saranno presi dal *dataset* del progetto NinaPro (Capitolo 5), nel quale i movimenti sono rappresentati tramite una serie di segnali provenienti da un sistema Surface Electromyography (sEMG) e da un guanto sensoriale *CyberGlove2* indossati dalle persone che compiono il movimento. I dati provenienti dal guanto, che andremo ad utilizzare per la costruzione dei classificatori, rappresentano la variazione di angolo compiuta da particolari punti della mano (chiamati giunti) durante il movimento, catturati da speciali sensori, uno per ogni giunto, per un totale di 22 sensori.

Figura 1.2: Guanto *Cyberglove2*

I movimenti di esempio saranno dunque formati da una più segnali che variano nel tempo. Per poterli classificare bisogna trovare una metodologia che consenta di rappresentare questi segnali tramite dei valori (o *features*) per renderli confrontabili tra loro.
La scelta è ricaduta sulle Dynamic Motion Primitive (DMP), utilizzate tipicamente come strategia di controllo e pianificazione traiettorie di movimento. Combinando assieme un numero limitato di funzioni base si possono ricostruire un insieme infinito di traiettorie, rappresentanto anch'esso tramite funzioni di spazio e tempo. Il contributo che ciascuna funzione base deve dare nella ricostruzione viene rappresentato da un peso, calcolato dalle DMP. Oltre ai pesi, le DMP calcolano una serie di altri parametri, utilizzati di norma per comandare semplici robot al fine di fargli compiere la traiettoria desiderata, partendo da una di esempio tipicamente fornita da movimenti compiuti da persone.

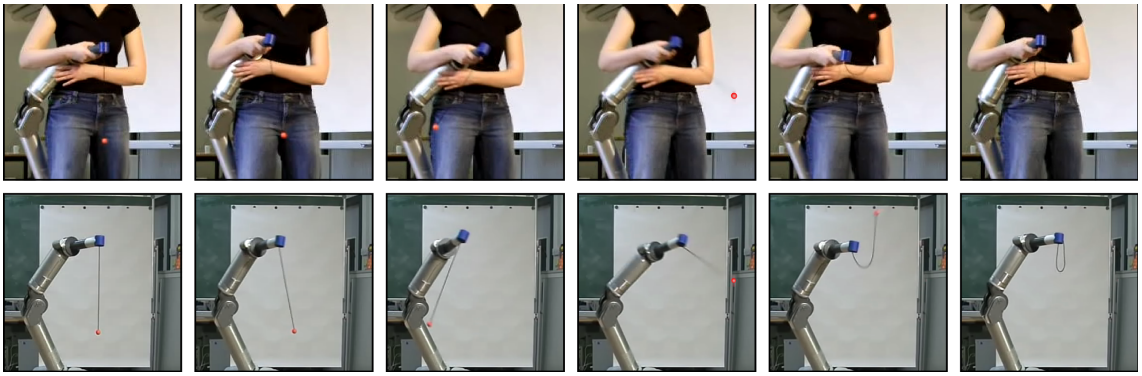

Figura 1.3: Dimostrazine di movimento e replica eseguita da un robot

Utilizzando come traiettorie di esempio gli andamenti dei giunti, si potranno utilizzare i pesi delle funzioni base come *features*. Ogni movimento sarà quindi rappresentato da un insieme di più *features* per ognuno dei 22 giunti.
Potrebbe rendersi necessaria una fase di *preprocessing*, nella quale i segnali dei giunti vengono elaborati al fine di renderli compatibili al software di calcolo delle DMP o per migliorarne la qualità ed ottenere delle *features* con cui sarà più facile classificare i vari movimenti.

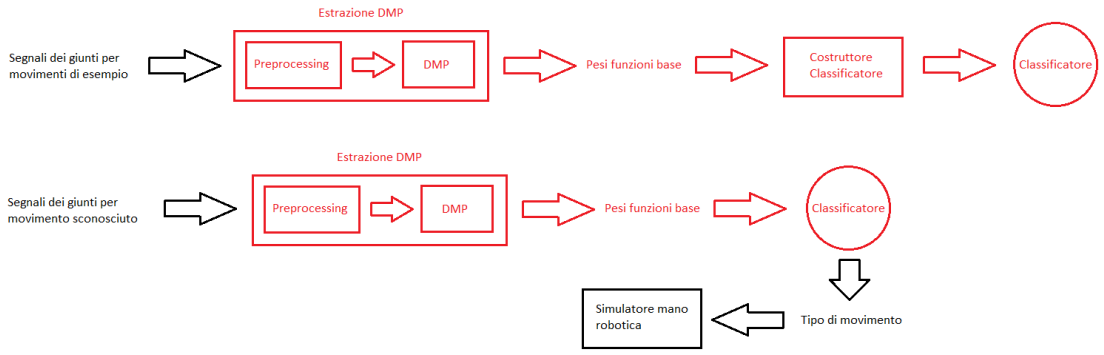

Figura 1.4: Schema a blocchi: evidenziate in rosso le parti che si vogliono sviluppare durante la tesi

L'idea base all'utilizzo delle DMP per rappresentare i movimenti sta nel fatto che se due movimenti sono uguali tra loro, i giunti della mano dovrebbero generare segnali uguali o molto simili, considerando l'improbabile realizzazione di movimenti perfettamente identici. Questa somiglianza si pensa debba ripercuotersi anche

nei pesi calcolati per le funzioni base, dato che viene utilizzato sempre lo stesso set di funzioni per ricostruire l'andamento dei giunti della mano. Per due movimenti differenti, i pesi per le funzioni base dovrebbero essere differenti per un qualche numero di giunti.

La motivazione principale per la quale useremo i segnali provenienti dal guanto *CyberGlove2* è la scarsa presenza di lavori sull'analisi di movimenti della mano avendo a disposizione una quantità così elevata di informazioni legate al movimento della mano. Il guanto ne produce una quantità elevata, considerando che è ricoperto da un totale di 22 sensori, e questa mole di dati potrebbe richiedere tempi di analisi elevati.
Inoltre, studi sull'utilizzo di questa tipologia di dati assieme alle DMP risulta addirittura quasi inesistente, soprattutto considerando l'elevato numero di soggetti diversi che eseguono le dimostrazioni di movimento in questo contesto.

Capitolo 2

Stato dell'arte

Lo scopo principale per cui vengono utilizzate le DMP è quello di pianificare una traiettoria di movimento, da far successivamente percorrere ad un robot. Per far ciò ogni giunto di un robot deve essere correttamente comandato in ogni istante del movimento. Le DMP rirescono a calcolare le informazioni necessarie affichè ciò avvenga, in particolare fornendo, per ogni istante, posizione, velocità ed accelerazione che ogni giunto deve assumere. Questi parametri vengono calcolati tramite apprendimento, cioè bisogna fornire al sistema di calcolo delle DMP uno o più esempi del movimento da far eseguire al robot.

I giunti dei robot vengono mossi grazie a dei motori, è necessario porre dei vincoli per riuscire a completare o rendere possibile il movimento. Ad esempio, è necessario che velocità ed accelerazione iniziali siano pari a zero per non danneggiare i motori dei giunti. Per questo, alla formulazione iniziale proposta nel 2002 da A. J. Ijspeert et al. sono state create nel tempo delle varianti, sia sulla tipologia di sistema dinamico da utilizzare, che sul tipo di *forcing term* da aggiungere.

2.1 Utilizzo tipico delle DMP

Molto spesso le DMP vengono utilizzate per la pianificazione di traiettorie per robot manipolatori di pochi giunti o per camminate robotiche, che devono muoversi dentro uno spazio a 3 dimensioni. Le traiettorie da eseguire sono tipicamente di imitazione umana: in precedenza una persona, a cui vengono applicati dei sensori per registrare il movimento sull'arto che lo eseguirà, compie il movimento per intero e successivamente i dati registrati vengono forniti al sistema di calcolo delle

DMP, come nel lavoro di R. Mao et al.[1] in cui ad un robot manipolatore Baxter vengono mostrate le posizioni da assumere con gli arti robotici al fine di sollevare una pallina.
Altri esempi simili risultano essere quelli di S. Schaal[2], in cui si cerca di far riprodurre ad un robot movimenti semplici, come uno swing di tennis, o come nel lavoro di J. Koberet et al.[3] in cui, utilizzando l'apprendimento tramite DMP, si vuole comandare un braccio robotico al fine di farlo palleggiare con una racchetta ed una pallina da ping pong. O. C. Jenkins et al.[4] includono un simulatore piuttosto che un apparato robotico. In questo lavoro vengono valutati movimenti più ampi, che comprendono l'utilizzo di entrambe le braccia e del busto, elaborati dalle DMP e riprodotti da un programma di tipo *skeletor traking*, dove i movimenti di braccia e busto sono simulati tramite pochi giunti principali (come ad esempio gomiti e spalle). Rispetto a questi lavori citati il nostro si vuole differenziare soprattutto per la tipologia di dati forniti dall'esempio umano e passati poi alle DMP, in cui il movimento è registrato tramite guanto *CyberGlove2* formato da molti più giunti rispetto quelli che possono essere presenti in un robot manipolatore.

Figura 2.1: Robot Baxter che esegue movimento elaborato con le DMP

Nel caso di imitazione umana tornano utili tutti i parametri calcolati tramite DMP, ma esiste un'altra tipologia di utilizzo delle DMP: la classificazione di movimenti.
In questa casistica si sfruttano solamente i dati relativi al *forcing term* (i pesi delle funzioni base θ) utilizzati come *features* caratterizzanti del movimento stesso. Su questi vengono applicate *machine learning* per generare un modello predittivo. Con questo si potrà successivamente analizzare i dati di un movimento sconosciuto per sapere a quale assomiglia di più fra quelli di esempio classificati, per poi farlo replicare ad un robot, oppure studiare le *performances* della classificazione con le *features* considerate.

Ad esempio, nel lavoro di A. Sant'Anna et al.[5] si utilizzano i valori delle primitive di movimento assieme a modelli probabilistici Gaussian Mixture Model (GMM) per classificare diverse tipologie di andatura e trovare *pattern* comuni che le rappresentino, o come A. Fod et al.[6] in cui la classificazioni di movimenti di braccia, rappresentati tramite primitive di movimento, vengono classificati utilizzando Principal Component Analysis (PCA). Molte altre tecniche possono essere adoperate per la classificazione di movimenti, da quelle più semplici come *k-nearest neighbor*, a quelle più complicate, ad esempio GMM già citati, Reti Neurali o Support Vector Machine (SVM), come quelli utilizzati da M. H. Alomari et al.[7], anche se con movimenti caratterizzati da *features* differenti rispetto alle DMP.
Rispetto a questi lavori, il nostro si vuole distinguere per la tipologia di classificatori provati (Classification tree e modelli ECOC), che cerca di distaccarsi da quelli maggiormente utilizzati nell'ambito della classificazione di movimenti.

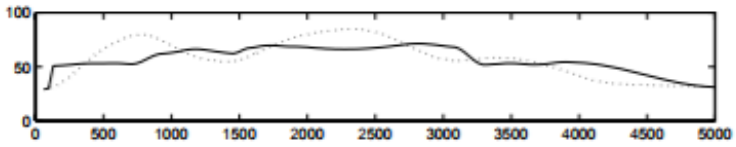

Figura 2.2: Ricostruzione del movimento di un giunto del braccio in [6]: la linea tratteggiata rappresenta il movimento ricostruito con le DMP in termini di variazioni d'angolo

Per l'imitazione di movimenti umani vengono spesso utilizzate informazioni prese da segnali di Electromyography (EMG), da elettrodi applicati alla persona che compie il movimento. Grazie alla loro analisi si riescono ad ottenere le informazioni di accelerazione e velocità per i giunti di un braccio robotico. I segnali EMG sono molto utilizzati in quanto facilmente recuperabili sia da *dataset* già presenti in ambito scientifico, che ottenuti *ex novo* tramite utilizzo di sensori facilmente reperibili in commercio.
Negli articoli di A. Gijsberts et al.[8], M. Pla-Mobarak et al. [9] e X. Zhai et al.[10] vengono utilizzati segnali sEMG provenienti dallo stesso *dataset* che impiegheremo in questa tesi, e classificati tramite PCA. In [8] e [9] l'insieme di movimenti utilizzati risulta limitato, mentre in questo elaborato cercheremo di utilizzare tutta la gamma di movimenti messa a disposizione.

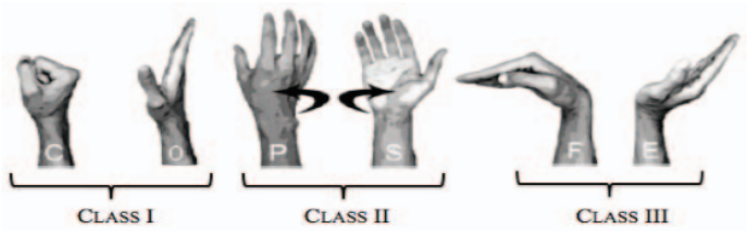

Figura 2.3: Movimenti classificati in [9]

Il lavoro svolto rientra nella casistica di classificazione e prende spunto, soprattutto per l'analisi dei movimenti presenti nel *dataset* NinaPro e l'utilizzo della classificazione tramite GMM di altri lavori sviluppati all'interno dell'Università di Padova, quali F. Stival et al.[11][12], R. Valentini et al.[13], S. Michieletto et al.[14] e R. Bortoletto et al.[15].

Come anticipato, il *dataset* utilizzato comprende informazioni su movimenti derivati sia da segnali EMG, che da segnali catturati da un guanto ricoperto di sensori *CyberGlove2*. Esistono pochi studi riguardanti l'utilizzo di questa tipologia di dati rispetto a quelli fatti utilizzando segnali EMG, per questo ci siamo focalizzati sulla creazione di un metodo efficace per la classificazione di movimenti della mano avendo un elevato numero di gradi di libertà dovuto al numero di sensori posti sul guanto *Cyberglove2*. L'utilizzo delle DMP per generalizzare movimenti eseguiti da più soggetti si è dimostrata molto efficace negli articoli citati. Per questo motivo è stata scelta, dato che nel *dataset* i movimenti da classificare sono stati compiuti da tipologie differenti di soggetti, quali maschi, femmine, mancini e destrorsi.
A differenza dei precedenti lavori, sono stati provati più metodi di classificazione con configurazioni diverse, al fine di trovarne uno con accuratezza soddisfacente, risolvendo le problematiche di volta in volta riscontrate a causa dell'elevato numero di *features* utilizzate per rappresentare i movimenti.

Capitolo 3

Dynamic Motion Primitives

3.1 Sistema dinamico

Un sistema dinamico è formato da uno stato, rappresentato come vettore di numeri, e da una regola che descriva come lo stato varia nel tempo. Un semplice esempio esplicativo potrebbe essere formato dallo stato x e dalla regola $\dot{x} = \frac{dx}{dt} = -\alpha x$. Quest'ultima serve anche per descrivere la relazione presente tra lo stato attuale $x(t)$ e quello successivo nel breve periodo $x(t + dt)$.

Se siamo a conoscenza del valore dello stato iniziale $x_0 = x(0)$ di un sistema dinamico, possiamo calcolare l'evoluzione del sistema nel tempo tramite l'integrazione numerica della regola $\dot{x}$ nel tempo infinitesimale dt. Questa procedura viene anche chiamata integrazione del sistema.
L'evoluzione di un sistema dinamico può essere determinata anche in maniera analitica, risolvendo l'equazione differenziale $\dot{x} = -\alpha x$, che come soluzione propone $S(t) = x_0 e^{-\alpha t}$

Se, entro un certo limite temporale, lo stato del sistema dinamico converge verso un determinato valore, questo viene definito "stato attrattore", e rappresentato nella regola dalla variabile x_g come $\dot{x} = -\alpha(x - x_g)$. Inoltre, un sistema dinamico possiede la proprietà di robustezza, in quanto anche se viene perturbato, convergerà comunque verso lo stato attrattore.
La velocità di cambiamento dello stato di un sistema può essere aumentata o diminuita tramite l'utilizzo di una costante temporale inserita nell'equazione differenziale della regola: $\dot{x} = -\alpha(x - x_g)/\tau$, con τ costante temporale.

Lo stato x non necessariamente è rappresentato da un unico valore, ma potrebbe essere un punto di un sistema n-dimensionale. In questo caso gli stati sarebbero rappresentati tramite un vettore di valori, e il sistema dinamico continuerebbe comunque ad esistere, cambiando solamente la rappresentazione degli stati: $\dot{\mathbf{X}} = -\alpha(\mathbf{X} - \mathbf{X}_g)/\tau$

3.1.1 Sistema dinamico di secondo ordine

L'ordine di un sistema dinamico equivale al grado più elevato fra tutte le derivate presenti nell'equazione differenziale. L'equazione $\dot{x} = -\alpha(x - x_g)/\tau$, ad esempio, appartiene ad un sistema di grado 1.
Un sistema *spring-damper*[16] , invece, rappresenta un sistema dinamico di grado 2, ed è caratterizzato dalle regole

$$m\ddot{x} = -kx - c\dot{x} \tag{3.1}$$

con k costante di *spring*, c coefficiente di *damp* e m massa. Se $c = 2\sqrt{mk}$ il sistema *spring-damper* converge verso il suo stato attrattore il più velocemente possibile senza *overshooting*, e viene definito *critical damping*.

3.2 Primitive di movimento

Ogni movimentazione di un robot richiede una forte coordinazione nei movimenti: ciò è fattibile tramite una accurata pianificazione, in quanto ogni grado di libertà posseduto dal robot deve essere supportato da un apposito comando durante tutto il movimento. Dato il loro numero, esistono infinite possibilità di configurazione dei gradi di libertà, al fine di far compiere al robot il movimento voluto. Questa grande libertà è uno svantaggio dal punto di vista della ricerca della migliore pianificazione, sopratutto se gli stati in cui un robot può trovarsi appartengono ad uno spazio di molteplici dimensioni.

Una metodologia di pianificazione della traiettoria che il robot deve eseguire per compiere il movimento è utilizzare le primitive di movimento. Per primitiva di movimento intendiamo un sistema con cui generare movimenti ritmici o discreti per ognuno dei gradi di libertà del robot. La maggior parte dei parametri necessari alla definizione di questi sistemi dinamici avviene attraverso apprendimento, fornendo al sistema una traiettoria di esempio simile a quella da far eseguire al

robot[3].

L'idea alla base delle *Dynamic Motion Primitives* (DMP)[17] è quella di rappresentare le primitive di movimento tramite una combinazione di sistemi dinamici (paragrafo 3.1), in cui lo stato rappresenta la traiettoria, compresa velocità e accelerazione, che i giunti del robot devono seguire per compiere il movimento voluto. Lo stato attrattore sarà formato dalla posizione finale che il robot deve assumere. Le DMP sono state esposte per la prima volta all'*International Conference on Robotics and Automation (ICRA)*[18] del 2002. Le formule utilizzate in questo elaborato seguono la struttura di quelle descritte in [19].

Il vantaggio principale portato dalle DMP è che grazie alle proprietà di un sistema dinamico lineare, aggiungendo alle regole un termine non lineare (*forcing term*), si riesce a rappresentare qualsiasi movimento si desideri. Oltretutto, il *forcing term* è facilmente determinabile tramite il metodo di apprendimento rinforzato dalla traiettoria di esempio fornita al sistema.

Un sistema molto utilizzato per il calcolo delle DMP è il *critical spring-damping* (equazione 3.1), definito nel seguente modo:

$$\tau\ddot{y} = \alpha(\beta(y_g - y) - \dot{y}) \tag{3.2}$$

dove con y indichiamo lo stato del sistema, $\alpha = c$, $\beta = \frac{k}{c}$ e $m = \tau$. Con y_g indichiamo invece lo stato finale del sistema (*goal*). Dato che utilizziamo un sistema *critical spring-damping*, in cui $c = 2\sqrt{mk}$, nella nuova equazione ciò si traduce con $\alpha = 2\sqrt{\alpha\beta}$, da cui si ricava che $\beta = \frac{\alpha}{4}$.
Il sistema, ora di grado 2, si può riscrivere con un'altra formulazione per portarlo al grado 1 e facilitarne la risoluzione:

$$\begin{bmatrix} \dot{z} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} (\alpha(\beta(y_g - y) - z))/\tau \\ z/\tau \end{bmatrix} \tag{3.3}$$

ponendo $\begin{bmatrix} 0 \\ y_0 \end{bmatrix}$ come stato iniziale e $\begin{bmatrix} 0 \\ y_g \end{bmatrix}$ come *goal*.

3.2.1 Forcing term

Il sistema definito dall'equazione (3.3) possiede ancora le proprietà dei sistemi dinamici, quali convergenza verso lo stato attrattore e la robustezza alle perturbazioni,

tuttavia può essere usato soltanto per rappresentare movimenti molto semplici, in quanto è ancora sprovvisto del *forcing term*. Per rendere possibile la modellazione di movimenti più complessi, occorre aggiungerlo al sistema, che diventa:

$$\begin{bmatrix} \dot{z} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} (\alpha(\beta(y_g - y) - z) + f(t))/\tau \\ z/\tau \end{bmatrix} \tag{3.4}$$

avendo $\begin{bmatrix} 0 \\ y_0 \end{bmatrix}$ come stato iniziale e $\begin{bmatrix} ? \\ y_g \end{bmatrix}$ come *goal*.
Come si può vedere, il *forcing term* $f(t)$ dipende solamente dal tempo, ed è tipicamente costituito da una funzione di approssimazione, come la locally weighted regression (LWR) o locally weighted projection regression (LWPR)[20][21], utilizzata in questo elaborato.

Entrambe le tecniche citate servono ad approssimare funzioni non lineari tramite l'utilizzo di una serie di modelli di regressione. Nel nostro caso, la funzione $f(t)$ da calcolare sarà quella che, combinata alla prima parte del sistema dinamico, riuscirà a replicare la traiettoria di esempio fornita, e costruita tramite l'utilizzo di n_b funzioni gaussiane la cui influenza sarà pesata tramite un valore θ_k, con $k = 1, ..., n_b$.

Considerando che con un sistema dinamico *critical spring-damping* si riuscirebbero a riprodurre soltanto movimenti semplici, la maggior influenza nella pianificazione di una traiettoria è data dall'andamento del *forcing term*. Questo viene definito dai pesi dati agli n_b modelli di regressione utilizzati. Per due traiettorie simili si avranno dei valori θ_k rispettivamente simili tra loro, e ciò ci ha indotto ad utilizzare i θ stessi come *features* rappresentative di una traiettoria.

Avendo aggiunto il forcing term al sistema *critical spring-damping* non siamo sicuri che questo continui a convergere verso lo stato attrattore y_g: per questo motivo è stato posto ? nella formulazione del *goal* nell'equazione 3.4.
Per avere garanzie sulla convergenza del sistema bisogna che il *forcing term* decresca fino ad arrivare a 0 alla fine del movimento. Al fine di rispettare questa regola, viene aggiunta al sistema una funzione di *gating*, che vale 1 ad inizio movimento e 0 alla fine. Il *gating* viene determinato tramite un sistema dinamico, solitamente

esponenziale:

$$\begin{bmatrix} \dot{z} \\ \dot{y} \\ \dot{x} \end{bmatrix} = \begin{bmatrix} (\alpha_y(\beta_y(y_g - y) - z) + xf(t))/\tau \\ z/\tau \\ -\alpha_x x/\tau \end{bmatrix} \tag{3.5}$$

ponendo $\begin{bmatrix} 0 \\ y_0 \\ 1 \end{bmatrix}$ come stato iniziale e $\begin{bmatrix} 0 \\ y_g \\ 0 \end{bmatrix}$ come *goal*.
Con calcolo delle DMP intenderemo, da qui in avanti, la risoluzione del sistema dinamico ottenuto per rappresentare una traiettoria di esempio fornita. In particolar modo ci riferiremo al calcolo dei valori θ ottenuti per il *forcing term*, in quanto, come abbiamo già detto in precedenza, viene individuato come termine più utile al confronto di traiettorie.

Il primo programma Matlab trovato per calcolare le DMP testato è stato quello degli ideatori di questa tecnica, A. J. Ijspeert et al.[18], disponibile *opensource* direttamente dal sito della *University of Southern California*[1].
Questo pacchetto non è più supportato, in quanto è stato migrato verso una nuova versione scritta in C++.

Un secondo pacchetto di codice Matlab è stato trovato nel sito per il *code sharing Github*[2]. In questo caso, il codice realizzato per il calcolo delle DMP risulta supportato fino al 2015 e comprende molte funzionalità aggiuntive, tra cui una forte parte di ottimizzazione per velocizzare i calcoli delle DMP[22].
Per il maggior supporto che ha ricevuto, e per le ottimizzazioni nel codice e nel calcolo delle DMP è stato scelto di lavorare con questo pacchetto per il calcolo delle DMP dai dati provenienti dal *dataset* utilizzato e descritto al capitolo 5.

Questo secondo pacchetto mette a disposizione 3 diverse tipologie di sistema dinamico per il calcolo delle DMP, identificati da 3 ordini differenti:

- il sistema indicato con ordine 1, detto anche sistema canonico, è formato dalla regola

$$\tau\dot{y} = -\alpha_y y \tag{3.6}$$

[1] `http://www-clmc.usc.edu/Resources/Software`
[2] `https://github.com/stulp/dmp_bbo_matlab_deprecated/tree/master_deprecated`

con $\alpha = -log(0,001)$. Questo risulta essere il più semplice fra i 3 sistemi messi a disposizione;

- il sistema indicato da ordine 2 utilizza invece un sistema *critical spring-damper* come quello dell'equazione 3.3, i cui parametri somo impostati come segue: $\alpha = 15$, $\beta = \frac{\alpha}{4}$, $m = 1$ e $z = \frac{\alpha}{2\sqrt{\alpha\beta}}$;

- il sistema indicato con ordine 3 utilizza invece un sistema sigmoidale[23], nel quale l'equazione differenziale è

$$\dot{y} = (1 + e^{[\alpha_v(\tau T - t)]^2})^{-1} \alpha_y e^{\alpha_v(\tau T - t)} \tag{3.7}$$

con $\alpha_v = \frac{log(100-1)}{0.5T}$ e T pari ad $\frac{1}{4}$ del tempo di durata della traiettoria di esempio.

Ad ognuno di questi sistemi dinamici è stato aggiunto un *forcing term* $f(t)$ di tipo LWPR, dunque utilizzano k funzioni gaussiane. I pesi di queste funzioni vengono restituiti al termine del calcolo delle DMP tramite ognuno dei 3 sistemi elencati e vengono indicati con θ.

3.3 Somiglianza tra segnali

3.3.1 Cross Covariance

La covarianza di due variabili X e Y indica la loro tendenza a variare nello stesso momento o periodo di tempo. Se cov(X,Y) è positiva, significa che quando X assume valori elevati, anche Y assume valori elevati o viceversa. Se invece la covarianza risulta negativa, si propone una situazione inversa a quelle appena descritte, ovvero per piccoli valori della X sono associati valori elevati per la Y o viceversa.

Nel corso della tesi è stata usata una funziona strettamente collegata alla covarianza: la covarianza incrociata o Cross Covariance ($xcov$), la quale misura la somiglianza tra la variabile X e la copia traslata nel tempo della variabile Y. Entrambe le variabili sono rappresentate in funzione del tempo.

Nella teoria dei segnali la Cross Covariance viene utilizzata come misura per calcolare la somiglianza tra due segnali, ed i risultati vengono impiegati soprattutto per:

- riconoscimento di particolari feature in un segnale ignoto;
- problemi di pattern recognition;
- analisi crittografica.

In Matlab, la Cross Covariance viene calcolata con la funzione *xcov(X, Y)*, dove X e Y sono due sequenze tempo-discrete.
Se X e Y hanno lunghezze diverse, la funzione provvede ad allungare la sequenza più corta aggiungendo degli zero, affinchè i due input della funzione abbiano lunghezza uguale.
xcov non esegue nessun controllo su eventuali errori, se non quello sul numero delle varibili in ingresso, delegando questa incombenza alla funzione *xcorr* chiamata a fine calcolo.

Se le variabili in input alla funzione *xcov* sono due processi aleatori stazionari x_n e y_n, la Cross Covariance corrisponde alla Cross Correlation fra i due processi dopo che viene sottratta loro la media:

$$\phi_{x,y}(m) = E\big\{(x_{n+m} - \mu_x)((y_n - \mu_y)^*)\big\} \tag{3.8}$$

dove μ_x e μ_y sono le medie dei due processi aleatori. L'asterisco denota l'operazione di complesso coniugato ed E rappresenta l'operatore di valore atteso.

xcov può unicamente stimare la sequenza risultante dato che è disponibile solo parte di una sola delle possibli realizzazioni dell'intero processo aleatorio a lunghezza infinita [33][34].

Di default, *xcov* calcola la Cross Covariance senza normalizzazione:

$$c_{xy}(m) = \begin{cases} \displaystyle\sum_{n=0}^{N-m-1} \left(x_{n+m} - \frac{1}{N}\sum_{i=0}^{N-1} x_i\right)\left(y_n^* - \frac{1}{N}\sum_{i=0}^{N-1} y_i^*\right) & \text{se } m \geq 0 \\ c_{yx}^*(-m) & \text{se } m < 0 \end{cases} \tag{3.9}$$

Il vettore c di output ha elementi dati da:

$$c(m) = c_{xy}(m - N), \quad m = 1, ..., 2N - 1 \tag{3.10}$$

La funzione di Cross Covariance richiederebbe una normalizzazione per poter eseguire una stima corretta. Tale normalizzazione è controllabile utilizzando il parametro opzionale *scaleopt* della funzione *xcov*.

3.3.2 Cross Correlation

La covarianza di due variabili X e Y dipende molto dalla scala dei valori coinvolti. Per questo motivo spesso viene normalizzata tramite divisione per $\sigma_x \cdot \sigma_y$, ottenendo così la misura di correlazione.
La correlazione ha i vantaggi di essere priva di unità di misura e di variare fra [-1; 1]. Ciò la rende una misura facilmente gestibile e descrive la relazione lineare tra due variabili.

Nel corso del lavoro svolto è stata utilizzata una misura derivante dalla correlazione: la Cross Correlation (*xcorr*).
Tale misura calcola la correlazione fra una variabile X e la copia della variabile Y traslata in funzione del tempo. Spesso la Cross Correlation viene utilizzata assieme alla Cross Covariance e applicata nella stessa gamma di problematiche.

In Matlab la Cross Correlation viene calcolata dalla funzione *xcorr(X, Y)*, con X e Y sequenze tempo-discrete. Anche in questo caso, se le due sequenze presentano lunghezze differenti, a quella più corta vengono aggiunti alla fine un numero sufficiente di zeri per riportare le due lunghezze alla pari.
La Cross Correlation di due processi aleatori stazionari x_n e y_n viene calcolata da:

$$R_{xy}(m) = E\{x_{n+m}y_n^*\} = E\{x_n y_{n-m}^*\} \tag{3.11}$$

con $-\infty < n < \infty$.
La funzione *xcorr* di Matlab può soltanto stimare la Cross Correlation, perchè tramite le variabili di ingresso viene fornita soltanto una parte finita di una possible realizzazione del processo aleatorio infinito[35][36].

Di default, *xcorr* calcola la correlazione tra due variabili senza normalizzazione:

$$\hat{R}_{xy}(m) = \begin{cases} \sum_{n=0}^{N-m-1} x_{n+m} y_n^* & \text{se } m \geq 0 \\ \\ \hat{R}_{xy}^*(-m) & \text{se } m < 0 \end{cases} \tag{3.12}$$

Il vettore c di output ha elementi dati da:

$$c(m) = \hat{R}_{xy}(m - N), \quad m = 1, ..., 2N - 1 \tag{3.13}$$

Generalmente, la correlazione richiede una normalizzazione per poter restituire risutati accurati. Tale normalizzazione può essere controllata tramite l'utilizzo del parametro *scaleopt* della funzione *xcorr*.

Capitolo 4

Tecniche di Classificazione

4.1 Classification Tree

Un Classification tree (albero di classificazione) è un modello predittivo nel quale, dai dati di input, seguendo un percorso ad albero si riesce ad ottenere (predire) una etichetta di classificazione (classe).
In ingresso, un Classification tree richiede un record del tipo

$$r = [x_1, x_2, ..., x_n] \tag{4.1}$$

nel quale sono valorizzate n variabili differenti. Seguendo il percorso selezionato dall'albero, partendo dal nodo radice fino ad arrivare in un nodo foglia, si ottiene la classificazione del record[37].

All'interno di ciascun nodo è presente un test, che prende in considerazione una delle n variabili dell'ingresso e controlla se il valore è maggiore o minore rispetto ad una determinata soglia, proseguendo di conseguenza per il ramo destro o sinistro del nodo. Una volta giunti ad una foglia, essa conterrà l'etichetta con cui classificare il record in input.

4.1.1 Costruzione

Un Classification tree si costruisce utilizzando un insieme di record, chiamati anche esperimenti od osservazioni, del quale si conosce la classe di appartenenza. Questo insieme viene suddiviso in due parti:

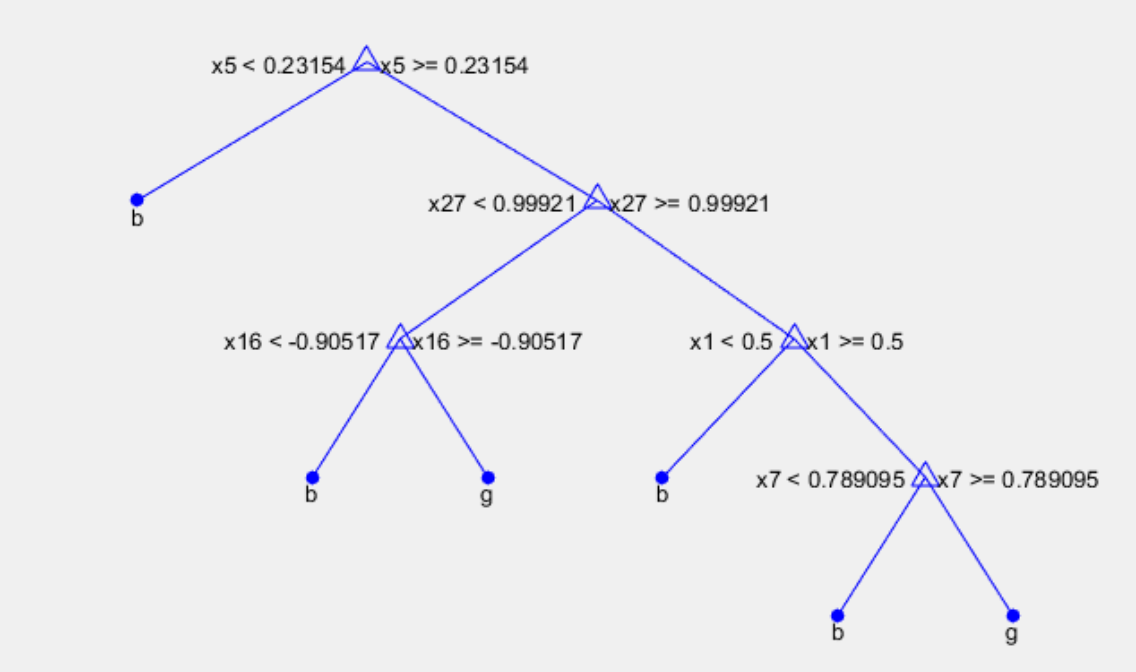

Figura 4.1: Esempio di un semplice Classification tree

- il *training set*, sulla base del quale si crea la struttura dell'albero;
- il *test set*, che verrà utilizzato per testare l'accuratezza del modello predittivo

Una volta costruito l'intero albero grazie al *training set*, è utile passarlo per un criterio di potatura (*pruning*) al fine di limitarne la profondità massima. Questo perchè il crescere della profondità di un albero (ovvero della sua dimensione) non influisce positivamente la bontà del modello.
Infatti, una crescita eccessiva della dimensione dell'albero potrebbe portare un aumento sproporzionato della complessità computazionale portando limitati benefici riguardo l'accuratezza delle previsioni.

In Matlab la funzione per creare un Classification tree viene chiamata *fitctree(TBL, LBL)*, il cui ingresso TBL è una matrice $m \times n$ contenente gli m record del *training set* disposti in riga, mentre LBL è un vettore colonna lungo m contenente la classificazione appartenente ad ognuno dei record.

Per costruire la struttura dell'albero viene utilizzato un algoritmo ciclico. Durante ciascuna iterazione viene trovata la miglior variabile x_i della quale testare il

valore e creato un nodo con tale test. Questo nodo dividerà il *training set* in due parti, a seconda del valore di soglia della variabile x_i di ogni record e su queste porzioni viene eseguito nuovamente l'algoritmo[38][39].

Per scegliere quale test eseguire su di un nodo t, *fitctree* segue questi passi:

- calcola impurità i_t dell'insieme di record che il nodo t dovrà suddividere. L'impurità di un nodo viene calcolata grazie al Gini's Diversity Index (gdi) del nodo:
$$gdi(t) = 1 - \sum_i p^2(i) \tag{4.2}$$
dove la sommatoria viene fatta per ogni classe i presente nei record associati al nodo, e $p(i)$ è la frazione dei record del nodo a cui è associata la classe i. Se ad un nodo sono associati record con una sola classe, tale nodo viene definito puro e il suo gdi risulta pari a 0, altrimenti il gdi risulta maggiore di 0;

- stima la probabilità che un record appartenga all'insieme del nodo t tramite
$$P(T) = \sum_{j \in T} w_j \tag{4.3}$$
dove w_j è il peso associato al record j e T è l'insieme di tutti i record associati al nodo T. Senza informazioni specifiche per pesare i record $w_j = 1/m$, con m indicante il numero di osservazioni presenti nel *training set*;

- ordina le variabili x_i, prese da ogni record del nodo, in ordine crescente. Ognuna delle variabili ordinate viene candidata come variabile sulla quale potrà essere eseguito il test del nodo.
Se ad un record una o più variabili non fossere valorizzate, tale record verrebbe inserito nell'*unsplit set* T_U;

- *fitctree* determina la miglior suddivisione dei record del nodo t utilizzando tutte le possibili scelte tra le variabili x_i ordinate. Questa scelta viene presa confrontando gli *impurity gain* (ΔI) generati da ogni variabile candidata, che vengono calcolati come segue:
  - viene spezzato in due parti il set di record di t secondo la soglia della variabile x_i considerata, una per ogni nodo figlio di t. La parte associata al nodo sinistro viene chiamata T_L, quella al nodo destro T_R;

– viene calcolato l'*impurity gain* di questa suddivisione. Chiamando t_L i record associati al nodo figlio sinistro e t_R i record associati al nodo figlio destro:
  * se ogni record associato al nodo t ha valorizzata la variabile x_i

$$\Delta I = P(T)i_t - P(T_L)i_{t_L} - P(T_R)i_{t_R} \tag{4.4}$$

  * se per qualche record del nodo t la variabile x_i non è valorizzata

$$\Delta I_U = P(T - T_U)i_t - P(T_L)i_{t_L} - P(T_R)i_{t_R} \tag{4.5}$$

    con $T - T_U$ indicante l'insieme dei soli record di t in cui x_i è valorizzata
– viene scelta come variabile per il test sul nodo t (e rispettiva soglia) quella che ha generato il maggior valore di *impurity gain*.

I passaggi vengono successivamente ripetuti per il nodo figlio sinistro e destro di t.

L'algoritmo si ferma, non suddividendo più il set di record solamente quando il nodo da trattare risulta puro. In alternativa, si può forzare l'algoritmo a fermarsi quando il numero di record che verrebbe associato ad un nodo figlio è minore del parametro *MinLeafSize* specificato alla funzione *fitctree*. In questo caso, il nodo t viene considerato foglia e la classe ad esso associata è quella a cui appartiene il maggior numero di record associati al nodo.

Quando si cerca il test da inserire in t, l'algoritmo seleziona il migliore tra tutti gli $m \times n$ possibili. In alternativa, per velocizzare i tempi si può scegliere il test utilizzando un algoritmo euristico per trovare un buon test, anche se non il migliore.

La fase di *pruning* di un Classification tree viene eseguita sfruttando la funzione *prune(tree, l)*, in cui *tree* rappresenta un albero di classificazione e l il livello di *pruning* a cui viene sottoposto.
Questa funzione rimuove dall'albero tutti i nodi presenti negli ultimi l livelli di profondità dell'albero: con $l = 1$ vengono rimossi i nodi all'ultimo livello, con $l = 2$ vengono rimossi tutti i nodi presenti all'ultimo e penultimo livello, e così di seguito.
Quando un nodo viene rimosso nella fase di *pruning*, i record associati ad esso

vengono aggiunti al set del nodo genitore. Se un nodo interno t diventa foglia, successivamente alla fase di *pruning* la classe associata a t diventa quella presente in maggioranza sul set di record, dopo averci aggiunto anche quelli del nodo figlio eliminato.

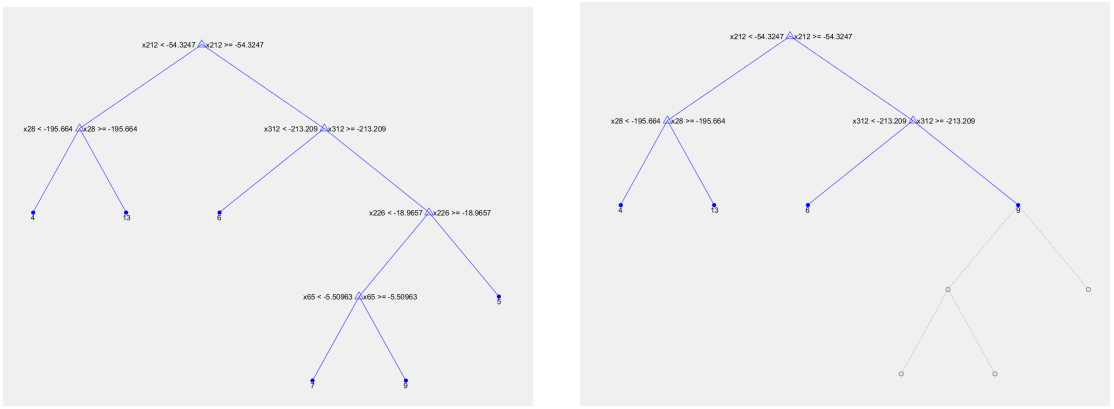

Figura 4.2: Esempio di *pruning* livello 2 per un Classification tree

4.1.2 Predizione

Un Classification tree viene usato per avere una classe da associare ad un record r. Anche questo record deve essere della stessa tipologia di quelli utilizzati per la costruzione dell'albero.

Ottenere la classe per un record risulta semplice:

- si parte dal nodo radice dell'albero eseguendone il test associato alla variabile x_i e, se $x_i \in r$ è minore del valore di soglia si scende lungo il ramo sinistro fino al nodo figlio sinistro, altrimenti si procede verso il nodo figlio destro;
- se il nodo figlio è un nodo interno, proporrà un altro test per un'altra variabile x_j e si procederà nello stesso modo descritto al punto precedente;
- se il nodo figlio è un nodo foglia, conterrà la classe c da associare al record r.

La classe c si dice predetta per r dal classificatore, che in questo caso è un Classification tree.

Di un Classification tree, o di un classificatore in generale, è fondamentale conoscere l'accuratezza (o precisione) delle previsioni. Questa quantità indica il livello di fiducia per una previsione del classificatore sulla classe per un record r. La precisione di un classificatore viene calcolata utilizzando il *test set* T_s, nel quale per ogni record r_s si conosce la classe c_s:

- viene creato un contatore *counter* inizializzato a zero;
- $\forall r \in T_s$ viene predetta la classe c dal classificatore;
- se $c = c_s \Rightarrow counter = counter + 1$.

La precisione viene calcolata da $counter/|T_s|$, ottenendo un valore nell'intervallo [0; 1].

Matlab dispone della funzione *predict(tree, X)* per la previsione di classi, dove *tree* rappresenta il Classification tree da utilizzare, mentre nella matrice X sono presenti, posti per riga, gli m record per i quali vogliamo ottenere la classificazione. Come output viene proposto un vettore colonna di m righe, ciascuna contenente la classe predetta per il rispettivo record di X.

4.2 Gaussian Mixture Model

Il modello probabilistico GMM permette di ottenere la somma pesata di K componenti gaussiane che verranno utilizzate per predire quale movimento è rappresentato dai dati in ingresso, riuscendo così a classificarlo fra un insieme limitato di possibili classi. Il *dataset* usato per allenare il modello GMM, chiamato anche qui *training set*, sarà composto da N record di esempio:

$$\zeta_j = \{\Theta(t), C(t)\} \in \mathbb{R}^D \qquad j = 1, ..., N \tag{4.6}$$

in cui $\Theta(t) = \{\theta_1(t), \theta_2(t), ..., \theta_G(t)\}$ rappresenta l'andamento di G funzioni nel tempo, e C un vettore colonna contenente la classe associata a $\Theta(t)$ per ogni istante t. Da questo di può notare quindi che $D = G + 1$.
Inoltre tutte le funzioni θ saranno a valori reali, cioè $\theta_i(t) \in \mathbb{R}, \forall i \in \{1, ..., G\}$.
L'output del modello sarà una mistura di K variabili aleatorie gaussiane multivariate, definite su D dimensioni.

Il modello GMM è costruito sfruttando all'algoritmo Expectation Maximization (EM) [41], che permette di ottenere una distribuzione di probabilità partendo dal *training set* usata successivamente sul *test set* per calcolare la precisione della distribuzione e delle sue previsioni. Gli M record presenti nel *test set* sono strutturati analogamenete a quelli del *training set*, a cui però è stato rimosso il vettore C con le classificazioni:

$$\lambda_j = \{\Theta(t)\} \in \mathbb{R}^G \qquad j = 1, ..., M \tag{4.7}$$

L'algoritmo EM stima iterativamente i parametri ottimali $\upsilon_i = (\mu_i, \Sigma_i), i \in \{1, ..., K\}$ che caratterizzeranno le K variabili aleatorie gaussiane ottenute per il modello GMM, dove μ_i e Σ_i rappresentano il vettore media e la matrice di covarianza per la *i-esima* variabile aleatoria gaussiana multivariata in D dimensioni.

L'algoritmo è diviso in due fasi racchiuse in un unico ciclo: la fase *Expectation* e la fase *Maximization*. Il ciclo viene fermato solamente quando l'incremento della probabilità logaritmica (*log-likelihood*) $\mathcal{L} = \Sigma_{j=1}^{N} log(p(\zeta_j|\upsilon))$, che diminuisce ad ogni ciclo, supera la soglia rappresentata dalla disuguaglianza $\frac{\mathcal{L}(t+1)}{\mathcal{L}(t)} < \epsilon$.
Ognuna delle K componenti gaussiane è definita dai parametri (π_k, μ_k, Σ_k) definiti come segue:

$$\mu_k = \{\mu_{p,k}\mu_{c,k}\} \qquad \Sigma_k = \begin{bmatrix} \Sigma_{p,k} & \Sigma_{pc,k} \\ \Sigma_{cp,k} & \Sigma_{c,k} \end{bmatrix} \tag{4.8}$$

dove μ_p e Σ_p rappresentano la media e la matrice di covarianza dell'informazione nota a priori e descritta dal *training set*. L'etichetta di classificazione e la rispettiva covarianza viene stimata tramite le equazioni:

$$\hat{c} = E[c|t, \xi] = \sum_{k=1}^{K} \beta_k \hat{c}_k \tag{4.9}$$

$$\hat{\Sigma}_s = Cov[c|t, \xi] = \sum_{k=1}^{K} \beta_k^2 \hat{\Sigma}_{c,k} \tag{4.10}$$

dove:

- $\beta_k = \frac{\pi_k \mathcal{N}(t, \xi_c|\mu_{p,k}, \Sigma_{p,k})}{\Sigma_{j=1}^{K} \mathcal{N}(t, \xi_c|\mu_{p,j}, \Sigma_{p,j})}$ è il peso assegnato alla componente k della mistura gaussiana;

- $\hat{c}_k = E[c_k|t,\xi] = \mu_{c,k} + \Sigma_{cp,k}(\Sigma_{p,k})^{-1}(\{t,\xi\} - \mu_{p,k})$ è l'aspettazione di c_k condizionata a $\{t,\xi\}$;

- $\hat{\Sigma}_{c,k} = Cov[c_k|t,\xi] = \Sigma_{c,k} + \Sigma_{cp,k}(\Sigma_{p,k})^{-1}\Sigma_{pc,k}$ è la covarianza di c_k sempre condizionata a $\{t,\xi\}$.

L'algoritmo EM calcola così tutti gli elementi necessari alla creazione delle K gaussiane che formeranno il modello, e come dati di ingresso richiede solamente il *training set* e il numero K di componenti gaussiane che si vogliono calcolare.

Per la creazione di modelli GMM sono stati utilizzati 3 diversi programmi: il primo è stato sviluppato internamente all'Università di Padova durante lo svolgimento di una tesi di Laurea, e scritto interamente in linguaggio C++. Il suo funzionamento richiede che il *training set* sia fornito tramite file di testo.
Il secondo programma, scritto anch'esso in codice C++, è stato reperito dal sito di *code sharing* GitHub[1]. Principalmente verrà utilizzato per la verifica dei risultati ottenuti con il codice del primo pacchetto, per evitare che da un *training set* buono non si riescano ad ottenere altrettanto buoni risultati per problemi dovuti ad errori presenti nel codice. Sono state necessarie alcune modifiche al codice C++ per far accettare come file di input gli stessi realizzati ed utilizzati nel primo programma. Infine, per ulteriore conferma, verrà utiizzata anche la funzione *fitgmdist* di Matlab per la costruzione di modelli GMM. Anche se più lenta rispetto ai due codici precedenti in C++, la versione Matlab riesce ad avere una maggiore precisione per i valori dei risultati ottenuti, ed è facilmente implementabile la lettura del *training set* da file di input.

4.3 Error-Correcting Output Codes Model

Error-Correcting Output Codes (ECOC) è un modello di classificazione costruito appositamente per la risoluzione di problemi di classificazione multiclasse, in cui ci siano $k > 2$ possibili classi tra cui scegliere. La risoluzione del problema multiclasse avviene tramite la combinazione di più classificatori binari, chiamati *learners*.

Un modello ECOC, per essere creato e poi utilizzato, necessita di alcune informazioni:

[1] `https://github.com/chaohe1024/GMM`

- un insieme di record del tipo $r = [x_1, x_2, ..., x_n, c]$, chiamato *train set*, in cui ogni variabile x_i è valorizzata e dove c indica la classe associata al record r, con i quali verranno allenati i *learners*;
- la tipologia di classificatori binari da utilizzare come *learners*;
- un *coding design*, per determinare quali classi ogni *learner* deve suddividere;
- un *decoding scheme*, per interpretare i risultati forniti dai *learners* e riuscire a classificare un nuovo record.

Il *coding design* è costituito da una matrice in cui gli elementi indicano come sarà valutata una classe per ognuno dei *learner*. In questo modo, un problema di classificaione con molteplici classi viene ridotto a più problemi di classificazione binaria.
Ogni riga del *coding design* rappresenta una differente classe, mentre ogni colonna indica un singolo *learner*. Gli elementi della matrice possono assumenre uno dei seguenti 3 valori:

- 1, per indicare al *learner* che tutti i record di questa classe appartengono alla classe positiva;
- -1, per indicare al *learner* che tutti i record di questa classe appartengono alla classe negativa;
- 0, per far ignorare al *learner* tutti i record di questa classe.

Se il numero k di classi è relativamente piccolo, vengono provate tutte le possibili combinazioni di una o più classi raggruppate come positive. In caso di k molto elevati, vengono create matrici per il *coding design* tramite algoritmi casuali[27]

All'interno del lavoro svolto, sono state provate due tipologie di *coding design*:

- *one-versus-all*, in cui per ogni classificatore binario una classe viene considerata positiva, mentre tutte le altre vengono considerate negative. Con questa strategia vengono creati k *learners*, in ognuno dei quali la classe positiva è sempre differente dagli altri, fino a coprirle tutte;
- *one-versus-one*, in cui per ogni classificatore binario una classe viene considerata positiva, una negativa, e le restanti ignorate. In questo caso, vengono creati $k(k-1)/2$ *learners*, per considerare tutte le possibili coppie di classi.

Il *decoding scheme* consiste in una funzone utilizzata per capire quale classe assegnare ad un nuovo record, e quanto correttamente questa viene assegnata dai vari classificatori binari.
Nel nostro caso è stata utilizzata come *decoding scheme* il *loss-weighted decoding*[28], in cui, grazie all'utilizzo della funzione di *Binary loss* di default impostata da Matlab per i modelli ECOC, viene calcolata la classe $\hat{k}$ da assegnare ad un nuovo record. Il suo funzionamento è definito da:

$$\hat{k} = \underset{x}{argmin} \frac{\sum_{j=1}^{L} |m_{kj}| g(m_{kj}, s_j)}{\sum_{j=1}^{L} |m_{kj}|} \tag{4.11}$$

in cui m_{kj} è l'elemento della matrice di *coding design* M per la classe k ed il *learner* j, s_j è il punteggio che il classificatore j assegna al nuovo record riguardante la sua classificazione nella classe positiva, e g è la funzione di *binary loss*, ed L indica il numero di classificatori utilizzati.

Un esempio di modello ECOC potrebbe essere quello ottenuto per un problema con 3 classi, scegliendo come *coding design* una strategia *one-versus-one*, come *decoding scheme* una strategia *loss-weighted decoding* con funzione di *Binary loss* indicata da g, e come tipologia di classificatori binari le SVM.
La prima fase di creazione del modello consiste nell'allenare tutti i *learners* in base allo schema imposto dal *coding design*, che possiamo vedere in tabella: In questo

	Learner 1	*Learner* 2	*Learner* 3
Class 1	1	1	0
Class 2	-1	0	1
Class 3	0	-1	-1

Tabella 4.1: Esempio di *coding design* con strategia *one-versus-one*

caso, il *learner* 1 crea una SVM per classificare tutti i record delle classi 1 e 2, utilizzando la classe 1 come positiva e la 2 come negativa ed ignorando tutti i record relativi alla classe 3. Il *learner* 2, invece, utilizzerà i record della classe 1 come positivi, i record della classe 3 come negativi e quelli della classe 2 saranno scartati, e così via per i restanti *learner*.
La seconda fase consiste nel'utilizzo del modello ECOC per la classificazione di un nuovo record, calcolata tramite la funzione definita dal *decoding scheme loss-weighted* con l'equazione 4.11. La classe $\hat{k}$ sarà quella assegnata al nuovo record

dal modello di classificazione ECOC.

In Matlab la costruzione di un modello ECOC viene eseguita tramite la funzione *fitcecoc*, la quale in ingresso necessita di avere la matrice X contenente i record del *train set*, un vettore colonna Y nel quale sono indicate le classi associate ad ogni record presente nella matrice X, la tipologia di *learners* da utilizzare, selezionata da quelli descritti nella sezione successiva e la tipologia di *coding design* per la classificazione binaria che tutti i *learners* eseguiranno. Il resto dei parametri necessari al corretto funzionamento è impostato di default da Matlab.
Una volta costruito il modello ECOC, è possibile utilizzarlo per la classificazione di nuovi record tramite la funzione *predict*, alla quale viene passata una matrice Z contenente i nuovi record d classificare. In uscita viene fornito un vettore colonna contenente la classe calcolata dal modello per ogni nuovo record presente nella matrice Z.

4.3.1 Tipologie di Learners

Fra le tipologie di classificatori binari disponibili, per i modelli ECOC testati sono stati utilizzati tutti quelli di seguito elencati:

- classificatore Lineare;
- *k-Nearest Neighbors* (k-NN);
- classificatore *Naive Bayes*;
- Support Vector Machine (SVM);
- Analisi Discriminante.

Un classificatore Lineare cerca di suddividere i dati del *train set* tramite una funzione del tipo $f(X) = \beta' X + b$, dove β è un vettore colonna contenente n coefficienti, X contiene i record del *train set* formati tutti da n variabili, e b indica un termine scalare. La funzione $f(X)$ viene calcolata utilizzando tecniche apposite per ridurre il tmpo di calcolo dovuto all'elevato numero di dimensioni presente nei record del *train set*, come ad esempio la *stochastic gradient descent*[29]

Un classificatore k-Nearest Neighbors (k-NN) è un classificatore nel quale i record del *train set* vengono convertiti in punti di uno spazio S a n dimensioni. Come parametro di configurazione richiede un valore intero positivo k, che indica

quanti punti del *train set* verranno coinvolti per la classificazione di nuovi record. Non richiede invece una particolare fase di allenamento di un modello, in quanto è sufficiente il posizionamento dei record nello spazio S per poter procedere poi alla classificazione di nuovi record. Per questo motivo k-NN appartiene alla tipologia *lazy learning* di classificatori.
La classificazione di un nuovo record avviene tramite i seguenti passi:

- posizionamento del nuovo record nello spazio S;
- ricerca dei k punti del *train set* più vicini al nuovo punto, tramite il calcolo della distanza euclidea da esso;
- confronto delle classi assegnate ai k punti vicini: il nuovo record sarà classificato con la classe presente in maggioranza nei punti vicini.

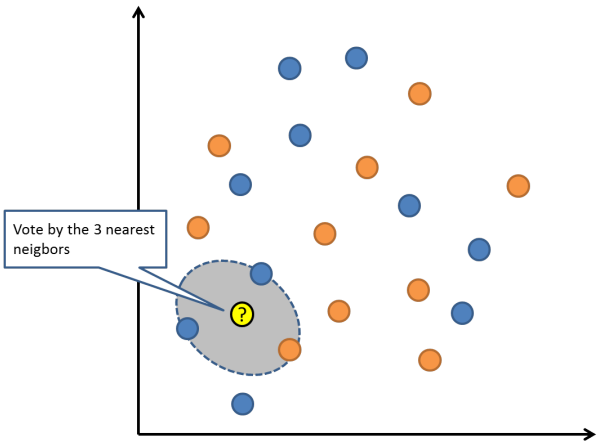

Figura 4.3: Esempio di classificazione di un nuovo punto con k=3 e n=2. In questo caso, il nuovo punto viene classificato come blu

Il classificatore k-NN è uno dei più semplici metodi di *machine learning* fra tutti quelli ad ora esistenti.

Un classificatore *Naive Bayes* è un algoritmo di classificazione che applica la stima di densità di probabilità ai dati del *train set* che gli vengono forniti. L'algoritmo si basa sul teorema di Bayes e sul fatto che le previsioni per nuovi record siano tra loro condizionalmente indipendenti, date le classi. Considerando che questa condizione è spesso violata nella realtà, un classificatore *Naive Bayes* per

prima prova a stimare la distribuzione di probabilità a posteriori, molto più robusta rispetto alla stima di densità di probabilità a valori scalari[30].
I classificatori *Naive Bayes* assegnano ad ogni nuovo record la classe più probabile (quella che massimizza la probabilità a posteriori). L'algoritmo di classificazione funziona secondo queste fasi:

- stima della densità di probabilità per ognuna delle classi;
- modellazione della probabiltà a posteriori secondo il teorema di Bayes. Per ogni classe $k = 1, ..., K$

$$\hat{P}(Y = k|X_1, ..., X_n) = \frac{\pi(Y = k) \prod_{j=1}^{n} P(X_j|Y = k)}{\sum_{k=1}^{K} \pi(Y = k) \prod_{j=1}^{n} P(X_j|Y = k)} \tag{4.12}$$

  dove Y è la variabile casuale corrispondente all'indice della classe del nuovo record, $X_1, ..., X_n$ sono le variabili del record, e $\pi(Y = k)$ è la probabilità a priori che l'indice della classe del record sia k;
- stima della probabilità a posteriori per ognuna delle classi, e classificazione del nuovo record con la classe k che massimizza la probabilità a posteriori.

Un classificatore SVM si può utilizzare solamente per problemi di classificazione binaria. La classificazione dei record avviene tramite la ricerca del miglior iperpiano che riesca a suddividere lo spazio di n dimensioni generato dai record del *train set*, in modo da avere tutti i record di una classe in uno dei semispazi, e tutti i record della seconda classe nell'altro. Il miglior iperpiano indica quello che riesce ad avere i margini più grandi possibile, cioè la più grande larghezza possibile per una fascia di spazio attorno all'iperpiano trovato in modo tale che al suo interno non vi siano punti rappresentativi di un record.
I *support vectors* sono i punti più vicini all'iperpiano di separazione, per entrambe le classi. Nella figura seguente è rappresentato un esempio di SVM in uno spazio di $n = 2$ dimensioni, in cui le due classi sono rappresentate una da un +, e l'altra da un -

Ogni record x_j del *train set* viene convertito in un punto dello spazio di numeri reali a n dimensioni, a cui viene associata una classe y_j fra le 2 disponibili, indicate con +1 o -1. L'iperpiano viene trovato sfruttando il posizionamento di questi punti, in modo tale che l'iperpiano, definito dalla funzione $f(x) = \beta' x + b$, con

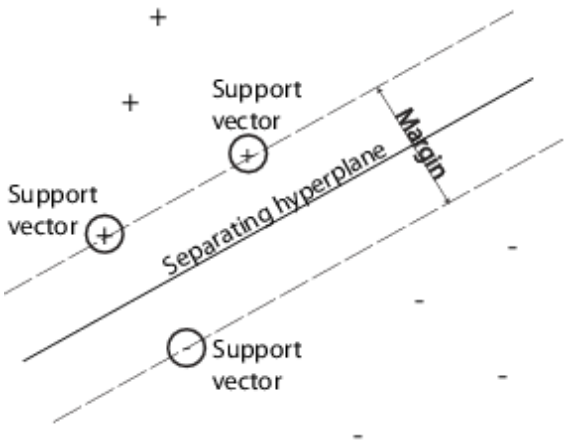

Figura 4.4: Esempio di SVM

$\beta \in \mathbb{R}^n$ e $b \in \mathbb{R}$, li riesca a separare tutti nei due semispazi distinti che genera[31]. Per avere il mioglior iperpiano separatore bisogna trovare i valori di β e b che minimizzano la distanza $\|\beta\|$, e che per ogni punto x_j e relativa classe y_j valga la proprietà $y_j f(x_j) \geq 1$, in modo da avere ancora la separazione di tutti i punti del *train set*. In questo caso, i *support vectors* saranno i punti per i quali $y_j f(x_j) = 1$.

Nel caso in cui i dati del *tain set* non si prestino bene alla separazione tramite iperpiano, per le SVM viene utilizzata la tecnica del *soft margin*, che consiste nel trovare l'iperpiano che suddivida la maggior parte di punti delle due classi, tollerando di avere qualche punto nel semispazio sbagliato, o di averne all'interno della fascia definita dai margini dell'iperpiano.
Esistono due formulazioni standard per l'implementazione dei *soft margin*, ed entrambe utilizzano l'aggiunta di variabili di *slack* ξ_j e di un parametro di penalizzazione C.
La prima corrisponde al problema posto da:

$$\min_{\beta,b,\xi}(\frac{1}{2}\beta' b + C\sum_j \xi_j) \tag{4.13}$$

i cui vincoli sono $y_j f(x_j) \geq 1 - \xi_j$ e $\xi_j \geq 0$, $\forall j$. In questa formulazione, le variabili di *slack* vengono usate normalmente, mentre nella seconda formulazione standard

si impiega il loro quadrato nella formula, che risulta essere:

$$\min_{\beta,b,\xi}(\frac{1}{2}\beta' b + C\sum_{j}\xi_j^2) \tag{4.14}$$

Così facendo, un incremento del valore di C influisce molto più pesantemente nell'intervallo di *slack*, il che induce l'ottimizzazione a cercare un margine di separazione più stretto tra le due classi.

Non tutti i problemi di classificazione binaria si possono risolvere con la costruzione di un semplice iperpiano di separazione, ad esempio quando tutti punti di una delle due classi possono essere facilemnte raggruppati in un *cluster*. Tuttavia esistono delle varianti all'approccio matematico fornito con le SVM che, pur non usando iperpiani, mantengono la loro caratteristica di semplicità. A questo scopo vengono utilizzate delle funzioni di *kernel*:

- esiste una classe di funzioni $G(x_1, x_2)$ con la proprietà che, in uno spazio lineare S esiste una funzione φ per mappare x in S in modo tale che $G(x_1, x_2) = <\varphi(x_1), \varphi(x_2)>$. Il prodotto scalare delle due componenti calcolate con φ viene posizionato nello spazio S;
- in questa classe di funzioni sono comprese:
  - funzioni polinomiali, nella forma $G(x_1, x_2) = (1 + x_1' x_2)^p$ co p intero positivo;
  - funzioni Gaussiane, nella forma $G(x_1, x_2) = e^{(-\|x_1 x_2\|)}$.

L'approccio con utilizzo delle funzioni di *kernel* si basa su quello per la costruzione dell'iperpiano separatore, in quanto anche questo approccio non utilizza altro che prodotti scalari. La differenza consiste nell'utilizzare la funzione di *kernel* scelta al posto di quella che definisce l'iperpiano. Il classificatore ottenuto rappresenta una ipersuperficie nello spazio S, il quale non è detto che sia noto o ben definito.

L'analisi discriminante è un metodo di classificazione basato sulla generazione di k differenti distribuzioni Gaussiane. Per poter classificare un nuovo record, bisogna prima calcolare i parametri delle distribuzioni gaussiane, una per ogni classe presente nei record del *train set*, e successivamente il classificatore utilizza queste distribuzioni per assegnare ad un nuovo record la classe che realizza il minor costo di classificazione errata. L'analisi discriminante lineare utilizzata nel lavoro

svolto, in quanto impostata di default da Matlab che andremo ad illustrare, viene chiamata *Fisher discriminant*[32].

Il modello per la classificazione con analisi discriminante viene costruito assumendo che i dati X del *train set* abbiano una *Gaussian mixture distribution* per ogni classe. Viene quindi utilizzata la stessa matrice di covarianza per ognuna di esse, soltanto il valore della media cambia. Con queste assunzioni, vengono calcolati:

- il valore medio della distribuzione gaussiana per ogni classe k;
- la covarianza semplice, sottraendo il valore medio di ogni classe ai record relativi alla stessa classe, e costruendo la matrice di covarianza dai risultati delle sottrazioni.

Valori medi e matrice di covarianza vengono calcolati utilizzando i dati presenti nel *train set* soltanto doppo che questi vengano pesati. Supponiamo che M sia la matrice rappresentativa delle classi assegnare ai vari record:

- $M_{rk} = 1$ se il record r appartiene alla classe k;
- $M_{rk} = 0$ altrimenti.

Il calcolo del valore medio per la classe k avviene tramite

$$\hat{\mu}_k = \frac{\sum_{r=1}^{n} M_{rk} w_r x_r}{\sum_{r=1}^{n} M_{rk} w_r} \tag{4.15}$$

dove w_r rappresenta il peso assegnato al record r. La matrice di covarianza, sapendo che i pesi assegnati ai record sommano ad 1, viene calcolata con:

$$\hat{\Gamma} = \frac{\sum_{r=1}^{n} \sum_{k=1}^{K} M_{rk} w_r (x_r - \hat{\mu}_k)(x_r - \hat{\mu}_k)^T}{1 - \sum_{k=1}^{K} \frac{W_k^{(2)}}{W_k}} \tag{4.16}$$

dove

- $W_k^{(2)} = \sum_{r=1}^{n} M_{rk} w_r^2$ è la somma dei quadrati dei pesi associati ai record della classe k;
- $W_k = \sum_{r=1}^{n} M_{rk} w_r$ è la somma dei pesi associati ai record della classe k.

Una volta ottenuti i valori per le gaussiane richieste dal metodo, queste si possono utilizzare per classificare un nuovo record. Per far ciò si cerca per il nuovo record la classe che ne minimizza il costo di classificazione atteso, cioè:

$$\hat{y} = \underset{y=1,\ldots,K}{argmin} \sum_{k=1}^{K} \hat{P}(k|x)C(y|k) \tag{4.17}$$

dove $\hat{y}$ indica la classe predetta, K indica il numero totale di classi disponibili, $\hat{P}(k|x)$ è la probabilità a posteriori per la classe k assegnata al nuovo record x, e $C(y|k)$ è il costo di classificare un record con la classe y quando la sua vera classe sarebbe k.
La probabilità a posteriori che un record x appartenga alla classe k è il prodotto tra la probabilità a priori e la multivariata densità normalizzata per la classe k, calcolata utilizzando i valori di media μ_k e covarianza Γ_k tramite la formula:

$$P(x|k) = \frac{1}{(2\pi|\Gamma_k|)^{1/2}} exp(-\frac{1}{2}(x-\mu_k)^T\Gamma_k^{-1}(x-\mu_k)) \tag{4.18}$$

con $|\Gamma_k|$ determinante della matrice di covarianza Γ_k e Γ_k^{-1} sua matrice inversa.
Chiamando $P(k)$ la probabilità a priori della classe k, la probabilità a posteriori che il record x appartenga alla classe k viene calcolata con

$$\hat{P}(k|x) = \frac{P(x|k)P(k)}{P(x)} \tag{4.19}$$

dove $P(x)$ è una costante di normalizzazione, ottenuta dalla somma per ogni k ddel prodotto $P(x|k)P(k)$.
Il costo $C(i,j)$ per la classificazione di un record x nella classe i, quando la sua vera classe è j viene semplicemente ottenuto tramite le regole:

- $C(i,j) = 0$ se $i = j$;
- $C(i,j) = 1$ altrimenti.

In altri termini, il costo di classificazione è pari a 0 per le classificazioni corrette, mentre è pari a 1 per quelle sbagliate.

Capitolo 5

NinaPro DataSet

5.1 Descrizione

I dati relativi ai movimenti della mano utilizzati per costruire i classificatori provati sono appartenenti al database del progetto Non-Invasive Adaptive Hand Prosthetics (NinaPro)[1][25].
Questo database contiene informazioni riguardanti segnali sEMG, cinematica e dinamica della mano ottenuti tramite i sensori del guanto *CyberGlove2*, registrati tutti contemporaneamente durante movimenti compiuti da più soggetti.

Lo scopo del progetto NinaPro è quello di rendere disponibile gratuitamente vari *dataset* contenenti segnali ed informazioni sopra citate, in quanto la comunità scientifica soffre la mancanza di dateset ricchi di soggetti e di movimenti su cui fare esperimenti, per poi successivamente poterne confrontare i risultati più facilmente, dato che l'insieme di dati di partenza risulterà uguale per tutti.
Inoltre, grazie alla condivisione del database a tutti i ricercatori che lo desiderino, spera di riuscire a sviluppare nuove tipologie di algoritmi applicabili a protesi robotiche per aumentarne la destrezza e ridurre le tempistiche di allenamento per poterle usare correttamente da parte di un soggetto con mani o arti amputati.

Attualmente il database NinaPro è formato da 3 diversi *dataset*:

- il primo contiene dati provenienti dai movimenti eseguiti da 27 soggetti normodotati;

[1] `http://ninapro.hevs.ch/`

- il secondo contiene dati provenienti dai movimenti di 40 soggetti normodotati;
- il terzo contiene dati provenienti dai movimenti di 11 soggetti con arti amputati a livello transradiale.

Altri 3 *dataset* sono al vaglio della comunità scientifica internazionale per poter essere validati ed inseriti all'interno del database NinaPro, a dimostrazione che il progetto è valido ed in continua crescita.

5.2 Dataset utilizzato

Il secondo *dataset* del database NinaPro, che utilizzaremo per la costruzione dei classificatori, è descritto approfinditamente in [24][26] e reperibile liberamente dal sito del progetto NinaPro[2].

Tale *dataset* è stato generato da movimenti appartenenti a 3 esercizi differenti:

1. movimenti base delle dita e del polso;
2. prese e movimenti funzionali;
3. movimenti di forza.

Nei primi due esercizi, ogni soggetto deve ripetere i movimenti che gli vengono mostrati su uno schermo, durante i quali vengono salvati i dati cinematici catturati dal guanto *CyberGlove2* indossato dal soggetto.
È presente anche un accelerometro posto attorno al polso, per ottenere ulteriori informazioni dal movimento eseguito.

Nel terzo esercizio, i soggetti devono premere uno o più dita, con forza crescente, su di un dispositivo per il calcolo della forza utilizzata, chiamato *Finger Force Linear Sensor* (Castellini 2012).

L'attività muscolare è raccolta utilizzando 12 elettrodi wireless del tipo *Delsys Trigno Wireless EMG system*. Gli elettrodi sono posizionati come segue:

- otto elettrodi sono equidistanti intorno all'avambraccio in corrispondenza dell'articolazione della radio omerale;

[2] `http://ninaweb.hevs.ch/data2`

- due elettrodi sono posizionati sui punti principali delle dita (flessore ed estensore);
- due elettrodi sono posizionati sui punti principali del bicipite e del tricipite.

Le posizioni degli elettrodi descritte sono state scelte per combinare un metodo di campionamento a valori densi e una strategia di posizionamento anatomico preciso.

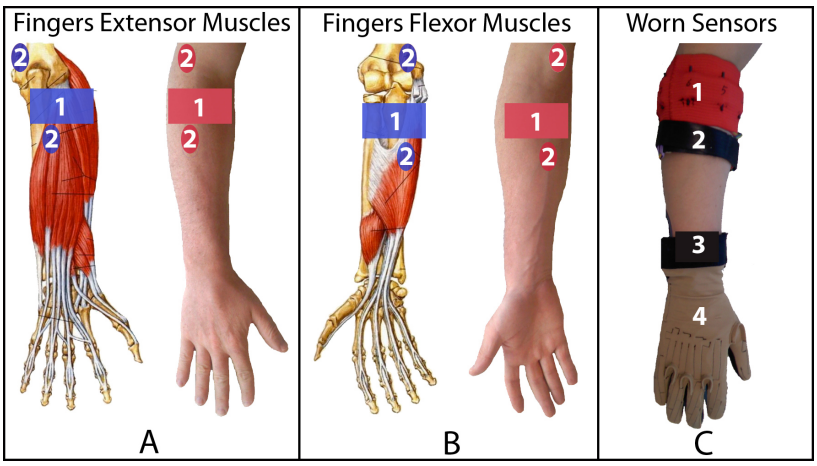

Figura 5.1: Muscoli coinvolti nei vari esercizi

Gli elettrodi sono stati fissati all'avambraccio con le loro bande adesive standard. Inoltre, una banda in lattice elastico anallergico è stata posta intorno agli elettrodi in modo da tenerli fissi durante l'esecuzione dei movimenti e l'acquisizione dati.

I dati cinematici del movimento eseguito sono catturati tramite guanto *CyberGlove2*[3], questo dispositivo wireless viene utilizzato per il *motion-capture*, grazie a 22 sensori ad alta precisione per misurazione di angoli dei giunti della mano effettuate durante l'esecuzione di un movimento. La disposizione dei sensori è strutturata in questo modo:

- 3 sensori per misurare la flessione di ogni dito;
- 4 sensori per le misure di abduzione;
- 1 sensore per il movimento dell'arco palmare;

[3] `http://www.cyberglovesystems.com/cyberglove-ii/`

- i restanti sono utilizzati per registrare la rotazione effettuata dal polso.

Il campionamento dati è stato eseguito con una frequenza di 2 kHz.

Durante l'acquisizione, i soggetti sono stati invitati a ripetere più volte i movimenti con la mano destra.
Ogni ripetizione è durata 5 secondi ed è stata seguita da 3 secondi di riposo.
Il protocollo comprende 6 ripetizioni di 49 diversi movimenti (più la posizione di riposo) eseguite da 40 soggetti normodotati.
I movimenti da eseguire sono stati selezionati dalla letteratura sulle mani robotiche.

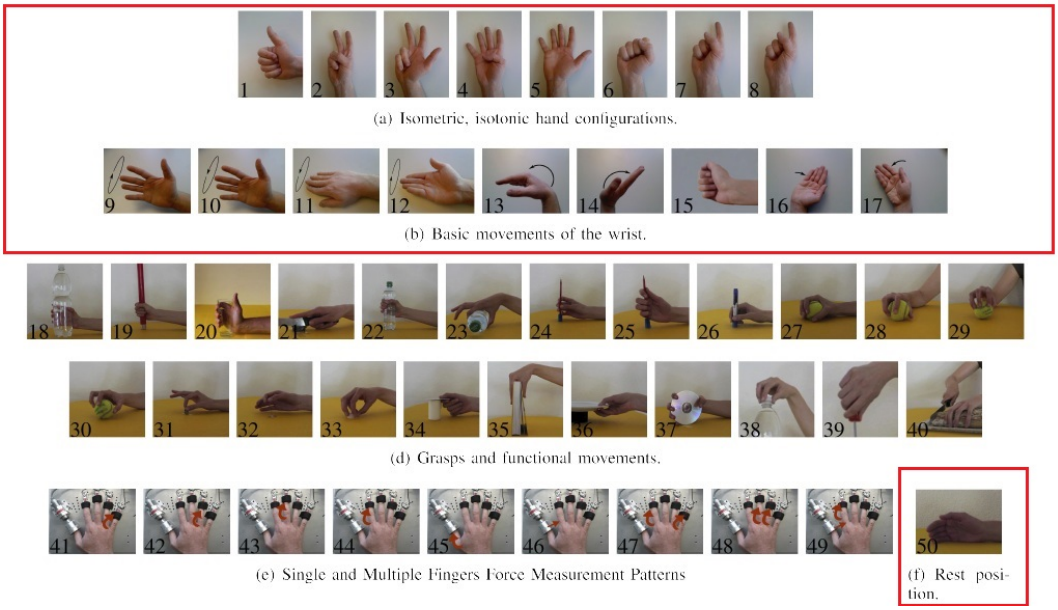

Figura 5.2: Movimenti eseguiti nei 3 esercizi

Per ogni esercizio eseguito da ogni soggetto, il *dataset* presenta un file Matlab che contiene:

- subject: numero del soggetto;
- exercise: numero dell'esercizio;
- acc (36 colonne): accelerometri a tre assi dei 12 elettrodi;
- emg (12 colonne): segnali EMG dei 12 elettrodi;

- glove (22 colonne): segnali non calibrati provenienti dai 22 sensori del *CyberGlove2*;
- inclin (2 colonne): segnali a due assi provenienti dall'accellerometro posizionato sul polso;
- stimulus (1 colonna): movimento ripetuto dal soggetto;
- restimulus (1 colonna): movimento ripetuto dal soggetto. In questo caso, l'etichetta è definita a posteriori per avere una maggiore corispondenza tra i dati raccolti e il movimento reale;
- repetition (1 colonna): ripetizione dello stimulus;
- rerepetition (1 colonna): ripetizione del restimulus;
- force (6 colonne): forza registrata durnte l'esecuzione del terzo esercizio;
- forcecal (2 x 6 valori): i valori di calibrazione dei sensori di forza (minimo e massimo).

I 22 sensori del guanto *CyberGlove2* forniscono dati grezzi, salvati nei file Matlab forniti dal database.
I dati utilizzati per la costruzione dei classifiactori saranno quelli provenienti dalla matrice glove.

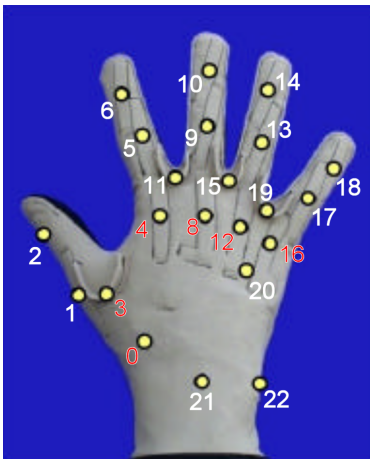

Figura 5.3: Sensori e loro posizioni nel guanto *CyberGlove2*

Durante lo svolgimento della tesi sono stati utilizzati i dati dei movimenti da 1 a 17 di Figura 5.2, relativi al primo esercizio del secondo *dataset* del progetto

NinaPro.

Il codice per l'analisi e la creazione dei classificatori è stato scritto in modo tale che, impostando correttamente un set di parametri, si possa ripetere il lavoro eseguito anche per gli altri esercizi del *dataset* in modo relativamente semplice.

5.3 Estrazione dati dal dataset

Il secondo *dataset* NinaPro risulta composto da 40 directory, una per ogni soggetto, etichettate *s1, s2, ..., s40*.
All'interno di ogni directory sono presenti 3 file Matlab, uno per ogni esercizio svolto. Ciascun file contiene una matrice di risultati, una per ogni punto presente nell'elenco al paragrafo 5.2

I dati utilizzati nella tesi sono quelli presenti nella matrice glove, contenente le misurazioni dei sensori del guanto *CyberGlove2* fatte durante le ripetizioni dei vari movimenti appartenenti al primo esercizio.
Queste informazioni sono state estratte dal *dataset* e messe in una matrice a 22 colonne, una per ogni giunto del guanto, le cui righe sono formate dai valori campionati dal guanto durante l'esecuzione di un movimento. È stata creata una di queste matrici per i valori registrati durante ognuna delle 6 ripetizione di ognuno dei 17 movimenti, per ognuno dei 40 soggetti.

Grazie a queste matrici, si può rappresentare la variazione di angolo catturata da ogni sensore del *CyberGlove2* durante il movimento eseguito da un soggetto come un segnale che varia nel tempo, il cui andmento può essere rappresentato graficamente come nelle immagini seguenti.

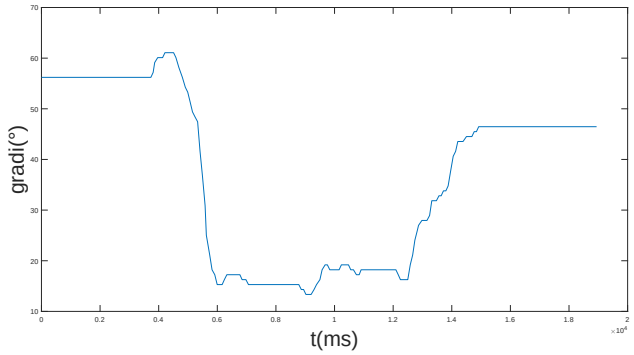

Figura 5.4: Andamento del segnale relativo al giunto 1 per il tentativo 1 del movimento 1 eseguito dal soggetto 1

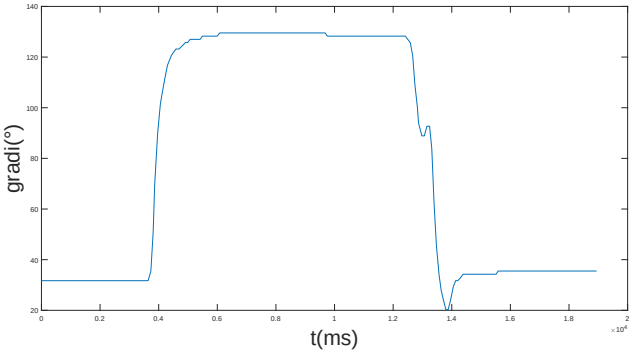

Figura 5.5: Andamento del segnale relativo al giunto 9 per il tentativo 1 del movimento 1 eseguito dal soggetto 1

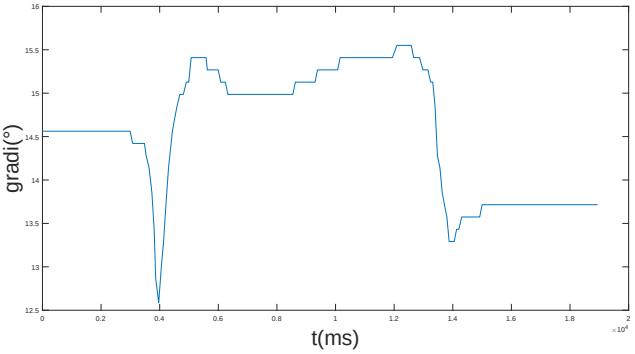

Figura 5.6: Andamento del segnale relativo al giunto 19 per il tentativo 1 del movimento 1 eseguito dal soggetto 1

Capitolo 6

Sviluppo

Per raggiungere lo scopo di costruire un classificatore di movimenti della mano partendo dal *dataset* fornito dal progetto NinaPro bisogna innanzitutto analizzare i dati forniti per cercare delle possibili tecniche di estrazione *features* che rappresentino i movimenti stessi.

I dati con cui si è lavorato sono quelli registrati dal guanto *CyberGlove2* durante i movimenti eseguiti dai soggetti. Durante ogni movimento, la mano parte dalla posizione di riposo, esegue la traiettoria del movimento, e ritorna alla posizione di riposo. Avendo una frequenza di 2KHz, il guanto registra 2000 campioni al secondo, che possono essere rappresentati come un segnale in funzione del tempo. Per ogni movimento eseguito si hanno così 22 segnali che lo rappresentano.

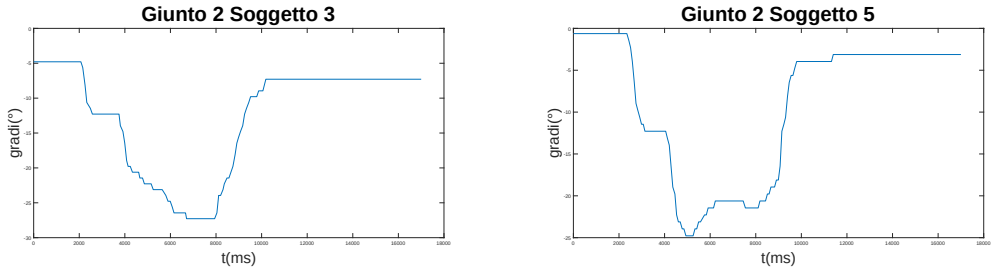

Figura 6.1: Esempio di segali ottenuti per uno stesso movimento e stesso giunto per due soggetti differenti

6.1 Estrazione DMP

Le DMP possono essere calcolate conoscendo il moto di un punto, avendo cioè posizione iniziale e finale, traiettoria, velocità e accelerazione. Queste informazioni possono essere ricavate dai segnali forniti dal guanto in quanto:

- il punto iniziale e finale sono il primo e ultimo valore registrato dal guanto durante il movimento;
- la traiettoria è rappresentata dall'andamento del grafico stesso;
- la velocità viene ricavata derivando il segnale;
- l'accelerazione è ottenuta derivando a sua volta l'andamento della velocità ottenuta.

Il segnale di un giunto risulta essere un array di valori contenente i campioni misurati dal sensore del guanto, e quindi un insieme di valori discreti finito

$$S = [c_1, c_2, ..., c_n] \tag{6.1}$$

Eseguendo una derivazione discreta su di esso si ottiene un altro array rappresentante la velocità del questo movimento. Ciò è possibile in Matlab tramite l'utilizzo della funzione *diff(S)*.

Oltre alle informazioni sulla traiettoria del giunto appena calcolate, la funzione per il calcolo delle DMP sfruttata, chiamata *dmptrain*, vuole in ingresso altri 2 parametri: il numero di funzioni base n_b che verranno utilizzate per ricostruire la traiettoria fornita e il grado del sistema da risolvere per calcolare le DMP. Come output viene fornito un array di dimensioni n_b chiamato Θ

$$\Theta = [\theta_1, \theta_2, ..., \theta_{n_b}] \tag{6.2}$$

contenente i pesi che avranno le funzioni base nella ricostruzione. I θ ottenuti verranno considerati successivamente come *features* caratterizzanti del segnale per la costruzione di un Classification tree (paragrafo 4.1).
L'idea di fondo è quella che, se due movimenti sono uguali, la traiettoria compiuta dai giunti della mano durante i movimenti dovrà essere molto simile, e quindi le DMP dovrebbero generare dei valori di Θ simili per ogni coppia di giunti.

Il calcolo delle DMP per ogni segnale dei giunti è stato eseguito per un numero di funzioni base n_b pari a 8, 16 e 32. Più alto è il numero di funzioni, maggiore è la possibilità di ricostruire molto precisamente segnali con andamento articolato. Di contro, anche se due segnali sono somiglianti, ci saranno delle discrepanze nei θ calcolati, dovute alle differenze tra i due segnali.

Inizialmente si è cercato di ricavare le DMP per il movimento 1 eseguito dal soggetto 1, per testare il programma scelto, capirne il funzionamento, e successivamente utilizzarlo con tutti i movimenti di più soggetti.

È stato riscontrato che, per alcuni giunti, l'andamento del loro movimento presentava punti in cui era problematico il calcolo della derivata, o in cui la traiettoria compiuta era troppo "spigolosa" per permettere alle DMP di risolvere il sistema e calcolarne i θ delle funzioni base.
Per togliere queste imperfezioni da un segnale, Matlab mette a disposizione la funzione *smooth(X)*, con X array contenente i valori del segnale nel tempo.
È possibile anche specificare la tipologia di *smooth* da applicare, scelta fra quelle messe a disposizione dalla funzione. I migliori risultati si sono ottenuti con il metodo Savitzky-Golay[40]. Questa tipologia di *smoothing* è basata su media mobile generalizzata del segnale, filtrata da coefficienti determinati da una regressione lineare quadrata minima e non ponderata, su di un modello polinomiale di grado conosciuto.
Seppur impercettibili ad occhio nudo, le migliorie portate dallo *smoothing* sono risultate essenziali per il proseguimento del lavoro, permettendo di utilizzare il codice di estrazione delle DMP correttamente.

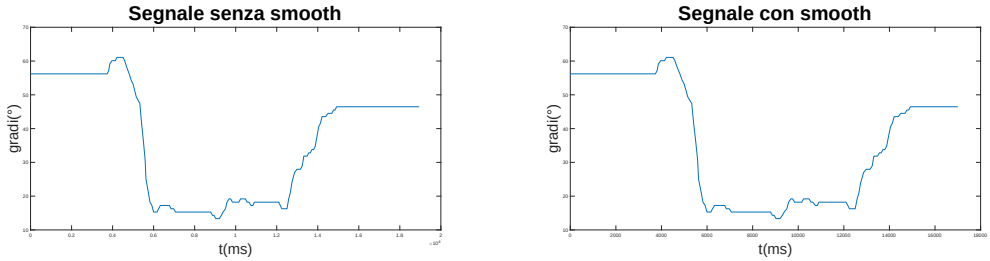

Figura 6.2: A destra: il segnale del giunto 1 per il movimento 1 eseguito dal soggetto 1 nel primo tentativo. A sinistra: segnale del giunto dopo *smoothing*

Fornendo in ingresso alla funzione *dmptrain* il segnale di un giunto dopo lo *smoothing* come traiettoria, con velocità e accelerazione ottenute tramite derivata,

ed i punti di inizio e fine sempre ricavati da segnale del giunto vengono calcolate tutte le componenti delle DMP, visualizzabili anche tramite una serie di grafici esplicativi. La parte che ci interessa e che utilizzeremo per il proseguo del lavoro è formata dai θ estratti, dal segnale ricostruito confrontato con quello originale e dall'andamento e numero delle funzioni base utilizzate.

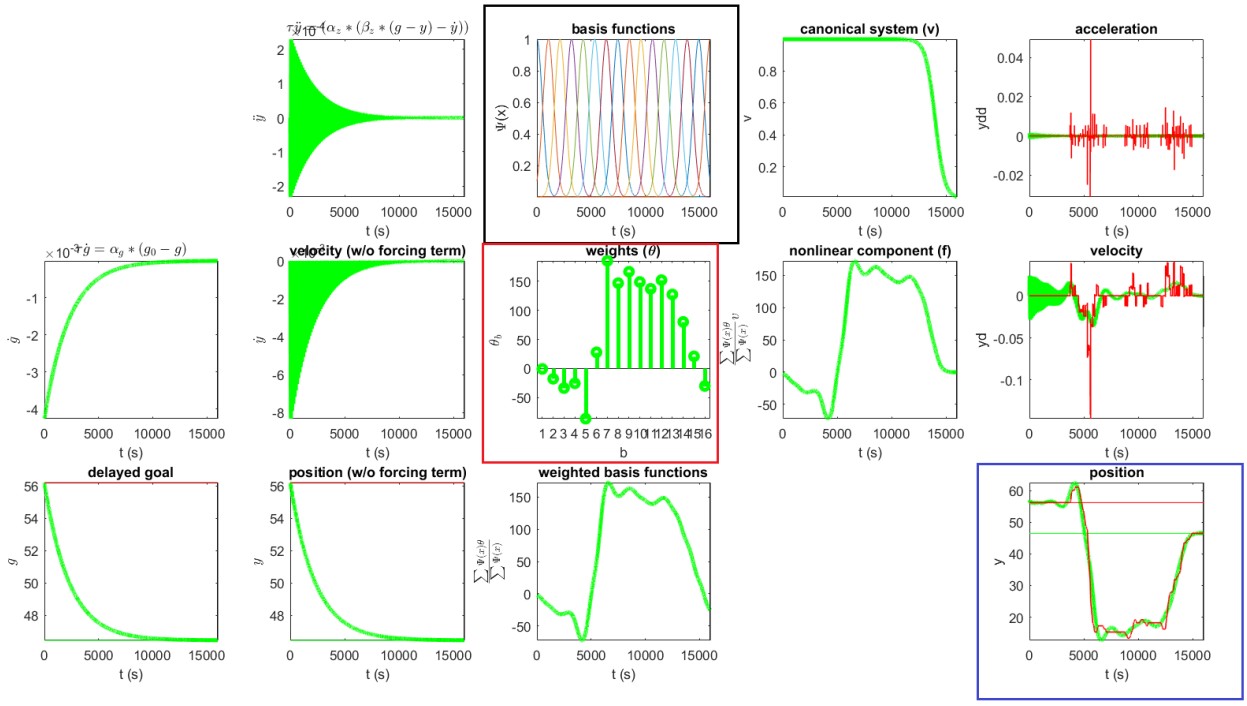

Figura 6.3: Calcolo delle DMP per il giunto 1 del movimento 1 per il soggetto 1.
Evidenziato in nero: andamento delle 16 funzioni base utilizzate
Evidenziato in rosso: i 16 θ calcolati dalle DMP
Evidenziato in blu: andamento del giunto tratto dal database NinaPro (verde) e andamento del segnale ricostruito grazie alle funzioni base e ai loro θ (rosso)

Per l'estrazione delle DMP sono stati provati tutti e 3 i sistemi proposti dalla funzione *dmptrain* illustrati al paragrafo 3.2.1. Teoricamente, il sistema più avanzato disponibile è quello di ordine 3 (sigmoidale), ed è quello che utilizzeremo in tutti i calcoli delle DMP nel proseguo del lavoro. Praticamente, la traiettoria del giunto ricostruita grazie ai θ ottenuti dal sistema di ordine 3 risulta di poco migliore a quelle ricostruite sfruttando i valori invece ottenuti dai sistemi di ordine 1 e 2.

6.1.1 Somiglianza segnale ricostruito

Per verificare che i θ ottenuti dal calcolo delle DMP possano essere utilizzati come *features* caratterizzanti del segnale stesso, si è reso necessario confrontare l'andamento tra il segnale originale preso dal *dataset* e quello ricostruito sfruttando le funzioni base e i rispettivi θ. Come riferimento di partenza, è stato preso il movimento 1 eseguito dal soggetto 1 nei suoi 6 tentativi, con particolare attenzione ai segnali provenienti dai giunti 1 e 2.

Il primo confronto è stato eseguito visivamente, prendendo in analisi il grafico fornito dalla stessa funzione *dmptrain* in cui i due andamenti venivano disegnati sullo stesso grafico. Si può ben vedere che, all'aumentare del numero di θ impostati per le DMP, il segnale ricostruito è maggiormente somigliante a quello di ingresso. Ciò nonostante, sia da 8, da 16 e da 32 θ si ottiene un segnale ricostruito che segue notevolmente l'andamento di quello originale.

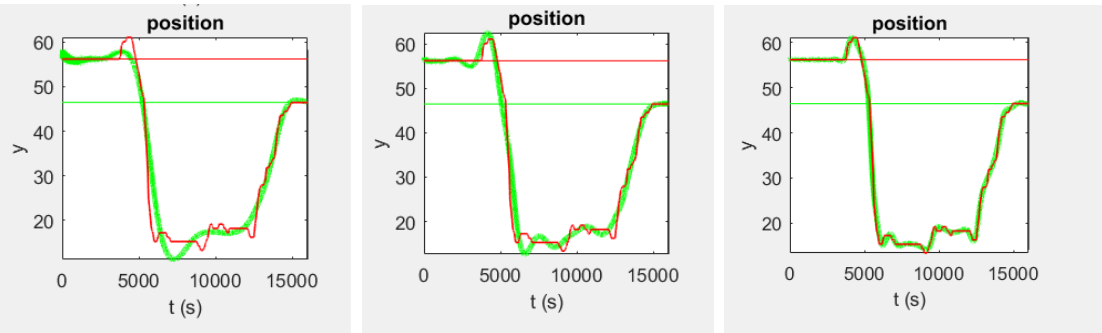

Figura 6.4: Segnale del giunto 1 per il movimento 1 eseguito dal soggetto 1 nel primo *trial* e ricostruito con 8, 16 e 32 funzioni base

Dunque, i valori di θ estratti dalla funzione *dmptrain* ricostruiscono il segnale originale in maniera molto precisa. Per avere un riscontro maggiore della bontà della ricostruzione, si è passati a misurare tale bontà, tramite l'utilizzo della Cross Covariance (*xcov*) e Cross Correlation (*xcorr*), spiegate rispettivamente nei paragrafi 3.3.1 e 3.3.2. Dato che entrambe le misure confrontano un segnale con la trasposizione nel tempo del secondo, in uscita dalle funzioni *xcov* e *xcorr* si ottiene l'andamento di una funzione in cui, per ogni istante, viene calcolata la rispettiva misura di somiglianza tra le due sezioni di segnale confrontate. Per questo motivo si avrà, per entrambe le misure, un andamento che presenta un solo valore come massimo assoluto, corrispondente ai due segnali da confrontare perfettamente alli-

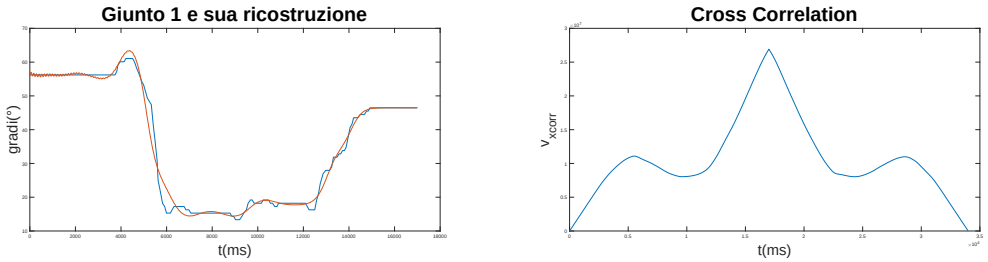

Figura 6.5: A sinistra: Confronto segnale del giunto 1 del movimento 1 (blu) con sua ricostruzione a 16 θ (rosso).
A destra, andamento di $xcorr$ con $v_{xcorr} = 2.4737e{+}07$

neati. Grazie al valore di questo picco si può confrontare la bontà della somiglianza fra due segnali:

$$v_{xcov} = max(xcov(S_i, R_i)) \qquad v_{xcorr} = max(xcorr(S_i, R_i)) \tag{6.3}$$

con S_i andamento del segnale ottenuto dal giunto i ed R_i andamento ricostruito grazie alle DMP del segnale del giunto i.

Sono stati ottenuti valori alti per entrambe le misure: ciò ci induce a pensare che il segnale originale e quello ricostruito siano molto somiglianti, a conferma di quanto accertato visivamente.
Tuttavia, tra le due misure di somiglianza è risultata migliore $xcorr$, in quanto per alcuni giunti, la misura $xcov$ presentava valori maggiori per due andamenti visivamente diversi, presi da due giunti differenti, piuttosto che per due andamenti molto simili in quanto derivati dallo stesso giunto, ma per due *trial* differenti. Ciò può essere dovuto al fatto che $xcov$ considera in maniera più determinante l'ampiezza del segnale rispetto a $xcorr$.

Per riuscire a creare un numero sufficientemente alto di record, a partire dalle DMP estratte, da passare poi ai modelli di classificazione sono stati calcolati i θ per i 22 giunti di tutti i 17 movimenti eseguiti nei 6 tentativi dai primi 10 soggetti, utilizzando un numero n_b di funzioni base pari a 8, 16 e 32. Una volta che si sarà ottenuto un modello di classificazione soddisfacente, verranno aggiunte le informazioni di tutti i soggetti del *dataset* per la creazione di un modello definitivo: per capire però quali impostazioni dargli, serviranno varie prove, effettuate a partire dal campione rappresentativo composto dai primi 10 soggetti.

I θ estratti per i primi 10 soggetti sono stati ricavati sia dai segnali dei giunti grezzi, sia dalla loro versione normalizzata, dove il supporto del segnale viene rimappato tra zero e uno. Per fare ciò viene utilizzata la formula:

$$S_{i_{normalized}} = \frac{S_i - min(S_i)}{max(S_i) - min(S_i)} \tag{6.4}$$

Verranno sfruttati i segnali normalizzati e non per allenare i classificatori, per capire se sarà utile aggiungere la fase di normalizzazione permanentemente per ottenere risultati migliori dai modelli creati.
La fase di normalizzazione potrebbe rivelarsi utile in quanto renderebbe i segnali molto più limitati evidenziandone l'andamento, soprattutto fra due segnali simili. Di contro, però, si avrebbe una perdita di informazioni riguardo l'ampiezza del movimento eseguito dal giunto, fattore molto importante per l'estrazione delle DMP.

6.2 Alberi con DMP

I valori di θ estratti per il movimento di un giunto vengono utilizzati, in questa fase, come *features* rappresentative del segnale stesso. La riproduzione di un movimento della mano è formata dai dati raccolti dal guanto *CyberGlove2* per tutti e 22 i giunti presenti, quindi per ogni tentativo di movimento fatto da un soggetto costruiamo un record che lo rappresenti formato in questo modo:

$$r_i = [\theta_{G1_1}, ..., \theta_{G1_{n_b}}, \theta_{G2_1}, ..., \theta_{G2_{n_b}}, \theta_{G22_1}, ..., \theta_{G22_{n_b}}, c_i] \tag{6.5}$$

in cui i valori $[\theta_{G1_1}, ..., \theta_{G1_{n_b}}]$ sono quelli estratti dalle DMP dal segnale del giunto 1, $[\theta_{G2_1}, ..., \theta_{G2_{n_b}}]$ sono quelli estratti dalle DMP dal segnale del giunto 2, ecc. e dove c_i indica il movimento eseguito nel tentativo rappresentato dal record r_i.
I record creati in questo modo formeranno il *training set* per la costruzione di un Classification tree (paragrafo 4.1).

Non sono stati utilizzati fin da subito i dati provenienti da tutti i soggetti e loro movimenti: prima è necessario capire se la classificazione di movimenti tramite alberi di classificazione e DMP sia una strada promettente da percorrere. Per

questo motivo, i primi Classification tree sono stati costruiti soltanto con i dati provenienti dal soggetto 1 per classificare 3 movimenti distinti tra di loro.

Una prima scrematura sui 3 movimenti da scegliere è stata fatta visivamente, cercando di capire quali parti della mano erano maggiormente coinvolte in ciascun movimento. Si sono cercati movimenti con parti coinvolte differenti, anche per facilitarne poi la classificazione. Se già per 3 movimenti distinti il modello predittivo non fosse abbastanza accurato sarebbe un segnale che gli alberi di classificazione non sono il giusto modello da utilizzare con i θ usati come features.

Figura 6.6: I 3 movimenti di partenza selezionati

La scelta è ricaduta sui movimenti 1, 4 e 12, in quanto:

- per il movimento 1 il dito maggiormente utilizzato è il pollice;
- per il movimento 4 vengono usate tutte le altre 4 dita;
- il movimento 12 esegue una rotazione che coinvolge principalmente il polso e nessun dito.

Sono stati analizzati tutti i giunti coinvolti in questi movimenti, per capire quali fossero quelli rappresentativi per i 3 movimenti. In questo caso si avrebbe la conferma della diversità dei 3 movimenti anche strumentalmente e quindi una maggiore facilità di classificazione.

Un giunto viene definito rappresentativo per un movimento se è molto coinvolto nello stesso, e presenta andamenti del segnale simili per tutti i tentativi fatti da ogni soggetto. Ciò si può capire disegnando gli andamenti del giunto e riuscendo a distinguere un andamento comune. Se invece un giunto è marginale al movimento, disegnandone gli andamenti per i vari tentativi dei soggetti si ottiene un grafico in

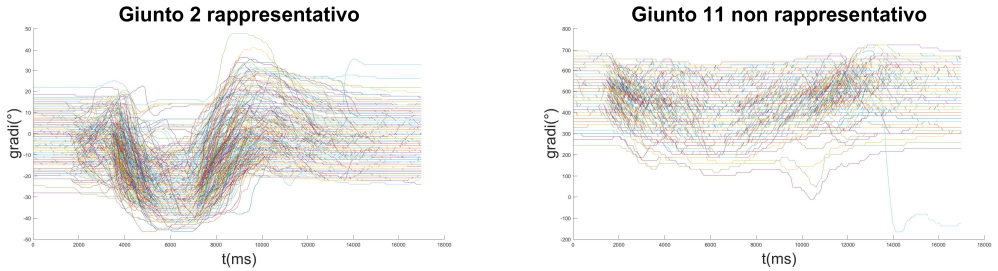

Figura 6.7: Grafici degli andamenti per un giunto rappresentativo ed uno no del movimento 1

cui quasi tutta l'area viene coperta dai segnali dei giunti, in quanto non hanno un andamento comune e non utilizzabile per distinguere il movimento da altri.

Per i 3 movimenti selezionati i giunti trovati più rappresentativi analizzando tutti gli andamenti presi da ogni *trial* di ogni soggetto sono riassunti nella tabella:

Movimento	**Giunti migliori**					
M_1	2	3	6	9	13	17
M_4	1	2	5	8	11	17
M_{12}	1	9	11	12	13	22

Tabella 6.1: Giunti rappresentativi per i movimenti 1, 4 e 12

Come si può vedere, per ogni movimento è sempre presente almeno un giunto non caratterizzante anche per gli altri due movimenti.

I *training set* per la costruzione degli alberi vengono ora creati prendendo 5 *trial* da ogni movimento e mettendo quello restante nel *test set* per la valutazione della precisione. Da questi pochi dati vengono costruiti alberi molto semplici, in quanto si riesce a trovare quasi subito una variabile che suddivida bene i pochi record presenti. I risultati di precisione risultano comunque bassi: avendo pochi record nel *test set*, basta che uno non sia classificato correttamente per abbassare di molto la precisione dell'albero. In queste condizioni, utilizzando i θ estratti da segnali normalizzati si sono ottenuti risultati identici a quelli avuti con i θ dei segnali senza normalizzazione.

La causa principale dei scarsi risultati di precisione nella classificazione ottenuta dagli alberi creati è la poca numerosità di record presenti nel *train* e *test set*.

	Trial di test					
n_b	**1**	**2**	**3**	**4**	**5**	**6**
8	1/3	1/3	1/3	1/3	1/3	1/3
16	1/3	1/3	0/3	1/3	0/3	1/3
32	1/3	1/3	0/3	0/3	1/3	1/3

Tabella 6.2: Classificazioni corrette per i movimenti 1, 4 e 12

Questi sono stati aumentati utilizzando per il *train set* i θ ottenuti da 5 *trial* di un soggetto e il rimanente per il *test set*, considerando tutti i 17 movimenti e i primi 10 soggetti.

L'aumento di record nel *train* e *test set* ha portato a miglioramenti negli alberi creati: con $n_b = 8$ l'albero migliore ha ottenuto il 41% di accuratezza, con $n_b = 16$ il 58,8% di accuratezza, mentre con $n_b = 32$ ci si è fermati al 52%. Da questi nuovi dati emerge il fatto che i risultati migliori sono ottenuti utilizzando un numero di funzioni base pari a 16: infatti per ognuno dei 10 soggetti l'accuratezza degli alberi creati con $n_b = 16$ sono state quai sempre maggiori rispetto a quelle ottenute con $n_b = 8$ o 32 . Da qui in avanti, quindi, verranno utilizzati principalmente le DMP calcolate con questo valore di n_b.

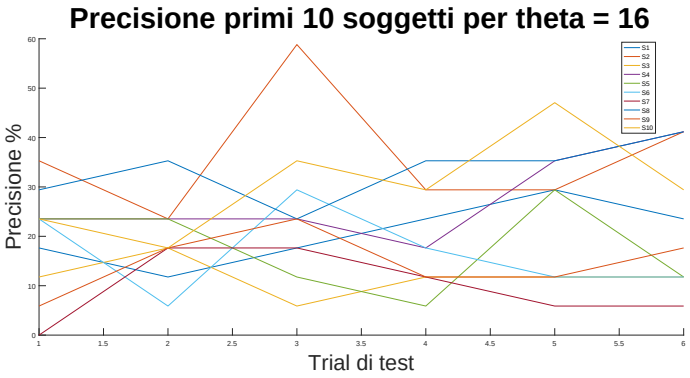

Figura 6.8: Andamenti precisione alberi con $n_b = 16$ dei primi 10 soggetti

Sono stati costruiti gli stessi alberi utilizzando i θ ottenuti dai segnali dei giunti normalizzati, per testare se portassero a risultati migliori. Così non è stato: le percentuali di precisione si sono rivelate sempre inferiori rispetto a quelle ottenute da alberi creati con i θ non normalizzati. Questa ulteriore constatazione induce a pensare che la fase di normalizzazione dei θ non sia necessaria, ma che sia meglio

utilizzare i valori grezzi ottenuti direttamente in uscita dalla funzione *dmptrain*. Gli alberi di classificazione migliori ottenuti in questa fase sono stati quelli generati dai dati appartenenti al soggetto 2. In particolare, il migliore di tutti è stato creato utilizzando il terzo *trial* di ogni movimento come *test set*.

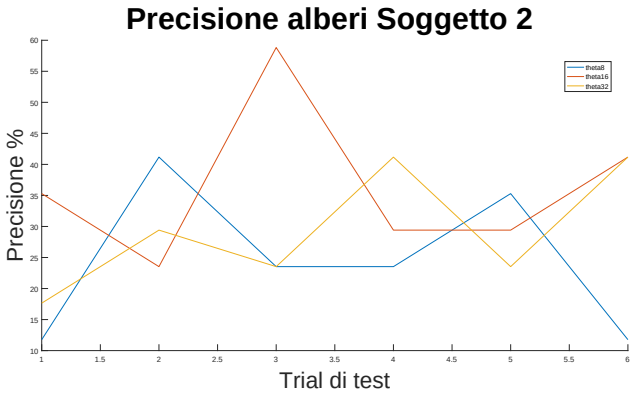

Figura 6.9: Andamenti precisione alberi del soggetto 2: in rosso i valori per $n_b = 16$

Il miglioramento di risultati ottenuto induce a pensare che la classificazione tramite DMP e Classification tree sia buona. La prova successiva coinvolge un maggior numero di record nei *train* e *test set*: dei primi 10 soggetti, i θ di 9 di loro vengono utilizzati come *train set*, il dati del soggetto escluso viene utilizzato invece come *test set*. Con questa configurazione, si hanno un buon numero di record per entrambi i set di dati, in particolare abbiamo:

- 6 *trial* * 17 movimenti * 9 soggetti = 918 record per il *train set*;
- 6 *trial* * 17 movimenti * 1 soggetto = 102 record per il *test set*;
- ciascun record ha ovviamente 22 giunti * 16 θ = 352 variabili valorizzate.

Con un numero alto di variabili a disposizione nei record, gli alberi creati risultano molto più complessi rispetto a tutti i precedenti, per cui sono stati provati diversi livelli di *pruning* per limitarne la dimensione ed evitare il problema dell'*overfitting*.

Questi alberi, creati con $n_b = 16$ e con i θ grezzi, non sono risultati buoni come si sperava. Infatti la precisione migliore, che si è ottenuta utilizzando il soggetto 10 come *test set* ed un livello di *prune* di 3, è risultata pari soltanto al 32,3%. Per capire se la bassa precisione fosse dovuta al tipo di dati scelti, sono stati creati

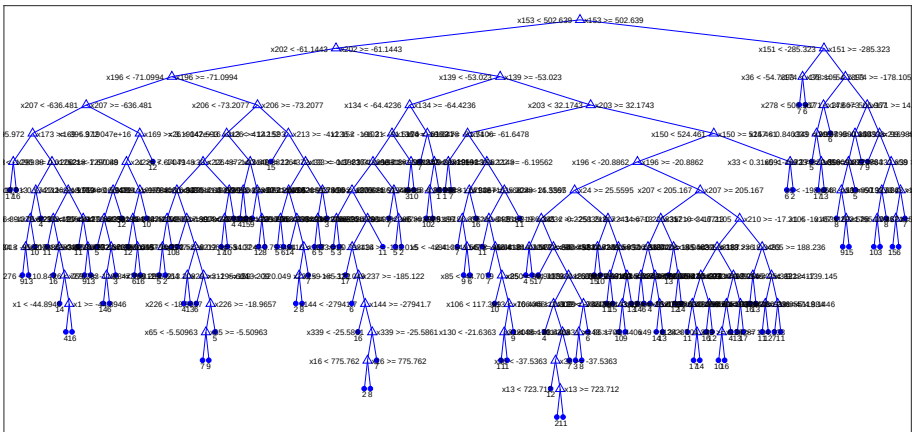

Figura 6.10: Classification tree costruito dai dati di 9 soggetti

anche i Classification tree utilizzando i θ provenienti dai segnali normalizzati e anche per gli altri valori di funzioni base 8 e 32. I risultati, riassunti in tabella, sono comunque pari o inferiori: ciò avvalora il fatto che $n_b = 16$ sia una buona scelta fra quelle prese in considerazione e lo stesso dicasi per l'utilizzo di θ grezzi rispetto a quelli normalizzati.

n_b	**Soggetto di test**	**Livello di *prune***	**Precisione**
8	7	3	26,4 %
16	10	3	32,3 %
32	2	3	32,3 %

Tabella 6.3: Percentuali massima correttezza per alberi con θ grezzi

n_b	**Soggetto di test**	**Livello di *prune***	**Precisione**
8	1	3	25,5 %
16	10	3	32,3 %
32	2	3	32,3 %

Tabella 6.4: Percentuali massima correttezza per alberi con θ normalizzati

Per confermare la tesi secondo cui i θ calcolati non siano sufficientemente buoni per poter costruire un Classification tree soddisfacente sì è provato a disegnare gli andamenti dei θ stessi per i 3 movimenti inizialmente scelti. Se visivamente si riesce a distinguere un andamento simile dei θ per ognuno dei 3 movimenti significa che i dati sono buoni e la bassa percentuale di classificazioni corrette deriva

soltanto dal fatto che i dati nei due set non sono ancora sufficienti.
Disegnamo, ad esempio, l'andamento dei θ per il giunto 9 ottenuto dai 3 movimenti eseguiti dal soggetto 2: in rosso sono colorati i θ generati dal movimento 1, in verde quelli del movimento 4, in blu quelli del movimento 12.

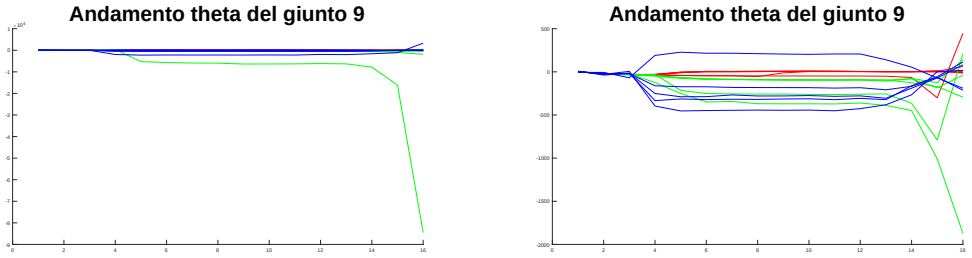

Figura 6.11: Andamento θ per il giunto 9 con *outlier* e senza

Dal grafico a destra si può notare che un *trial* ha generato dei θ *outlier*, cioè dei θ il cui valore è molto elevato o molto minore rispetto alla media ottenuta dagli altri *trial*. Questi valori fuori media possono influire negativamente sulla classificazione, in quanto forniscono un esempio molto meno rappresentativo del movimento rispetto agli altri 5, soprattutto se questi ultimi si assomigliano molto tra loro.
Nel grafico di sinistra gli *outlier* sono stati rimossi, per poter procedere con l'analisi del grafico e per riuscire a distinguere eventuali andamenti uguali per i θ dei movimenti 1 e 12, per i quali il giunto 9 è rappresentativo. Purtroppo soltanto alcuni *trial* generano andamenti nei θ somiglianti tra loro, mentre alcuni presentano, per *trial* diversi dello stesso movimento, valori discordanti. Questa situazione è più facilmente individuabile nel grafico del giunto 2, sul quale non si riesce a distinguere i singoli movimenti soltanto dai valori dei θ.
Queste situazioni (presenza di *outlier* ed indistinguibilità) sono presenti nei grafici di ciascuno dei 22 giunti.

Un possibile miglioramento potrebbe essere ottenuto normalizzando i valori dei θ stessi invece dei segnali dei giunti, in quanto ci interessa che i θ di movimenti uguali abbiano lo stesso andamento per facilitarne la classificazione. Tuttavia, anche in questo caso, i 3 movimenti non risultano distinguibili dagli andamenti grafici, pertanto anche una eventuale normalizzazione dei θ non porterebbe benefici agli alberi di classificazione.

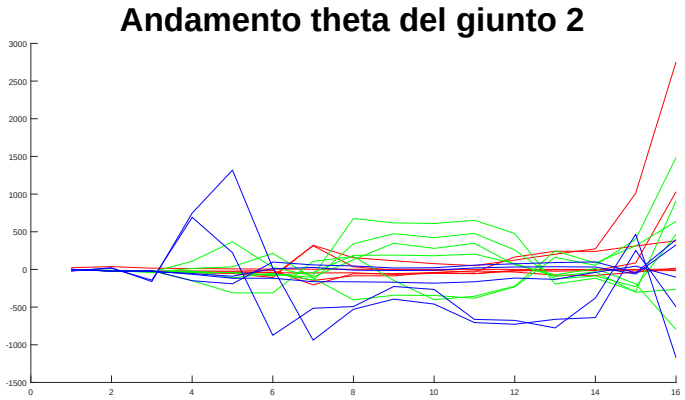

Figura 6.12: Andamento θ del giunto 2

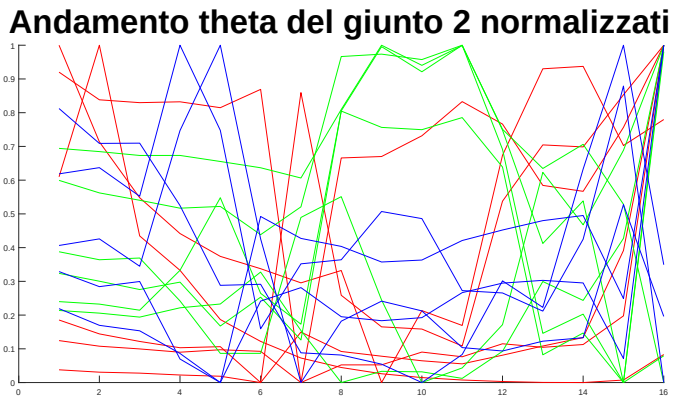

Figura 6.13: Andamento θ del giunto 2 dopo averli normalizzati

Date le basse percentuali di classificazioni corrette ottenute e l'analisi degli andamenti dei θ che hanno evidenziato come gli stessi non erano abbastanza buoni per la classificazione tramite alberi, si è passati alla costruzione di modelli GMM per capire se, tramite questa diversa tecnica, si potessero ottenere risultati di classificazione migliori.

6.3 GMM con DMP

Per poter costruire un modello GMM è necessario, come prima cosa, disporre i θ provenienti dal calcolo delle DMP come richiesto dal modello, per poter essere utilizzate come *train set* su cui allenare le GMM. Seguendo le informazioni presentate

al paragrafo 4.2, viene costruita la matrice di *training*:

$$R_6 = \begin{array}{c} \\ S_1M_1 \\ \vdots \\ S_1M_1 \\ S_1M_4 \\ \vdots \\ S_1M_4 \\ S_1M_{12} \\ \vdots \\ S_1M_{12} \\ S_2M_1 \\ \vdots \\ S_9M_{12} \end{array} \begin{array}{c} \begin{array}{ccccc} g_1 & g_2 & \cdots & g_{22} & m \end{array} \\ \begin{pmatrix} t_1 & t_1 & \cdots & t_1 & m_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ t_5 & t_5 & \cdots & t_5 & m_1 \\ t_1 & t_1 & \cdots & t_1 & m_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ t_5 & t_5 & \cdots & t_5 & m_2 \\ t_1 & t_1 & \cdots & t_1 & m_3 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ t_5 & t_5 & \cdots & t_5 & m_3 \\ t_1 & t_1 & \cdots & t_1 & m_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ t_5 & t_5 & \cdots & t_5 & m_3 \end{pmatrix} \end{array} \tag{6.6}$$

con i dati provenienti dai θ estratti quando $n_b = 16$.
Vengono create 6 matrici di *training* e altrettante di *test*, in quanto per ognuna viene scelto uno dei 6 *trial* da utilizzare come *test set*, mentre tutti gli altri finiranno nella matrice R_e, con e indicante il numero di *trial* escluso dal *train set*.
Ogni elemento t della matrice è formato da un vettore colonna contenente i 16 θ estratti per il *trial* i dall'andamento del giunto g_k, relativi a un movimento del soggetto S di cui fanno riferimento. Si può dire che un elemento della matrice R_e contenga l'andamento del segnale formato dai valori di θ

$$t_i^T = [\theta_1, \theta_2, ..., \theta_{16}] \in \mathbb{R}^{16}, \qquad \forall k \in \{1, ..., 22\} \tag{6.7}$$

con $i \in 1, ..., 6$ escluso il *trial* di *test* e.
La matrice di *training* è formata quindi da un numero di colonne pari al numero di giunti della mano, con l'aggiunta di una contenente l'etichetta associata al movimento m da cui sono stati presi i valori θ. Gli elementi di questa colonna sono formati da un vettore colonna con valori tutti uguali rappresentanti l'etichetta associata al movimento eseguito nel *trial* di a cui fanno riferimento gli altri elementi della riga i

$$m_i^T = [h, h, ..., h] \in \mathbb{R}^{16} \tag{6.8}$$

Per un corretto funzionamento del modello GMM come classificatore è necessario che le classi associate alle righe della matrice di *training* (nel nostro caso rappresen-

tate dalle etichette dei movimenti) appartengano ad un intervallo di numeri interi consecutivi. Dato che l'utilizzo delle GMM viene prima testato con i movimenti 1, 4 e 12, come fatto nel caso dei Classification tree per provare la bontà della scelta tecnica di classificazione, è necessario che le etichette per i 3 movimenti siano rimappati come detto in numeri interi e consecutivi. Per questo motivo, all'interno della matrice di classificazione, le righe contenenti i *trial* relativi al movimento 1 avranno, nell'ultimo elemento, un vettore m_i con $h = 1$, per le righe del movimento 4 avremo un vettore di $h = 2$ mentre per le righe del movimento 12 avremo associato un vettore di $h = 3$.
Le matrici di *test* presentano la stessa struttura descritta per le matrici di *training*. Contengono però solamente i dati provenienti dai *trial* esclusi per i 3 movimenti.

Dopo aver ottenuto un modello GMM allenato con una delle matrici di *training* e per cui è stato impostato un numero di componenti gaussiane pari a 15, questo viene utilizzato per classificare i record della rispettiva matrice di test R_t. In particolare, viene calcolata la probabilità, per ogni riga della matrice R_t, che la sua classe sia una fra le etichette dei movimenti rimappati, ovvero 1, 2 o 3. I risultati ottenuti, però, danno ciascuna probabilità pari a 0, utilizzando il primo pacchetto di codice C++. Quindi, ad ogni record di test, viene attribuito il movimento 1. Ciò è sbagliato, in quanto in ogni matrice di test sono presenti lo stesso numero di record per tutti e 3 i movimenti selezionati.
Gli stessi risultati, cioè probabilità pari a 0, sono stati ottenuti utilizzando il codice del secondo pacchetto C++ per la creazione di modelli GMM, anche provando tutte le altre 5 matrici di *training* e *test* il risultato è stato sempre lo stesso.
Nenache provando diversi valori (quali 3, 10, 17, 20 e 25) per le componenti gaussiane da calcolare nel modello si sono ottenuti risultati migliori.

Provando ad utilizzare le funzioni disponibili in Matlab, invece, sono stati ottenuti risultati leggermente diversi: le probabilità che i record vengano associati ad uno dei 3 movimenti disponibili non sono 0, ma comunque molto piccole, nell'ordine di 10^{-100}. Nel codice C++, questi valori infinitesimali vengono arrotondati a 0, il che spiega i risultati ottenuti precedentemente.

Per scoprire il motivo di questi risultati prossimi allo 0, si è pensato di analizzare i parametri che formano i modelli GMM creati, con particolare attenzione alla matrice contenente le covarianze delle componenti gaussiane calcolate. Qui notiamo che alcuni valori sono molto grandi, nell'ordine di 10^{16}. Una varianza così

elevata spiegherebbe perchè si continuano ad ottenere valori di probabilità sempre prossimi allo 0, per qualunque configurazione tentata finora per la costruzione di modelli GMM.

6.3.1 Problematica degli outlier

La più probabile causa delle elevate varianze riscontrate è la presenza nei dati di *training* e *test* di *outlier*, come quelli mostrati in Figura 6.11. Infatti, i valori elevati per le varianze presentano all'incirca lo stesso ordine (10^{16}) dei valori di *outlier* calcolati per alcuni giunti nei *trial* nella matrice di *training* del modello GMM. Si rende necessario quindi capire cosa, nei dati di ingresso delle DMP, generi la presenza degli *outlier*, dato che non tutti gli andamenti dei θ di un *trial* ne sono soggetti.

Analizzando i segnali che venivano forniti alla funzione *dmptrain* per il calcolo dei θ si è scoperto che, nel caso in cui il punto di arrivo della traiettoria da ricostruire coincidesse con lo stesso punto di partenza (che nel caso dei movimenti dei giunti rigurda i gradi di movimento del giunto), si presentano θ con valori considerati *outlier*. Avere giunti che presentano gli stessi gradi all'istante di partenza e di fine movimento risulta molto probabile, in quanto nell'esecuzione del movimento si parte sempre dalla posizione di riposo, e si torna sempre in essa dopo averlo compiuto.
Nel caso il punto di partenza ed arrivo della traiettoria coincidano si ha l'annullamento del *forcing term* presente nei sistemi dinamici di ognuno dei 3 gradi messi a disposizione dal pacchetto per il calcolo delle DMP. Nella funzione che definisce il *forcing term*, infatti, è presente una moltiplicazione che coinvolge la differenza $(y_0 - y_g)$, con y_0 punto di partenza della traiettoria e y_g punto di arrivo. Mancando il contributo del *forcing term*, il sistema dinamico trova molte difficoltà nel ricopiare la traiettoria di esempio, e compensa tale difficoltà con l'utilizzo di θ elevati per sopperire a tale mancanza.

Capita la problematica, la soluzione applicata al fine di evitare la presenza di *outlier* è stata quella di aumentare di un grado la posizione finale di un giunto, nel caso questa non necessariamente combaci, ma sia molto vicina alla posizione di partenza. Tale soglia di vicinanza è stata posta al valore di 1 grado.

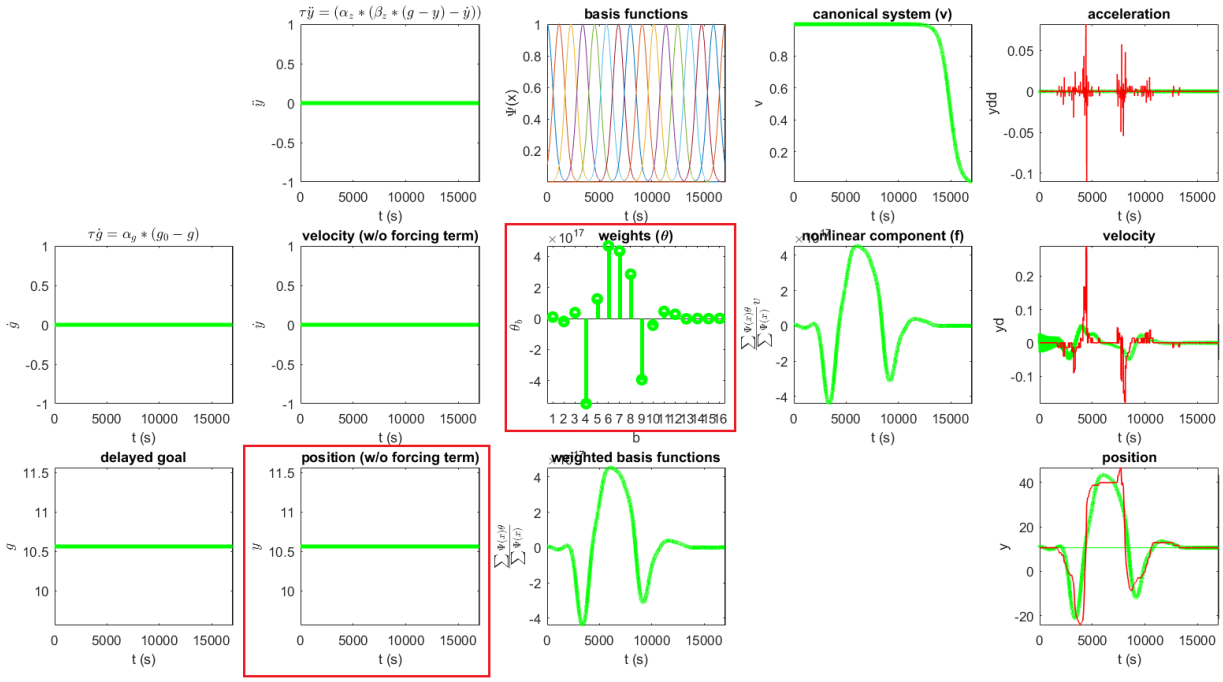

Figura 6.14: Calcolo DMP per il giunto 3 del movimento 4 *trial* 2 fatto dal soggetto 2 con generazione valori θ outlier

Questa modifica non influisce sul processo di classificazione, in quanto viene applicata a tutti i giunti per ogni movimento che ne necessiteranno, ed in futuro sarà applicata anche ai dati provenienti da nuovi *trial* da classificare. Inoltre, se si volessero applicare le traiettorie ricostruite per comandare una mano robotica che esegua il movimento, la differenza di posizione portata da un grado in più rispetto a quella originale non influenzerebbe il buon esito del movimento stesso.

Sono stati ricalcolati quindi i θ per tutti i *trial* di ogni movimento per i primi 10 soggetti, solamente per $n_b = 16$ e con segnali non normalizzati, che precedentemente abbiamo constatato essere la migliore tra le configurazioni provate. Una volta ricalcolate le DMP, sono stati ricostruiti i Classification tree sfruttando questi nuovi dati, ottenendo però solo un leggero miglioramento delle precisioni di classificazione.
La precisione migliore tra i Classification tree costruiti è passata dal 32,3% precedente al 35,3% attuale, ancora comunque troppo bassa per essere considerata soddisfacente.

n_b	**Soggetto di test**	**Livello di *prune***	**Precisione**
16	2	1	35,3 %
16	6	3	35,3 %

Tabella 6.5: Percentuali miglior correttezza per alberi con θ calcolati con soglia a 1 grado

Con le nuove DMP calcolate sono state costruite anche altre matrici di *training* per modelli GMM, per vedere se questo tipo di classificazione abbia risentito maggiormente della correzione del problema dei punti di inizio e fine traiettoria combacianti. Sono stati compiuti gli stessi esperimenti con modelli GMM descritti precedentemente, purtroppo ottenendo gli stessi risultati deludenti. Infatti, anche con questi nuovi dati, le probabilità calcolate per le classi da associare ai record nelle matrici di *test* risultano ancora tutte molto prossime a 0.
Tornando ad analizzare le matrici di covarianza delle componenti gaussiane, si nota che il problema di valori *outlier* risulta essere ancora presente, anche se notevolmente ridimensionato, dato che ora i valori fuori scala risultano essere dell'ordine di 10^4. I valori *outlier* continuano comunque a sovrastare gli altri, riducendo di molto la loro importanza in fase di classificazione, per entrambi i metodi testati finora, continuando a realizzare risultati di classificazione con precisione troppo bassa e quindi non soddisfacente.

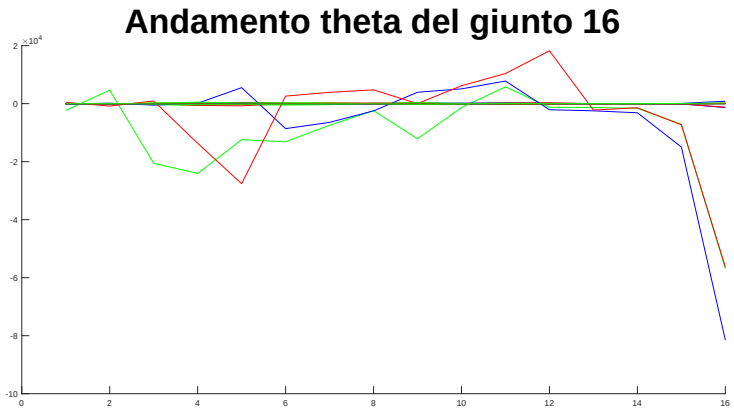

Figura 6.15: Andamento θ per il giunto 16 dei movimenti 1, 4 e 12 compiuti dal soggetto 2

6.4 ECOC con DMP

Constatato che Classificatio tree e GMM non si sono rivelati buoni metodi di classificazione per questa tipologia di dati, neanche dopo la risoluzione del problema legato alla coincidenza del punto iniziale e finale della traiettoria da copiare che generava valori elevati di θ *outlier*, si è provato ad utilizzare una nuova metodologia: i classificatori multiclasse ECOC (sezione 4.3).
Come tipologia di *learners* sono stati provati i seguenti:

- lineare;
- k-NN con $k = 5$ e $k = 10$ vicini come riferimento di classificazione;
- *Naive-Bayes*;
- SVM con kernel a tipologia:
  - lineare;
  - gaussiana;
  - polinomiale di grado 3;
- analisi discriminante.

Il funzionamento di ogni *learners* elencato è specificato al paragrafo 4.3.1.
Ognuno di questi è stato testato configurando un *coding design* prima di tipo *one-versus-one*, e poi di tipo *one-versus-all*.
Come *training set* e *test set* sono dapprima stati provati gli stessi forniti per gli alberi di classificazione, nei quali ogni record è formato da $16 * 22 = 352$ variabili (i θ di ogni giunto), e le classi da considerare sono 17, ovvero una per ogni possibile movimento tra tutti quelli disponibili nel dataset NinaPro. In particolare, sono stati provati i *training* e *test set* riguardanti l'utilizzo del soggetto 1 come soggetto di test. Le precisioni delle classificazioni ottenute sono riassunte nella tabella seguente.

Alcuni *learners* si sono rivelati migliori di altri (k-NN e analisi discriminante), anche se le precisioni di classificazione migliori rimangono in linea con i risultati

	Precisione	
Learners	***one-versus-one***	***one-versus-all***
Linear	34,3 %	28,4 %
k-NN $k = 5$	27,5 %	27,5 %
k-NN $k = 10$	21,6 %	38,2 %
Naive-Bayes	35,3 %	17,7 %
SVM	5,9 %	5,9 %
SVM kernel gaussiano	5,9 %	5,9 %
SVM kernel polinomiale	5,9 %	5,9 %
Analisi Discriminante	24,5 %	35,3 %

Tabella 6.6: Percentuali precisione modello ECOC con vari *Learners* per input dei Classification tree

ottenuti per i Classification tree. Prima di abbandonare anche la strada dei classificatori ECOC, però, questi sono stati provati anche utilizzando come input le matrici di *training* e di test realizzate per la costruzione dei modelli GMM descritte al punto 6.6.
In questo caso venivano considerati solamente i dati provenienti dai movimenti 1, 4 e 12 (classificati rispettivamente come 1,2, e 3) disposti in record con 22 variabili. Per identificare un movimento, invece, servono 16 di questi record, data la disposizione in verticale dei θ provenienti dalle DMP di ogni giunto. I *trial* di un movimento vengono quindi rappresentati da 16 record (anche quelli nella matrice di *test*), e si dispone di 16 valutazioni per la classe di un *trial* di movimento. Per classificare un nuovo movimento, si devono aggregare queste 16 classificazioni: il metodo più semplice è associare al movimento la classe che viene predetta con maggiore frequenza nei 16 valori forniti dal classificatore, come se ognuno rappresentasse un voto per una classe, per ogni nuovo *trial*. Chiamiamo questo metodo classificazione a votazione per praticità.
Le due tabelle seguenti riassumono i risultati di precisione ottenuti considerando prima i record singolarmente, e poi classificandoli con la classe presente in maggioranza fra i 16 risultati di ogni *trial*:

	Precisione	
Learners	***one-versus-one***	***one-versus-all***
Linear	40,6 %	36 %
k-NN $k = 5$	42,9 %	42,9 %
k-NN $k = 10$	50,4 %	48, 5 %
Naive-Bayes	37,5 %	37,1 %
SVM	30 %	35,8 %
SVM kernel gaussiano	33,3 %	33,1 %
SVM kernel polinomiale	35 %	32,9 %
Analisi Discriminante	46,9 %	46,7 %

Tabella 6.7: Percentuali precisione modello ECOC con vari *Learners* per input GMM

	Precisione votazione	
Learners	***one-versus-one***	***one-versus-all***
Linear	40 %	36,7 %
k-NN $k = 5$	40 %	40 %
k-NN $k = 10$	50 %	60 %
Naive-Bayes	33,3 %	33,3 %
SVM	33,3 %	30 %
SVM kernel gaussiano	33,3 %	33,3 %
SVM kernel polinomiale	30 %	33,3 %
Analisi Discriminante	46,7 %	43,3 %

Tabella 6.8: Percentuali precisione a votazione modello ECOC con vari *Learners* per input GMM

La classificazione a votazione ha prodotto i risultati migliori riscontrati finora. Il livello di precisione delle classificazioni non risulta essere ancora abbastanza elevato da poter essere considerato soddisfacente, anche considerando che per gli input costruiti per i modelli GMM vengono utilizzate soltanto le DMP prese da 3 movimenti scelti diversi tra loro.

6.5 Ri-campionamento del segnale

Si è reso necessario trovare un metodo per migliorare ulteriormente i dati provenienti dalle DMP, al fine di rendere i θ provenienti da *trial* di diversi movimenti

più differenti tra loro, e quelli provenienti da *trial* di uno stesso movimento più somiglianti per facilitare la classificazione.
Una possibile miglioria ai dati potrebbe arrivare utilizzando la funzione *resample* di Matlab applicata ai segnali dei giunti provenienti dal guanto, prima che questi vengano forniti alla funzione *dmptrain* per il calcolo delle DMP. La funzione *resample(X, resize, size)* non fa altro che ri-campionare il segnale presente nella variabile X formato da *size* campioni, ottenendone uno formato da solamente *resize* campionature (dunque $resize < size$). Inoltre *resample* applica ad X un filtro *antialiasing* passa-basso di tipo FIR e ne compensa il ritardo introdotto prima di procedere al ri-campionamento.
I segnali dei giunti originali sono formati da circa 16000 campioni ciascuno: per il *resample* è stato impostato un valore di *resize* pari a 2001.

I θ calcolati dai segnali ri-campionati si sono rivelati essere molto buoni: gli *outlier* sono quasi del tutto assenti, anche perchè lo spostamento del punto di fine traiettoria di 1 grado è stato comunque applicato, e quando presenti assumenvano valori ancora minori di quelli ottenuti nei tentativi precedenti. Inoltre si è arrivati a compimento dell'obiettivo prefissato: per *trial* provenienti dallo stesso movimento i valori dei θ sono risultati essere molto simili tra loro. Ciò si nota facilmente dai grafici degli andamenti dei θ: adesso, anche visivamente, si riescono a distinguere i θ estratti da movimenti diversi.

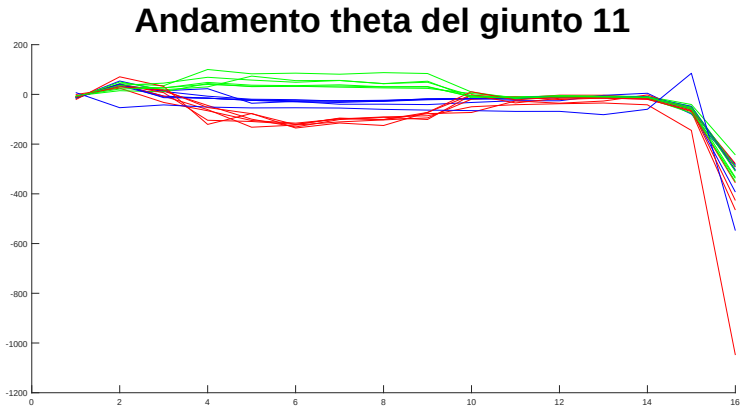

Figura 6.16: Andamento θ per il giunto 11 dei movimenti 1 (rosso), 4 (verde), e 12 (blu) con segnali *resample*

Ottenuti nuovi valori di θ per tutti i primi 10 soggetti, questi vengono utilizzati per costruire altri Classifcation tree, come fatto nei tentativi descritti precedentemente. Le precentuali di classificazioni corrette risultano di molto superiori per ognuno dei soggetti utilizzati come test, e vanno da un minimo del 45,1% ad un massimo del 66,7%. I risultati migliori sono stati ottenuti in questi casi:

n_b	**Soggetto di test**	**Livello di *prune***	**Precisione**
16	6	3	64,7 %
16	10	2	66,7 %

Tabella 6.9: Percentuali miglior correttezza per alberi con θ calcolati con segnali *resample*

Dato il miglioramento ottenuto nelle precisioni dei Classification tree, i θ ottenuti dai segnali dopo il *resample* sono sati utilizzati anche per la creazione di nuove matrici di *training* e *test* per modelli GMM, per vedere se stavolta producevano valori di probabilità diversi da 0, in modo da poter utilizzare il modello GMM come classificatore.
Nonstante la bontà dei nuovi θ i modelli GMM costruiti grazie ad essi continuano a presentare le stesse problematiche riscontrate nei casi precedenti, sia nelle probabilità calcolate che nei valori elevati presenti nelle covarianze delle componenti gaussiane, anche se minori rispetto a quelli dei casi precedenti. Tale modello di classificazione, pertanto, non si pone come utilizzabile per la classificazione di movimenti della mano sfruttando le informazioni provenienti dalle loro DMP.

Come ultimo tentativo, i nuovi valori di θ sono stati utilizzati nei classificatori ECOC, ricostruendo tutti quelli già provati precedentemente, per vedere se qualcuno raggiungesse dei valoi di accuratezza massima non solamente buoni, ma anche soddisfacenti.
Per prima cosa, i θ vengono disposti come gli input dei Classification tree, ottenendo i risultati presenti nella tabella seguente.

Anche in questo caso otteniamo buoni risultati, superiori a quelli ottenuti per i rispettivi modelli ECOC precedentemente creati, ed ancora in linea con quelli riscontrati per Classification tree che hanno utilizzato come input i θ calcolati dai segnali *resample*.

	Precisione	
Learners	***one-versus-one***	***one-versus-all***
Linear	33,3 %	39,2 %
k-NN $k = 5$	28,4 %	28,4 %
k-NN $k = 10$	59,8 %	62,8 %
Naive-Bayes	44,1 %	19,6 %
SVM	6,9 %	5,9 %
SVM kernel gaussiano	5,9 %	5,9 %
SVM kernel polinomiale	5,9 %	5,9 %
Analisi Discriminante	55,9 %	63,7 %

Tabella 6.10: Percentuali precisione modello ECOC con vari *Learners* per input dei Classification tree con resize dei segnali

Vengono infine costruiti i classificatori ECOC utilizzando come input i nuovi θ nella disposizione necessaria alle GMM, calcolando non solo la precisione per i singoli record ma anche quella ottenuta dalla classificazione a votazione, ottenendo i seguenti risultati:

	Precisione	
Learners	***one-versus-one***	***one-versus-all***
Linear	56 %	54,4 %
k-NN $k = 5$	42,3 %	42,3 %
k-NN $k = 10$	48,1 %	49,4 %
Naive-Bayes	33,1 %	33,1 %
SVM	45,6 %	40,6 %
SVM kernel gaussiano	33,3 %	33,3 %
SVM kernel polinomiale	33,8 %	33,1 %
Analisi Discriminante	57,7 %	59,2 %

Tabella 6.11: Percentuali precisione modello ECOC con vari *Learners* per input GMM con resize dei segnali

	Precisione votazione	
Learners	***one-versus-one***	***one-versus-all***
Linear	76,7 %	63,3 %
k-NN $k = 5$	50 %	50 %
k-NN $k = 10$	63,3 %	66,7 %
Naive-Bayes	33,3 %	33,3 %
SVM	40 %	43,3 %
SVM kernel gaussiano	33,3 %	33,3 %
SVM kernel polinomiale	33,3 %	33,3 %
Analisi Discriminante	76,7 %	86,7 %

Tabella 6.12: Percentuali precisione a votazione modello ECOC con vari *Learners* per input GMM con resize dei segnali

Si notano anche stavolta valori superiori per ogni classificatore ECOC confrontandoli con i corrispettivi dei casi precedenti. Inoltre, per le classificazioni a votazione, abbiamo quasi sempre valori elevati di precisione, fino ad ottenerne una massima pari a 86,7% per il *learner* analisi discriminante ed un *coding design* *one-versus-all*, che possiamo definire soddisfacente.

Capitolo 7

Conclusioni

In questo lavoro di tesi ci siamo concentrati sulla creazione di un metodo efficace per la classificazione di movimenti della mano. Si è scelto di utilizzare le DMP per rappresentare i movimenti di 22 giunti della mano, per riconoscere un insieme di movimenti eseguiti da persone diverse. L'uso delle DMP è diffuso in letteratura per rappresentare il movimento dei giunti di un robot durante l'esecuzione di un *task*, ma esistono pochi esempi applicati ad un così elevato numero di gradi di libertà. Inoltre, le DMP hanno dimostrato la loro efficacia nel generalizzare movimenti eseguiti da più soggetti distinti, e questo lo rende un candidato ideale per questo tipo di studio. Lo scopo della classicazione dei movimenti della mano è quello di riuscire a riconoscere la classe di appartenza di un movimento eseguito da un soggetto estraneo allo studio.

I primi test hanno preso in considerazione l'uso diretto dei dati forniti dal calcolo delle DMP su ogni singolo giunto della mano, raggruppandoli per soggetti diversi e cercando di distinguerli per tipo di movimento. I dati sono stati classificati utilizzando diverse tecniche: Classification Tree, GMM, e modelli ECOC. In tutti questi casi sono stati ottenuti risultati poco soddisfacenti in termini di percentuali di classificazione. Questi risultati hanno evidenziato che le problematiche riscontrate riguardavano più i dati presi in considerazione, che le tecniche di classificazione applicate. I test successivi, perciò, si sono concentrati sul comprendere quali fossero le cause che falsavano le caratteristiche dei modelli di classificazione. In particolare, sono stati evidenziati dei picchi di segnale in corrispondenza delle dimostrazioni in cui il punto di partenza del movimento coincideva in maniera sostanziale con la posa conclusiva in almeno uno dei giunti. Questi picchi, anche di 10 ordini di grandezza più elevati rispetto al normale andamento del segnale,

andavano ad appiattire le differenze tra le diverse classi di movimento, con una conseguente scarsa capacità di classificazione. Per risolvere l'incoveniente è stato introdotto un piccolo scostamento in uno degli estremi. Oltre a questo accorgimento si è deciso di introdurre anche un ri-campionamento dei dati elaborati dovuto ad alcune limitazioni nel numero di campioni fissate negli algoritmi presi in considerazione per il calcolo delle DMP. Dopo aver applicato questi accorgimenti, nuovi test basati sugli stessi classificatori, hanno fornito risultati significativamente migliori, lasciando però spazio ad un ulteriore margine di miglioramento.

I risultati migliori in assoluto sono stati ottenuti tramite l'utilizzo di modelli ECOC con tipologia di *learners* ad analisi discriminante ed una tipologia di *coding design one-versus-all*, passando come input i valori θ delle DMP ed utilizzando una classificazione a votazione. In queste condizioni, la precisioni delle classificazioni effettuate tocca 86,7% di correttezza, un valore alto che si può considerare soddisfacente. Con i Classification tree, invece, si sono toccate punte di correttezza nelle precisioni pari al 66,7%, altro buon risultato.

Esistono tuttavia delle differenze da far notare riguardo i due risultati citati: nei Classification tree vengono impiegati tutti i *trial* di tutti e 17 i movimenti presenti nel *dataset*. Questo va pesato nella scelta di quale dei metodi di classificazione risulti il migliore, poichè è vero che i modelli ECOC hanno generato precisioni migliori, ma utilizzando solamente 3 movimenti diversi tra loro.
Inoltre, anche lo stile di classificazione è differente, come si può notare anche dalla struttura dei dati in input

- per i Classification tree i record da classificare sono nella forma $r_i = [\theta_{G1_1}, ..., \theta_{G1_{16}}, \theta_{G2_1}, ..., \theta_{G2_{16}}, \theta_{G22_1}, ..., \theta_{G22_{16}}, c_i]$, le cui variabili formate dai valori dei θ prelevati da ogni giunto fotografano interamente le caratteristiche di un *trial* di un movimento ;

- per i modelli ECOC ogni *trial* è rappresentato da 16 record nella forma $r_i = [\theta_{G1_i}, \theta_{G2_i}, ..., \theta_{G22_i}, c_i], \quad \forall i = 1, ..., 16$, ognuno dei quali fotografa solo una parte del *trial*, fornendone una classificazione di tipo *online*. Avendo più classificazioni basate solo sui θ scoperti man mano, si rende necessario poi estrarne una unica tramite il metodo della votazione, che condensa tutte le classificazioni ottenute in un unico risultato finale. Dall'altra parte però, questa tecnica fornisce un risultato ad ogni istante dell'esecuzione del movimento, anche senza conoscere la futura evoluzione del segnale. Questa

situazione consentirebbe al sistema di elaborazione di reagire più in fretta per andare incontro alle azioni svolte dall'utente.

Queste considerazioni ci fanno supporre che entrambi i metodi di classificazione possano essere delle buone scelte, da approfondire in futuro per poter decidere quale delle due sia effettivamente la migliore per la tipologia di rappresentazione di movimenti tramite DMP.

7.1 Sviluppi futuri

Il lavoro svolto è servito principalmente per trovare delle buone strade da intraprendere per poter classificare movimenti della mano utilizzando un numero elevato di informazioni, provenienti dai dati catturati tramite il guanto *Cyberglove2*, data la scarsità di studi specifici presenti nel panorama scientifico, soprattutto legato all'utilizzo delle DMP come *features* caratterizzanti del movimento. I risultati ottenuti potranno essere utilizzati come punto di partenza per studi futuri. In particolar modo, la risoluzione dei problemi legati alla presenza di *outlier* e alle esigenze di ri-campionamento dei segnali, possono ridurre in maniera significativa l'*overhead* necessario per eleaborare correttamente i segnali utilizzando le DMP.

Una naturale prosecuzione potrebbe essere quella legata all'utilizzo delle informazioni provenienti da tutti e 40 i soggetti per proseguire l'analisi iniziata durante questo elaborato, in cui si prendendo in considerazione i movimenti eseguiti dai primi 10 soggetti. Utilizzandoli tutti si riuscirebbe ad ottenere classificatori potenzialmente superiori nei risultati, data la presenza di movimenti provenienti da un insieme più variegato di persone. Per completezza si potrebbe anche utilizzare le migliori tecniche di classificazioni analizzate con i movimenti presenti nei restanti due *dataset* del progetto NinaPro, nei quali ci sono movimenti di tipologie differenti rispetto a quelli considerati finora, per capire quanto la bontà dei classificatori ottenuti vari a seconda della tipologia dei movimenti da classificare.

Oltretutto si potrebbe indagare a fondo sul motivo per il quale la costruzione di modelli GMM per classificare i movimenti rappresentati dalle DMP forniscano sempre probabilità molto prossime a 0 quando si cerca di ottenere la classificazione di un nuovo movimento. La risoluzione dei problemi legati agli *outlier* e la tecnica di ri-campionamento, che molti miglioramenti hanno portato sia per i Classification tree che per i modelli ECOC, non hanno invece portato benefici per questa

tipologia di classificazione.
La scarsità dei risultati ottenuti potrebbe anche essere dovuta alla tipologia dei dati forniti come ingresso, dato che, in altre situazioni di classificazioni di movimenti più generali e non specificatamente della mano, i modelli GMM si sono spesso rivelati una soluzione vincente.

Si potrebbe inoltre continuare a testare la potenza di classificazione di movimenti della mano tramite DMP testando ulteriori modelli di classificazione, non inseriti in questo lavoro per mancanza di tempo. Una possibile scelta sarebbe l'utilizzo di reti neurali, già citate in altri lavori relativi alle DMP, o di alcuni classificatori qui usati come *learners* binari per i modelli ECOC, ma che anche presi da soli possono risolvere problemi di classificazione multiclasse (come ad esempio k-NN o analisi discriminante).

Infine si potrebbe provare ad utilizzare gli stessi modelli di classificazione provati durante lo svolgimento della tesi, ma fornendo come ingresso rappresentazioni dei movimenti tramite features diverse rispetto ai θ calcolati dalle DMP sui giunti della mano. In entrambe queste ultime due idee di possibili sviluppi futuri si ha la possibilità di considerare i risultati presentati in questo lavoro come mezzo di confronto per la valutazione dei possibili risultati ottenuti. Il confronto dei risultati risulterebbe inoltre più significativo continuando ad utilizzare i dati forniti nei *dataset* del progetto NinaPro anche in progetti futuri.

Bibliografia

[1] R. Mao, Y. Zhang, and Y Aloimonos. *Learning Hand Movement by Markerless Demonstration for Humanoid Robot Task*.

[2] S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert. *Control, Planning, Learning, and Imitation with Dynamic Movement Primitives*. IEEE International Conference on Intelligent Robots and Systems, 2003.

[3] J. Kober, and J. Peters. *Learning Motor Primitives for Robotics*. IEEE International Conference on Robotics and Automation (ICRA), 2009.

[4] O. C. Jenkins, M. J. Matarić, and S. Weber. *Primitive-Based Movement Classification for Humanoid Imitation*. IEEE International Conference on Humanoid Robotics, 2000.

[5] A. Sant'Anna, W. Ourique de Morais, and N. Wickström *Gait Unsteadiness Analysis from Motion Primitives*. Gerontechnology 2008.

[6] A. Fod, M. J. Matarić, and O. C. Jenkins. *Automated Derivation of Primitives for Movement Classification*. Autonomous Robots, vol. 12, issue 1, pp. 39-54, 2002.

[7] M. H. Alomari, A. Samaha, and K. Alkamha. *Automated Classification of L/R Hand Movement EEG Signals using Advanced Feature Extraction and Machine Learning*. International Journal of Advanced Computer Science and Applications, vol. 4, no. 6, 2013

[8] A. Gijsberts, M. Atzori, and C. Castellini. *Movement Error Rate for Evaluation of Machine Learning Methods for sEMG-Based Hand Movement Classification*. IEEE Transactions on Neural Systems and Rehabilitation Engineering, vol. 22, issue 4, 2014.

[9] M. Pla Mobarak, R. Munoz Guerrero, and J. M. Gutierrez Salgado. *Hand movement classification using transient state analysis of surface multichannel EMG signal*. Pan American Health Care Exchanges (PAHCE), 2014.

[10] X. Zhai, B. Jelfs, R. H. Chan, and C. Tin. *Short latency hand movement classification based on surface EMG spectrogram with PCA*. IEEE Engineering in Medicine and Biology Society, 2016.

[11] F. Stival, S. Michieletto, and E. Pagello *Online subject-independent modeling of sEMG signals for the motion of a single robot joint*. 6th IEEE International Conference on Biomedical Robotics and Biomechatronics (BioRob), 2016.

[12] F. Stival, S. Michieletto, and E. Pagello. *Subject-Independent Modeling of sEMG Signals for the Motion of a Single Robot Joint*. Workshop of Robotics: Science and Systems 2015 on Combining AI Reasoning and Cognitive Science with Robotics, 2015.

[13] R. Valentini, S. Michieletto, F. Spolaor, Z. Sawacha, and E. Pagello. *Processing of sEMG signals for online motion of a single robot joint through GMM modelization*. IEEE International Conference on Rehabilitation Robotics (ICORR), 2015.

[14] S. Michieletto, L. Tonin, M. Antonello, R. Bortoletto, F. Spolaor, E. Pagello and E. Menegatti. *GMM-based Single-joint Angle Estimation using EMG signals*. Advances in Intelligent Systems and Computing - IAS13 Conference, 2014.

[15] R. Bortoletto, S. Michieletto, E. Pagello and D. Piovesan. *Human Muscle-Tendon Stiffness Estimation during Normal Gait Cycle based on Gaussian Mixture Model*. Advances in Intelligent Systems and Computing - IAS13 Conference, 2014.

[16] C. R. Wylie. *Advanced Engineering Mathematics (4th ed.)*. McGraw-Hill, pp. 61-65, 1975.

[17] S. Schaal. *Dynamic Movement Primitives - A Framework for Motor Control in Humans and Humanoid Robotics*. Adaptive Motion of Animals and Machines, pp 261-280.

[18] A. J. Ijspeert, J. Nakanishi, and S. Schaal. *Movement imitation with nonlinear dynamical systems in humanoid robots*. IEEE International Conference on Robotics and Automation (ICRA), 2002

[19] A. J. Ijspeert, J. Nakanishi, P. Pastor, H. Hoffmann, and S. Schaal. *Dynamical Movement Primitives: Learning attractor models for motor behaviors*. Neural Computation, vol. 25, no. 2, pp. 328-373, 2013.

[20] S. Schaal, P. Mohajerian, and A. J. Ijspeert. *Dynamics systems vs. optimal control a unifying view*. Progress in Brain Research, vol. 165, no. 1, pp. 425-445, 2007.

[21] S. Vijayakumar, and S. Schaal. *Locally weighted projection regression: An O(n) algorithm for incremental real time learning in high dimensional spaces*. Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), vol 1, Stanford, CA, pp. 288-293, 2000.

[22] D. R. Jones, M. Schonlau, and W. J. Welch. *Efficient Global Optimization of Expensive Black-Box Functions*. Journal of Global Optimization, vol. 13, pp. 455-492, 1998.

[23] T. Kulvicius, K. Ning, and M. Tamosiunaite. *Joining Movement Sequences: Modified Dynamic Movement Primitives for Robotics Applications Exemplified on Handwriting*. IEEE Transactions on Robotics, vol. 28, pp. 145-157, 2012.

[24] M. Atzori, A. Gijsberts, C. Castellini, B. Caputo, A. G. Mittaz Hager, S. Elsig, G. Giatsidis, F. Bassetto, and H. Müller. *Electromyography data for non-invasive naturally-controlled robotic hand prostheses*. Scientific Data, 2014

[25] M. Atzori, A. Gijsberts, S. Heynen, A. M. Hager, O. Deriaz, P. van der Smagt, C. Castellini, B. Caputo, and H. Müller. *Building the NINAPRO Database: A Resource for the Biorobotics Community*.

[26] A. Gijsberts, M. Atzori, C. Castellini, H. Müller, and B. Caputo. *Measuring Movement Classification Performance with the Movement Error Rate*. IEEE Transactions on neural systems and rehabiliation engineering, 2014

[27] S. Escalera, O. Pujol, and P. Radeva. *Separability of ternary codes for sparse designs of error-correcting output codes*. Pattern Recog. Lett., vol. 30, issue 3, pp. 285-297, 2009.

[28] S. Escalera, O. Pujol, and P. Radeva. *On the decoding process in ternary error-correcting output codes*. IEEE Transactions on Pattern Analysis and Machine Intelligence. vol. 32, issue 7, pp. 120-134, 2010.

[29] L. Bottou *Online Algorithms and Stochastic Approximations*. Online Learning and Neural Networks, Cambridge University Press 1998.

[30] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Second Edition N.Y. Springer, 2008.

[31] N. Christianini, and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press, Cambridge, UK, 2000.

[32] R. A. Fisher. *The Use of Multiple Measurements in Taxonomic Problems*. Annals of Eugenics, vol. 7, pp. 179-188, 1936.

[33] S. J. Orfanidis. *Optimum Signal Processing: An Introduction. 2nd Edition*. McGraw-Hill, 1996.

[34] J. Larsen. *Correlation Functions and Power Spectra*. 2009.

[35] J. R. Buck, M. M. Daniel, and A. C. Singer. *Computer Explorations in Signals and Systems Using MATLAB. 2nd Edition*. Upper Saddle River, NJ: Prentice Hall, 2002.

[36] P. Stoica, and R. Moses. *Spectral Analysis of Signals*. Upper Saddle River, NJ: Prentice Hall, 2005.

[37] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Boca Raton, FL: CRC Press, 1984.

[38] D. Coppersmith, S. J. Hong, and J. R. M. Hosking. *Partitioning Nominal Attributes in Decision Trees*. Data Mining and Knowledge Discovery, vol. 3, pp. 197-217, 1999.

[39] W.Y. Loh, and Y.S. Shih. *Split Selection Methods for Classification Trees*. Statistica Sinica, vol. 7, pp. 815-840, 1997.

[40] A. Savitzky, and M.J.E. Golay. *Smoothing and Differentiation of Data by Simplified Least Squares Procedures*. Analytical Chemistry, vol. 36, pp. 1627-1639, 1964.

[41] A. P. Dempster, N. M. Laird, and D. B. Rubin. *Maximum-Likelihood from incomplete data via the EM algorithm*. Journal of the Royal Statistical Society, 1977.