UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INFORMATICA

Tesi di Laurea Magistrale in Informatica

Automatic fMRI Learning - AfL

Relatori

Prof. Roberto Tagliaferri

Prof. Fabrizio Esposito

Relatore esterno

Dr. Giancarlo Valente

Candidato

Simone Romano

Matr. 0522500294

ANNO ACCADEMICO 2015/2016

Ringraziamenti

Non avevo mai attraversato un cammino così duro e stimolante. Tutte le difficoltà si sono a me poste come un bel gioco. Ho sempre combattuto per vincere. Quanti fatti, persone, parole, pensieri, tutti a fare da contorno a questa indimenticabile parentesi di vita. Tra le persone, c'è sicuramente la mia compagna di vita. Lei ha sempre creduto in me e gioito per i miei traguardi, regalandomi forza e coraggio. C'è la mia famiglia; lei mi ha dato molto più di quanto si possa concretamente apprezzare. Ci sono gli amici, quelli che hanno giocato con me. C'è la fortuna, quella che ha sempre fatto da sottofondo alla mia strategia di gioco.

Non mi sento ancora un vincitore. Il vincitore ha già finito; a me piace pensare di essere ad un nuovo inizio. È da qui che voglio cominciare a studiare per l'esame più importante: la vita.

Indice

Elenco delle figure

Elenco degli Algoritmi

3 L'algoritmo descritto seleziona, a partire dal vettore di assegnamento dei punti ai vari cluster, k punti (se k è il numero di cluster) come rappresentanti dei vari cluster.

Abstract

Il lavoro di tesi presenta lo studio di un esperimento fMRI tramite tecniche di machine learning. I partecipanti dell'esperimento erano invitati a premere un pulsante con il dito indice o anulare, a seconda delle istruzioni visive che venivano fornite loro. La domanda a cui si vuole rispondere è la seguente: "esiste una differenza tra le risposte del cervello a queste due condizioni?".

L'intero lavoro mostra come algoritmi di machine learning sono utilizzati per rispondere a questa domanda. In particolare sono state confrontate diverse tecniche per ridurre le dimensionalità dei dati (PCA, clustering, features selection). In seguito alla riduzione delle dimensionalità, sono state valutate le componenti selezionate rappresentando la risposta emodinamica (HRF - hemodynamic response function) e le mappe di attivazione. Infine, è stato addestrato un classificatore SVM (lineare e non lineare) tramite cross-validation per classificare i pattern.

Capitolo 1

Introduzione

Il lavoro di tesi proposto si colloca nell'ambito delle neuroscienze, ovvero l'insieme degli studi condotti dalla comunità scientifica sul sistema nervoso. Diverse sono le aree scientifiche coinvolte nelle neuroscienze e vanno dall'area chimica, ingegneristica, sociale, linguistica, medica a quella informatica. Gli approcci più recenti utilizzati dai neuroscienziati consentono di studiare il funzionamento complessivo del cervello tramite l'utilizzo di tecniche di neuroimaging funzionale. Tali tecnologie rendono possibile l'analisi e lo studio delle relazioni che intercorrono tra l'attività di determinate aree cerebrali e specifiche funzioni cerebrali.

La risonanza magnetica funzionale (fMRI) è una tecnica in grado di visualizzare la risposta emodinamica (cambiamenti nel contenuto di ossigeno dei capillari) correlata all'attività neuronale del cervello. Quando eseguiamo un compito (movimento di una mano, visualizzazione di una immagine, lettura di una parola) vengono reclutate spe-

cifiche aree cerebrali; è in queste aree che avviene il consumo maggiore di ossigeno. Questa variazione di ossigeno viene rilevata dal segnale di risonanza magnetica e tradotta in immagini utilizzabili in pratica.

1.1 Machine learning ed fMRI

Interpretare le immagini cerebrali ottenute con tecniche di neuroimaging funzionale (es. fMRI) richiede una analisi molto complessa. Le domande a cui si vuole risponedere sono solitamente: "in quale zona cerebrale è concentrata una certa informazione" e "come è caratterizzata l'informazione". Gli approcci più diffusi utilizzano algoritmi di machine learning per addestrare classificatori allo scopo di individuare regioni cerebrali legate a particolari stimoli o stati mentali.

Il goal dell'analisi di dati fMRI è individuare quale parte del cervello mostra una maggiore attività negli istanti temporali in cui è applicato lo stimolo. Le immagini su cui si lavora forniscono una rappresentazione volumetrica del cervello nel tempo, fornendo per ogni voxel (unità base) un valore di attivazione [1] (vedi figura 1.1.1).

È possibile distinguere due principali metodologie di analisi di dati di risonanza magnetica funzionale:

1. Analisi univariate

2. Analisi multivariate - (Multi-voxel pattern analysis - MVPA) [2]

La prima categoria studia ogni voxel singolarmente, andando ad individuare i voxel che rispondono maggiormente ad un particolare esperimento. Una delle tecniche univariate più utilizzate è sicuramente il GLM (General Linear Model).

La seconda categoria studia il comportamento di insiemi di voxel, andando a valutare anche relazioni spaziali tra gruppi di voxel.

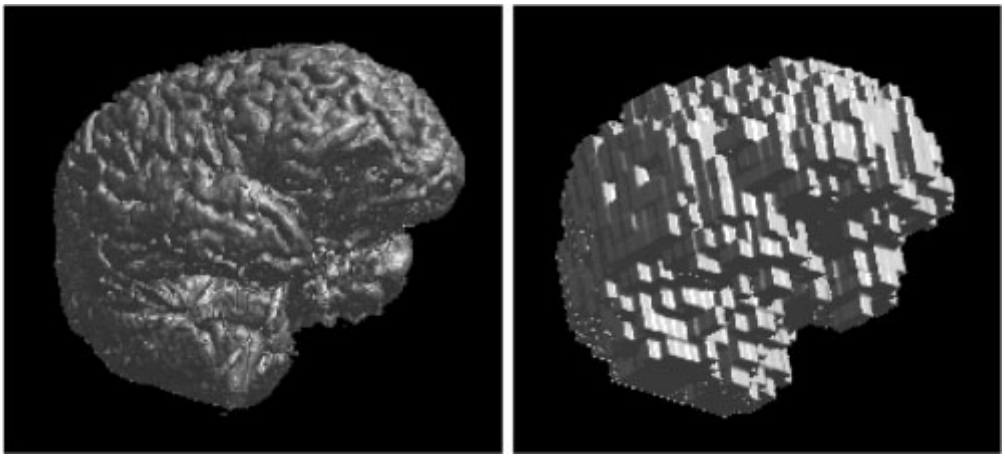

Figura 1.1.1: L'immagine mostra una ricostruzione 3D del cervello in due diverse risoluzioni: sulla sinistra l'immagine a più alta risoluzione con voxel più piccoli (1mm x 1mm x 1.5mm); sulla destra una immagine a risoluzione più bassa con voxel facilmente visibili (7mm x 7mm x 10mm).

1.2 Analisi multivariate in fMRI

La tipologia di analisi multivariate consente di individuare in maniera più affidabile, rispetto alle tecniche univariate, differenze tra diverse condizioni sperimentali. Si parla di multi-voxel pattern analysis (MVPA) in contesti di applicazioni di "brain reading", dove uno specifico stato della mente può essere individuato in un pattern fMRI dopo una prima fase di leraning.

I metodi MVPA tradizionali utilizzano tecniche di pattern classification dove i pattern da classificare sono tipicamente vettori contenenti valori di attività di voxel. Quattro sono gli step fondamentali di una analisi MVPA [2] (vedi figura 1.2.1):

1. features selection
2. pattern assembly
3. classifier training
4. generalization testing

La prima fase prevede la ricerca delle features che saranno utilizzate per la classificazione (una generica feature rappresenta il valore di un voxel nel tempo). I voxel selezionati nella prima fase sono dunque assemblati in una unica matrice e concatenati ad un vettore di label in accordo con le condizioni sperimentali. I pattern così creati vengono divisi in training-set e testing-set.

A questo punto i dati di training vengono utilizzati per addestrare un classificatore (es. SVM) in modo da ottenere un modello che rappresenta un mapping tra pattern e condizione sperimentale. Il modello

generato viene utilizzato con i dati di testing per valutare la capacità di generalizzazione su dati nuovi.

1.3 Automatic fMRI Learning - AfL

Il lavoro proposto, AfL (Automatic fMRI Learning), è un framework scritto in MATLAB per analisi multivariate di dati fMRI. Partendo da dati fMRI preprocessati e dalla descrizione della condizione sperimentale, AfL esegue una serie di analisi generando in output un modello con una buona capacità di generalizzazione su nuovi dati.

La elaborazione fatta sui dati in input è la seguente:

- compressione delle serie temporali
- clustering dei voxel
- generazione mappe dei cluster
- stima della risposta emodinamica
- addestramento di un modello in cross-validation
- analisi dei risultati

Partendo dunque da segnali di voxel nel tempo, il framework effettua una prima compressione di questi segnali andando a ridurre il numero di osservazioni per condizione sperimentale.

A questo punto i voxel vengono raggruppati per similarità con due diversi algoritmi di clustering; in questo modo si ottiene un partizio-

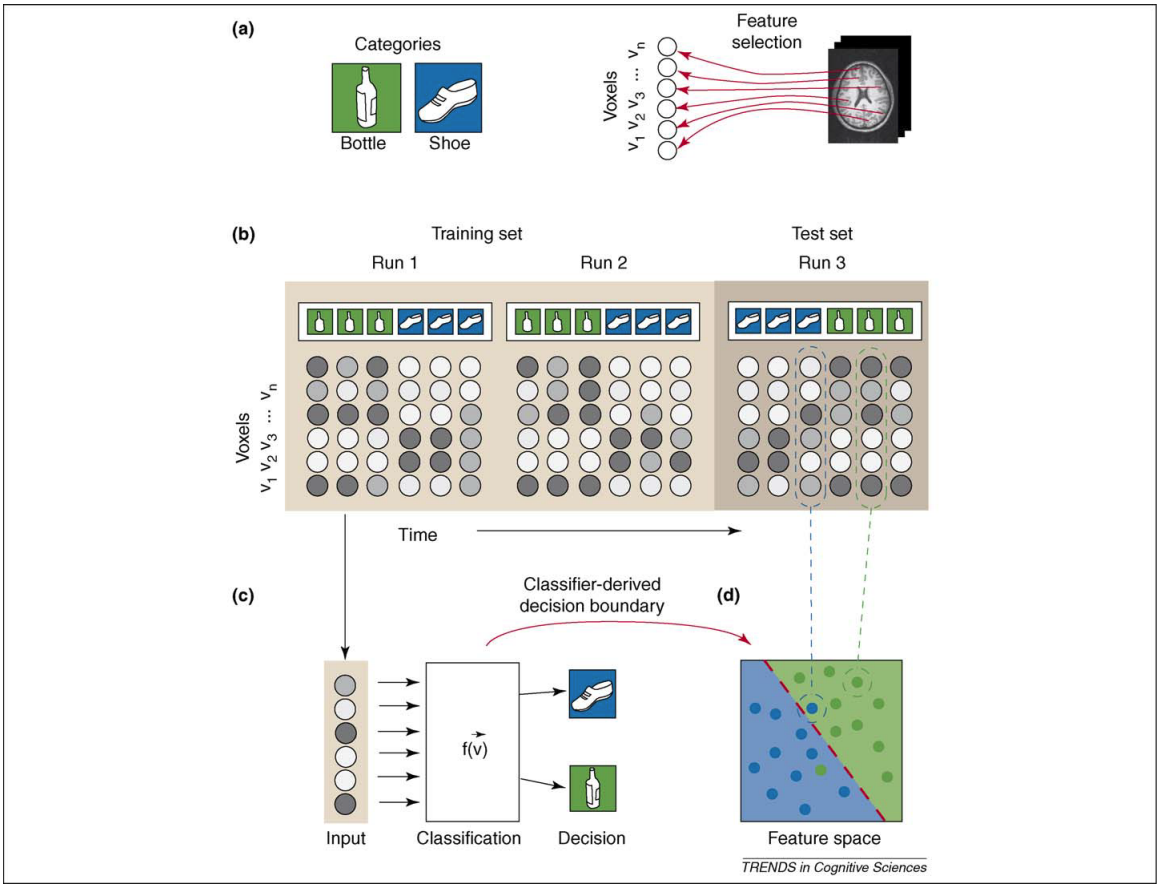

Figura 1.2.1: La figura descrive gli step di un processo di MVPA.

namento dei voxel e si estrae, per ogni partizione, un rappresentante dell'area (centroide); di conseguenza si ottiene una riduzione delle dimensioni del dataset.

Avendo per ogni voxel la posizione del cervello associata in uno spazio di riferimento, per ogni cluster generato è stato possibile realizzare una mappa e visualizzarla su una immagine anatomica di riferimento (vedi figura 1.3.1).

Altra importante operazione effettuata dal framework è la stima della risposta emodinamica: per i voxel rappresentativi dei cluster generati, viene stimata la risposta emodinamica considerando la media dei segnali dei vari trial (vedi figura 1.3.2).

Considerando il nuovo dataset contenente i soli voxel rappresentanti dei cluster, è stato utilizzato uno schema di cross-validation con classificatore SVM (lineare e non lineare) che include:

- ricerca delle migliori features
- la stima dei migliori parametri per il classificatore
- test delle permutazioni sulle label del test-set

Avendo a disposizione molti risultati, ottenuti con diverse scelte per il numero di cluster, metodologia di clustering, tipologia di classificatore SVM (lineare e non lineare), il framework filtra, in maniera automatica, il risultato migliore dal punto di vista dell'errore di classificazione.

Tutti i dettagli sul framework sono discussi nel capitolo 3.

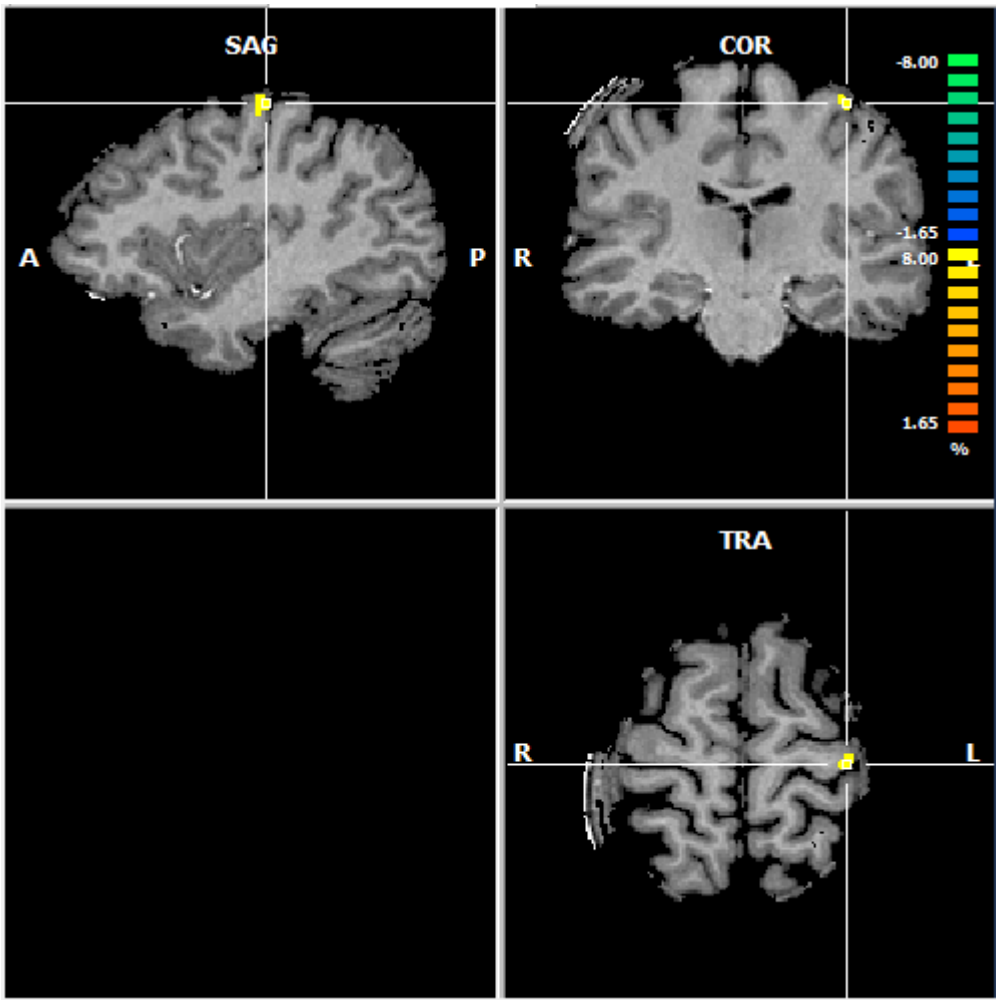

Figura 1.3.1: Mappa degli elementi di un cluster.

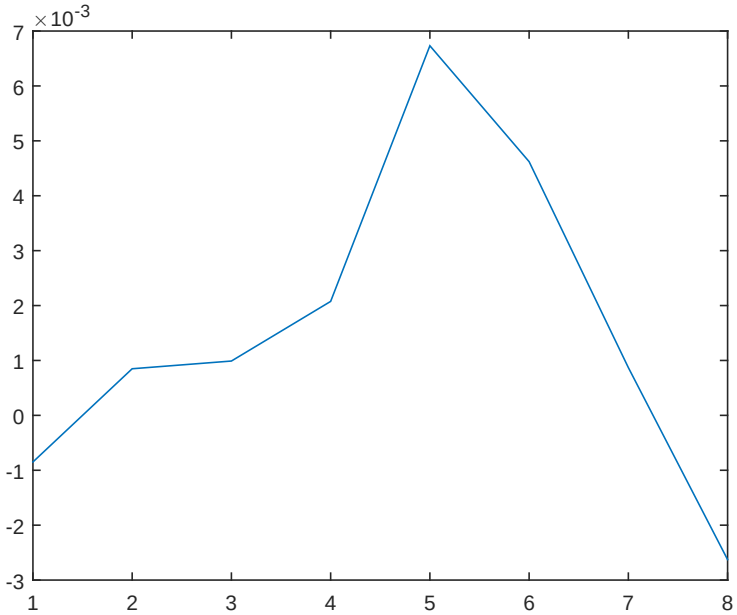

Figura 1.3.2: Stima della risposta emodinamica per i centroidi dei cluster.

Capitolo 2

Metodologie

Introduzione

In questa sezione sono illustrate le principali metodologie utilizzate nel framework AfL per l'analisi dei dati fMRI. In particolare sarà descritta la tecnica PCA (principal component analysis) lineare e non lineare, il GLM (modello lineare generalizzato), gli algoritmi di clustering PAM (partitioning around medoids) e clustering gerarchico con due metriche di distanza (Pearson e Spearman) ed infine verrà descritta la tematica della classificazione, con particolare importanza al classificatore SVM (lineare e non lineare).

2.1 PCA

L'analisi delle componenti principali (PCA) è la tecnica maggiormente diffusa quando si affronta il problema di riduzione delle dimensionalità, estrazione delle features o visualizzazione di dati ad alta dimensionalità. Nel caso di riduzione delle dimensionalità, la PCA minimizza la perdita di informazione.

Esistono due diverse descrizioni della PCA. La prima definisce la PCA come la proiezione ortogonale dei dati in uno spazio lineare più basso (sottospazio principale), tale che la varianza dei dati proiettati è massimizzata. Alternativamente la si può definire come la proiezione lineare che minimizza la media del costo di proiezione, definito come distanza media al quadrato tra i punti del dataset e le loro proiezioni. Le definizioni date sono mostrate nella figura 2.1.1.

2.1.1 Formulazione a massima varianza

Consideriamo un insieme di N osservazioni x_n con $n = 1, ..., N$ in uno spazio a D dimensioni. L'obiettivo della PCA è proiettare i dati in un sottospazio di dimensionalità M con $M < D$ che rende massima la varianza dei dati. Consideriamo il caso più semplice con $D = 2$ ed $M = 1$. Definiamo una direzione arbitraria nello spazio a 2 dimensioni con un vettore u che scegliamo, per semplicità, come vettore unitario. A questo punto il valore di ciascun x_n sulla componente u è dato dal prodotto $u_1^T x_n$ (che è, nel caso specifico, uno scalare). A questo punto è facile calcolare la varianza dei dati proiettati (che vogliamo

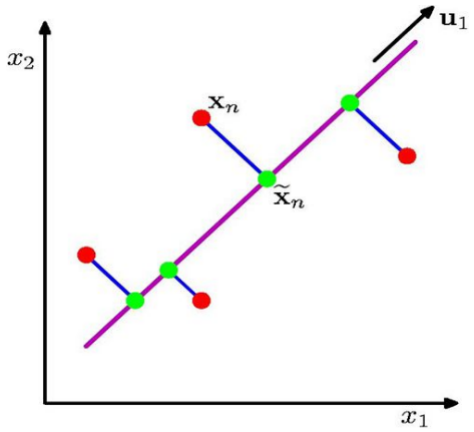

Figura 2.1.1: L'analisi delle componenti principali cerca uno spazio a dimensionalità più bassa detto sottospazio principale (linea magenta), tale che la proiezione ortogonale dei punti del dataset (punti rossi) nel sottospazio massimizza la varianza dei dati proiettati (punti verdi). Alternativamente si può definire la PCA come la minimizzazione della somma dei quadrati degli errori di proiezione (linee blu).

massimizzare) con la formula

$$\frac{1}{N}\sum_{n=1}^{N}\{u_1^T x_n - u_1^T \overline{x}\}^2 = u_1^T S u_1 \tag{2.1.1}$$

dove $\overline{x}$ è la media dei dati, calcolata come

$$\overline{x} = \frac{1}{N}\sum_{n=1}^{N} x_n \tag{2.1.2}$$

L'obiettivo è quello di massimizzare la varianza dei dati proiettati $u_1^T S u_1$[3].

2.1.2 PCA non lineare

La PCA non lineare (NLPCA)[5] è una estensione del principio base della PCA. Lo scopo è ancora quello di proiettare i dati in un sottospazio a più bassa dimensionalità, andando ad individuare componenti non lineari (vedi 2.1.2).

È noto che molti fenomeni naturali assumono un andamento non lineare nello spazio originario. Questo è facilmente osservabile con dati di serie temporali. Di conseguenza, ridurre la dimensionalità di questa tipologia di dati richiede l'utilizzo di approcci non lineari.

La PCA non lineare è implementata solitamente con una rete neurale MLP (Multi-Layers Perceptron), con topologia autoassociativa. La rete esegue un mapping di identità tra il vettore in input x ed il vettore di output$\hat{x}$ minimizzando l'errore $E = \frac{1}{2}||\hat{x} - x||^2$ (vedi figura 2.1.3).

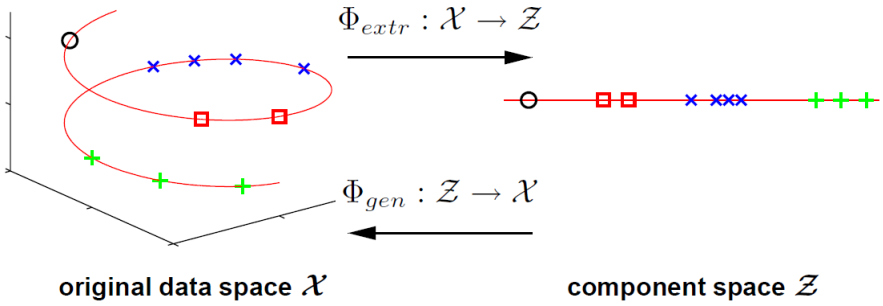

Figura 2.1.2: La figura mostra il mapping di punti da uno spazio tridimensionale ad uno spazio unidimensionale senza perdita di informazione. La trasformazione è data dalle due funzioni Φ_{extr} e Φ_{gen}. La funzione di estrazione Φ_{extr} effettua il mapping di ciascun sample nello spazio 3D in un nuovo sample in uno spazio monodimensionale. Il mapping inverso è fatto dalla funzione Φ_{gen}.

2.2 GLM

Il GLM è utilizzato per predire il valore di una variabile dipendente in funzione di una o più variabili indipendenti dette anche predittori. Supponiamo di condurre un esperimento in cui misuriamo il valore di una *variabile di risposta* Y_j. Supponiamo inoltre di avere un insieme di L predittori denotati con x_{jl}, dove $l = 1, ..., L$ indicizza la variabile esplicativa rispetto alla variabile di risposta.

Esprimere il valore di Y_j in termini di una combinazione lineare dei predittori più un termine di errore significa, matematicamente, risolvere:

$$Y_j = x_{j1}\beta_1 + ... + x_{jl}\beta_l + x_{kL}\beta_L + \epsilon_j$$

I parametri β_l con $(l = 1, ..., L)$ sono le incognite da stimare; il

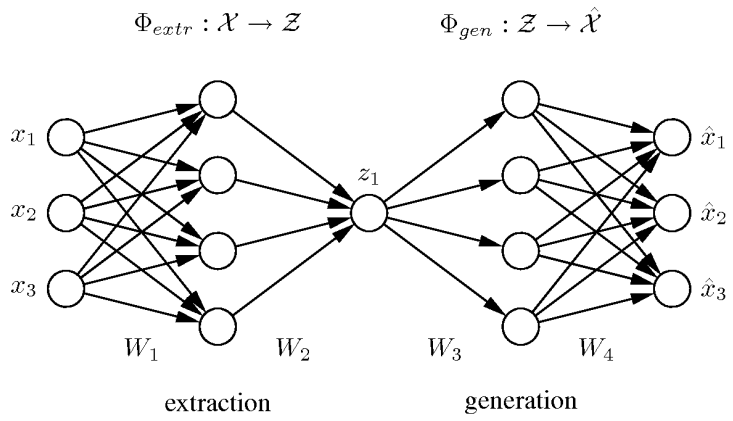

Figura 2.1.3: **Rete neurale autoassociativa.** Scopo della rete è fare in modo che l'output $\hat{x}$ sia uguale all'input x . La rete in figura ha una architettura 3-4-1-4-3. Le tre componenti in input sono compresse in un solo valore z nel centro della rete; a partire dal valore di z si ricostruisce l'output $\hat{x}$. La rete può essere costruita in modo da estrarre più componenti principali aggiungendo ulteriori unità nel layer centrale.

termine ϵ_j è il termine di rumore che modella tutto ciò che non si riesce a spiegare con i predittori nei dati.

2.2.1 GLM applicato all'fMRI

Lo scopo fondamentale del GLM nel caso di dati fMRI è quello di studiare ogni voxel in maniera univariata e capire se un voxel risponde diversamente a diverse condizioni sperimentali.

Nel caso di dati fMRI, per ottenere la variabile dipendente si procede come segue:

- definizione di un segnale che vale 1 negli istanti in cui una certa condizione è "on" e 0 negli istanti in cui è "off"
- convoluzione di questo segnale con la risposta emodinamica ideale

L'immagine in figura 2.2.1 mostra il processo descritto.

Ogni predittore avrà un β associato che spiega quanto il predittore è rappresentato dal time course.

2.3 Clustering

Il clustering è una tecnica che consente di dividere i dati in gruppi, detti cluster, contenenti elementi "vicini" tra loro. Tecniche di clustering sono state utilizzate in diversi campi che includono psicologia, biologia, statistica, pattern recognition e machine learning.

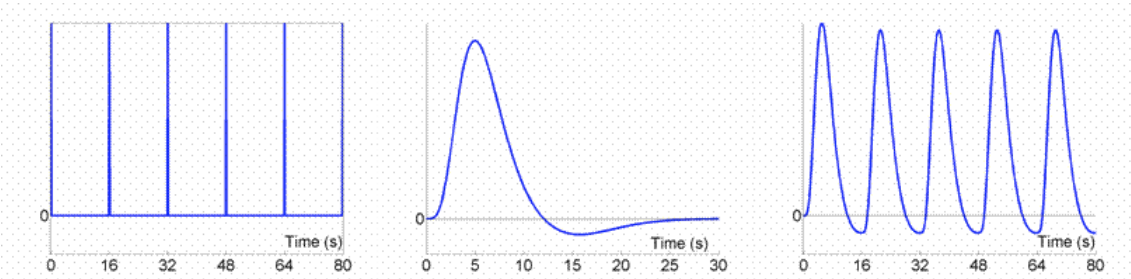

Figura 2.2.1: Modellazione predittore per GLM.

Il goal degli algoritmi di clustering è quello di individuare caratteristiche intrinseche nella natura dei dati, creando delle partizioni tali che elementi nella stessa partizione sono simili secondo qualche criterio.

Una delle scelte più difficili quando si utilizzano algoritmi di clustering è relativa al numero di cluster da creare. La figura 2.3.1 mostra tre diversi approcci per clusterizzare gli stessi oggetti.

La scelta del numero di partizioni è effettuata utilizzando:

- conoscenza dei dati
- metodi che, dato un partizionamento dei dati, valutano la "qualità" del partizionamento (es. Silhouette)

2.3.1 Silhouette

Il metodo della Silhouette fornisce una stima della consistenza di una partizionamento dei dati. Il coefficiente di Silhouette rappresenta una

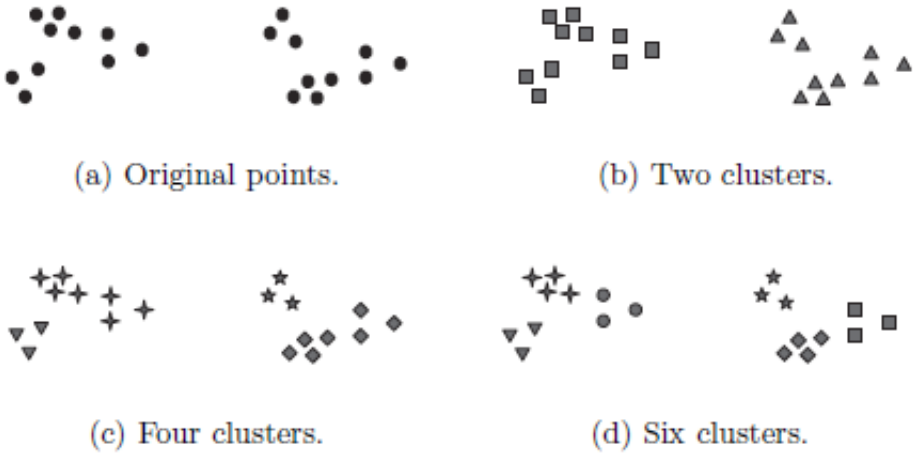

Figura 2.3.1: Differenti approcci per classificare gli stessi punti.

misura di quanto un oggetto è simili agli oggetti presenti nel suo stesso cluster (coesione) comparato agli altri cluster (separazione). Tale coefficiente è compreso tra -1 ed 1, dove il valore 1 indica che un oggetto si lega bene con altri oggetti nello stesso cluster e non lega con oggetti dei cluster vicini.

Se molti oggetti ottengono un valore alto si può dedurre che la configurazione fornita dall'algoritmo di clustering è appropriata. In caso contrario il numero di partizioni create potrebbe non essere coerente con la natura dei dati.

2.3.2 Metriche di distanza

Essendo lo scopo del clustering quello di raggruppare elementi "vicini", una delle prime considerazioni da fare è sicuramente legata alla metrica

di distanza da utilizzare. Diverse sono le definizioni di distanza (o similarità) tra oggetti; di seguito alcune tra le più comuni:

- distanza euclidea
- distanza di Manhattan
- distanza di Mahalanobis

Alternativamente può essere utilizzata una misura di correlazione tra le osservazioni. Il coefficiente di correlazione tra due variabili solitamente assume un valore compreso nell'intervallo $[-1; 1]$, dove:

- -1 indica una correlazione negativa
- 1 indica una correlazione positiva
- 0 indica che le due osservazioni sono incorrelate

A partire dalla correlazione, la distanza è calcolata come $1-\mathit{correlazione}$. Due esempi di misure di correlazione sono il coefficiente di correlazione di Pearson ed il coefficiente di correlazione di Spearman.

Il coefficiente di correlazione di Pearson, date due variabili X e Y, è definito come segue:

$$\rho_{XY} = \frac{\sigma_{XY}}{\sigma_X \sigma_Y}$$

dove σ_{XY} è la covarianza tra X e Y e σ_X, σ_Y sono le due deviazioni standard.

Il coefficiente di correlazione di Spearman è molto simile al coefficiente di Pearson ma prevede un ranking iniziale delle variabili prima

di calcolare la correlazione. Calcolata la differenza D dei ranghi delle due variabili come:

$$D_i = r_i - s_i$$

dove r_i ed s_i rappresentano rispettivamente il rango della prima e della seconda variabile alla i-esima osservazione, il coefficiente di correlazione di Spearman si ottiene come segue:

$$\rho_s = 1 - \frac{6 \sum_i D_i^2}{N(N^2 - 1)}$$

Stabilita una metrica di distanza tra le osservazioni è possibile utilizzare differenti algoritmi di clustering, tra cui:

- clustering gerarchico
- k-medoids

Di seguito saranno illustrati i concetti fondamentali di entrambi i metodi elencati.

2.3.3 Clustering gerarchico

Il clustering gerarchico ha come obiettivo quello di costruire una gerarchia di cluster. Esistono due differenti approcci per creare tale gerarchia:

- agglomerativo
- dispersivo

Il primo parte creando un numero di cluster pari al numero di elementi, mettendo ciascun elemento in un cluster diverso. In maniera bottom-up si accorpano cluster a due a due. L'approccio divisivo è invece del tipo top-down: tutti gli elementi sono inizialmente nello stesso cluster che viene suddiviso riscorsivamente in sotto-cluster.

Quando si utilizza l'approccio agglomerativo, man mano che si costruiscono i cluster c'è necessità di valutare la distanza tra gruppi di elementi; la matrice di dissimilarità non basta. Diverse sono le tecniche utilizzate, tra cui:

- *Single:* la distanza tra due cluster è la minima distanza tra un elemento in un cluster ed un elemento in un altro cluster (vedi figura 2.3.2)
- *Average:* la distanza tra due cluster è data dalla media aritmetica delle distanze tra tutte le coppie di unità che compongono i due gruppi (vedi figura 2.3.4)
- *Complete*: la distanza tra due cluster è la massima distanza tra un elemento in un cluster ed un elemento in un altro cluster (vedi figura 2.3.3)
- *Ward:* minimizza la varianza degli elementi nello stesso cluster quando due cluster sono combinati

2.3.4 Clustering PAM

Il clustering PAM (Partitioning Around Medoids) rappresenta una particolrae implementazione dell'algoritmo *k-medoids.* Tale algoritmo

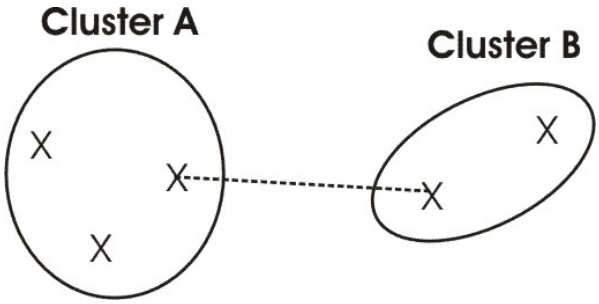

Figura 2.3.2: Clustering gerarchico agglomerativo con metodo *single.*

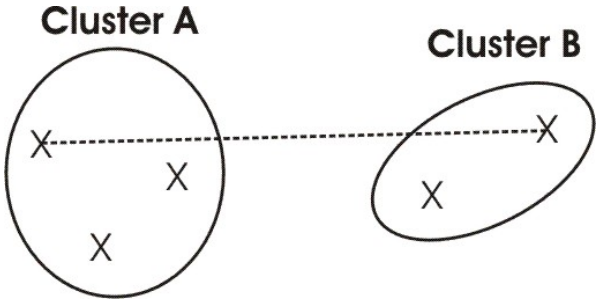

Figura 2.3.3: Clustering gerarchico agglomerativo con metodo *complete.*

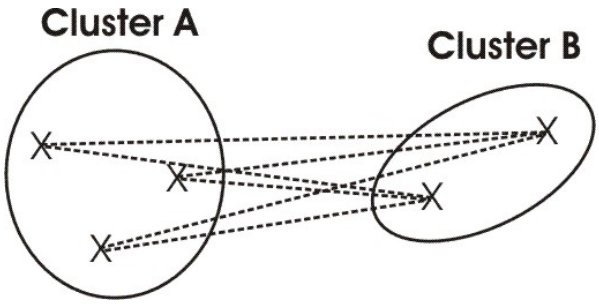

Figura 2.3.4: Clustering gerarchico agglomerativo con metodo *average.*

è concettualmente simile all'algoritmo *k-means* in quanto ha ancora come obiettivo quello di dividere i dati in gruppi minimizzando la distanza tra gruppi diversi ma, come centro dei cluster, k-medoids seleziona un punto stesso del dataset (medoid).

L'algoritmo PAM utilizza un approccio greedy per cercare una soluzione che potrebbe non essere ottima ma è computabile in tempi ragionevoli. Una volta stabilito il numero k di cluster da formare, l'algoritmo lavora come descritto in 1.

```
initialization: randomly select k of n points as medoids;
Associate each data point to the closest medoid;
while the cost of the configuration decreases do
    foreach medoid do
        foreach non − medoid do
            swap medoid and non − mdeoid;
            if new configuration cost > old configuration cost
             then
                undo the swap;
            end
        end
    end
end
```

Algorithm 1: Partitioning around medoids.

2.4 Features selection

Uno dei problemi più frequenti da risolvere quando si utilizzano tecniche di machine learning è legato alla alta dimensionalità dei dati. Il

tempo necessario, ad esempio, ad addestrare un modello di classificazione (vedi sezione 2.5) è molto dipendente dalla dimensionalità dei dati. Quello che si tende a fare quindi è ridurre la dimensionalità dei dati cercando di mantenere il più possibile il loro contenuto informativo. Un generico algoritmo di features selection prende in input un insieme di features con dimensione n e restituisce un nuovo insieme di features a dimensione $< n$.

Considerando un dataset di dimensionalità $n \times p$ dove p è la dimensionalità da ridurre, una semplice tecnica di features selection può essere la ricerca esaustiva di n' features (con $n' < n$) tali da rendere minimo l'errore di classificazione del modello.

2.5 Classificazione

L'obiettivo degli algoritmi di classificazione è quello di associare un vettore in input x ad una tra K classi C_K, dove $k = 1, ..., K$. Solitamente le classi sono disgiunte ed ogni punto può essere assegnato ad una sola classe. L'idea è quella di creare una suddivisione dello spazio in *decision regions* delimitate da *decision boundaries* che fanno da confine tra regioni di differenti aree (vedi [4]).

Le classi vengono solitamente rappresentate con dei valori, detti *target*. Per modelli probabilistici, nei casi con due sole classi, si usa una singola variabile target $t \in \{0, 1\}$ tale che $t = 1$ rappresenta la classe C_1 $t = 2$ rappresenta la classe C_2. Possiamo intuitivamente vedere t come la probabilità che l'input x appartiene alla classe C_1, assumendo che il valore della probabilità valga 0 oppure 1.

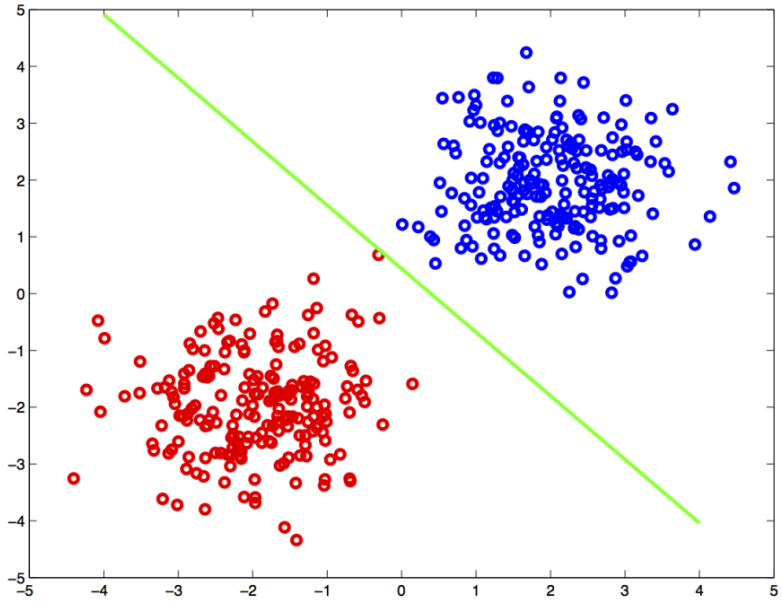

Figura 2.5.1: Esempio di classificazione di dati su due dimensioni.

Quando il numero K di classi è maggiore di due, si usa solitamente scegliere il target t come un vettore di lunghezza K tale che, se x appartiene alla classe C_j, l'elemento t_j sarà pari ad 1 e gli altri elementi saranno pari a 0.

La figura 2.5.1 mostra un esempio di classificazione di dati su due dimensioni. I punti blu e rossi appartengono a due classi diversi e, con un classificatore lineare, si riesce perfettamente a dividere i dati nelle due classi di appartenenza.

Il problema della classificazione può essere suddiviso in due categorie:

1. classificazione supervisionata

2. classificazione non supervisionata

I metodi supervisionati rappresentano una tecnica di apprendimento automatico che costruisce un modello a partire da coppie di input e output desiderati. I metodi non supervisionati costruiscono un partizionamento di dati senza conoscenze a priori ma basandosi su caratteristiche intrinseche dei dati.

2.5.1 Cross-validation

Per le tecniche di classificazione supervisionata c'è quindi la necessità di una fase di training durante la quale si crea un modello a partire dai dati e dalle label. La fase di training ha come obiettivo quella di minimizzare l'errore di classificazione e di avere alta capacità di generazione su nuovi dati. Il problema a cui si va incontro durante la fase di training è che il modello puo' apprendere troppo dai dati e non riuscire quindi a generalizzare su nuovi dati. Questa situazione, detta *overfitting,* è caratterizzata da un errore di classificazione sul training molto basso (vedi figura 2.5.2).

Un metodo per validare la bontà di un modello di classificazione supervisionato è la cross-validation. Tale metodo consiste nel dividere i dati di training in k fold (partizioni) e di utilizzare, ciclicamente, $k-1$ partizioni per il training e la partizione restante per il testing. In questo modo possiamo verificare la bontà del modello su diverse partizioni e calcolare, ad esempio, un errore medio del modello.

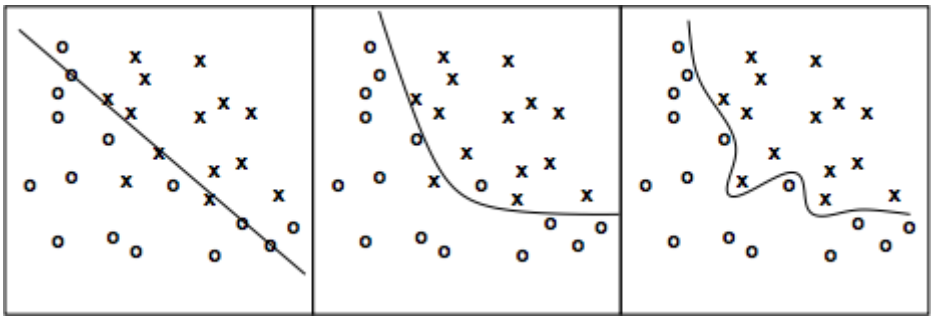

Figura 2.5.2: L'immagine mostra tre diversi modelli per partizionare dei dati divisi in due categorie (cerchi e crocette). Il modello a sinistra sembra descrive i dati con un errore abbastanza elevato. Il modello a destra non commette alcun errore ma è cucito ad hoc sui dati. Il modello centrale rappresenta il giusto compromesso.

2.5.2 SVM

L'SVM (Support Vector Machine) appartiene alla categoria di algoritmi di apprendimento supervisionati utilizzato per la classificazione. È necessaria dunque una fase di training durante la quale si costruisce un modello che descrive i dati in input; tale modello puo' essere utilizzato, in seguito, per generalizzare su nuovi dati.

Scopo dell'SVM è quello di costruire un iperpiano (o un insieme di iperpiani) che separa i dati. In particolare l'iperpiano costruito non solo dovrà minimizzare l'errore di classificazione (sui dati di training) ma dovrà anche massimizzare la distanza dal punto di training più vicino. L'immagine 2.5.3 mostra un esempio di classificazione con SVM.

Spesso ciò che accade è che i dati non sono separabili linearmente

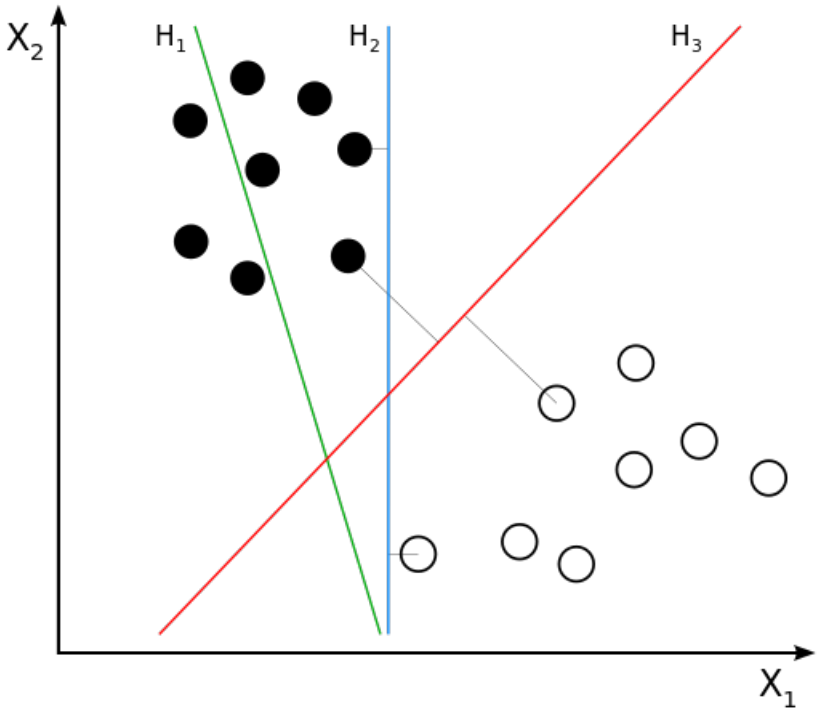

Figura 2.5.3: L'immagine mostra tre differenti iperpiani. L'iperpiano H1 non separa le classi; H2 le separa senza commettere errori ma con una distanza dal punto più vicino bassa; H3 separa i dati senza errore e massimizza la distanza dal punto più vicino.

ed è difficile trovare un iperpiano che separi bene i dati. In questi casi si rende opportuno l'utilizzo dei *kernel,* funzioni che effettuano il mapping dei dati in un nuovo spazio a dimensioni più alte dove si può riutilizzare un modello lineare. Un kernel k ha associato un mapping ϕ delle features. Un mapping ϕ prende in input una osservazione x appartenente allo spazio iniziale X ed effettua il mapping di x in un nuovo spazio F.

2.6 Analisi statistiche

Un test statistico di significatività è utilizzato per verificare la validità di una data ipotesi, detta *ipotesi nulla.* Ad esempio se lo scopo è quello di verificare la uguaglianza tra diversi campioni, l'ipotesi nulla prevede che i campioni siano uguali. Quello che si vuole studiare è la eventuale possibilità di rigettare l'ipotesi nulla. La probabilità di commettere errore rigettando l'ipotesi nulla, è chiamata *livello di significatività* del test.

Il livello di significatività del test può essere scelto in maniera arbitraria; solitamente, un test si ritiene significativo se si ottiene un livello di significatività < 0.05.

Il t-test è un particolare test statistico utilizzato quando si vogliono confrontare le medie di due gruppi di osservazioni e si vuole verificare se esiste o meno differenza (significativa) tra le due medie. Quando si utilizza il t-test, è necessario rispettare alcune ipotesi; in particolare:

- i dati devono seguire una distribuzione Normale

- non deve esserci dipendenza tra le osservazioni

Capitolo 3

AfL framework

Introduzione

Le tecniche spiegate nel capitolo 2 sono utilizzate per studiare dati complessi, dove la complessità è data dall'alta dimensionalità. Nel caso specifico, tali tecniche sono state utilizzate per analizzare dati di risonanza magnetica funzionale.

Questo capitolo spiega in dettaglio la struttura dei dataset analizzati e la pipeline implementata in AfL per trattare questa tipologia di dati.

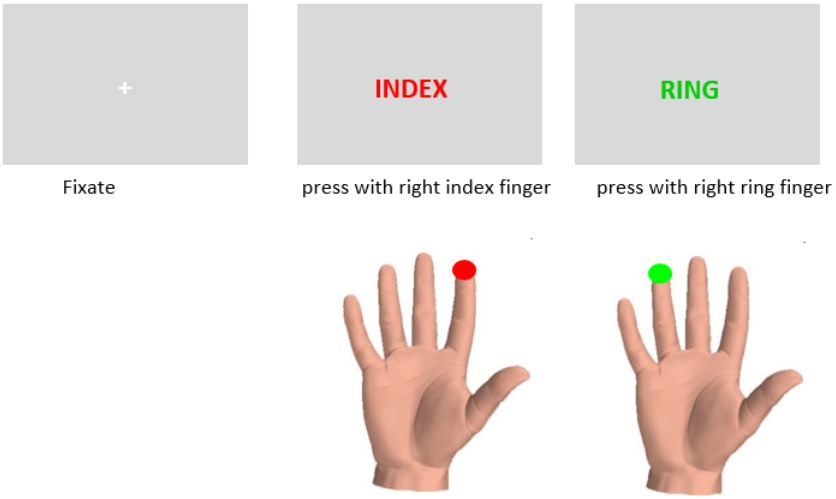

Figura 3.1.1: Istruzioni visuali mostrate ai partecipanti dell'esperimento.

3.1 Esperimento e dataset

I dati analizzati per questo lavoro di tesi sono stati raccolti con tre soggetti a cui veniva chiesto di effettuare il seguente task: seguendo delle istruzioni visuali, i partecipanti erano invitati a premere un pulsante con due dita diverse (indice ed anulare). La figura 3.1.1 mostra le istruzioni fornite ai partecipanti.

L'esperimento appartiene alla categoria *slow event related design* (è stata richiesta ai partecipanti una pressione di pulsante ogni 16 secondi), acquisendo un campione ogni due secondi. Ci si è focalizzati soltanto sull'area cerebrale legata (dalla letteratura) al particolare esperimento (corteccia motoria, somatosensoriale e controlaterale), ot-

tenendo una dimensione spaziale di circa 10^3 voxel. Il segnale misurato per una singola pressione è rappresentato da una matrice $t \times v$ dove:

- t è il numero di samples (8)
- v è il numero di voxel (circa 10^3)

In totale sono state registrate 96 risposte, 48 per dito. Avendo per ogni risposta 8 registrazioni (una ogni 2 secondi), la matrice a cui è stata applicata la pipeline realizzata, è delle dimensioni di circa $(96 \times 8) \times (2 \times 10^3)$ (vedi figura 3.1.2).

Partendo da dataset così composti, il framework realizzato effettua i seguenti step:

- Compressione dei dati
- Raggruppamento dei voxel
- Generazione delle mappe dei cluster
- Stima della risposta emodinamica
- Generazione modello SVM per la generalizzazione su nuovi dati

Di seguito sono illustrate nei dettagli tutte le operazioni elencate.

3.2 Compressione dei dati

Analizzare dati in spazi ad alta dimensionalità (come dati fMRI) con approcci di machine learning è molto complicato; in particolare si hanno due problemi legati alla dimensionalità:

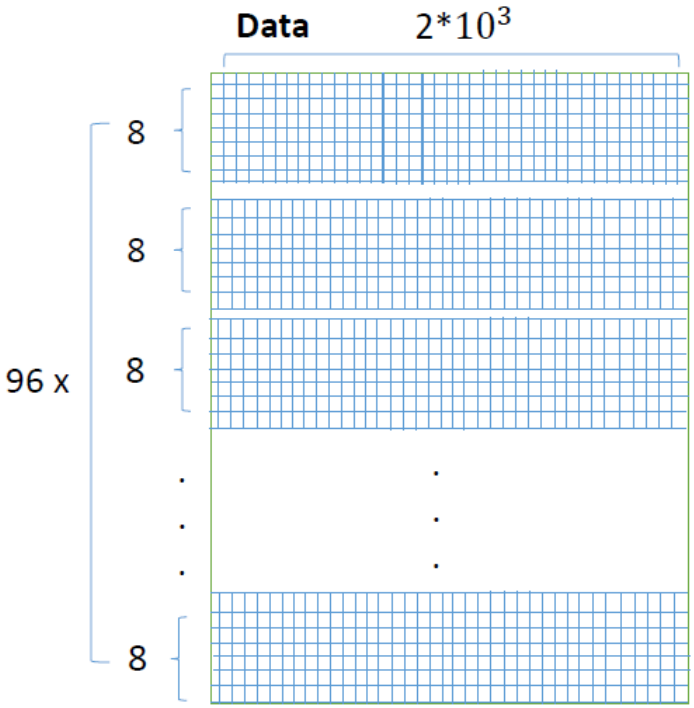

Figura 3.1.2: Struttura dataset analizzati.

1. ogni osservazione è rappresentata da più valori nel tempo (segnale)

2. dimensionalità legata al numero di voxel analizzati

Per quanto riguarda il punto 1, ogni serie temporale (pattern), è stata compressa a pochi punti utilizzando diverse tecniche (media aritmetica, PCA non lineare e GLM). Per il punto 2, nonostante i dati analizzati dati fossero già ridotti ad una specifica area del cervello (circa 2000 voxel), c'è stata necessità di raggruppare questi dati utilizzando algoritmi di clustering.

3.2.1 Compressione serie temporali

Trovare un metodo per *riassumere* l'informazione contenuta nelle serie temporali senza perdere informazione è un compito molto complesso. Diversi sono gli approcci utilizzati in questo lavoro di tesi per comprimere le serie temporali; in particolare:

- media aritmetica dei valori
- GLM su tre basi
- PCA non lineare

Specificando opportunamente i parametri in input al framework (vedi listato 2) è possibile selezionare la tipologia di compressione da utilizzare.

```
%--------PARAMETERS--------%
delimiters1 = 2;
delimiters2 = 7;
USE_MEAN_TIME_SERIES = 0;
USE_PCA_FOR_TRIALS = 0;
NUM_OF_COMPONENTS = 3; %for pca on trials
USE_GLM = 1;  %decide in function of USE PCA FOR TRIALS
clusterSize = [5:10 20:10:50];
isConfOk = checkConfiguration(USE_MEAN_TIME_SERIES,USE_PCA_FOR
    _TRIALS,USE_GLM);
if (isConfOk == 0)
  error('Error! check your configuration!');
  return;
end
%-------------------------%
```

Algorithm 2: Scelta dei parametri iniziali di AfL. I parametri *USE_MEAN_TIME_SERIES*, *USE_PCA_FOR_TRIALS* e *USE_GLM* consentono all'utente di specificare una metodologia di compressione delle serie temporali. Nel caso l'utente selezioni come metodologia di compressione la media aritmetica, è possibile restringere la media ai campioni racchiusi in un particolare intervallo, specificando i valori *delimiters1* e *delimiters2*. Il parametro *NUM_OF_COMPONENTS* è il numero di componenti da estrarre nel caso si utilizzi la compressione con PCA non lineare. Il vettore *clusterSize* specifica i diversi valori di k da utilizzare per il clustering.

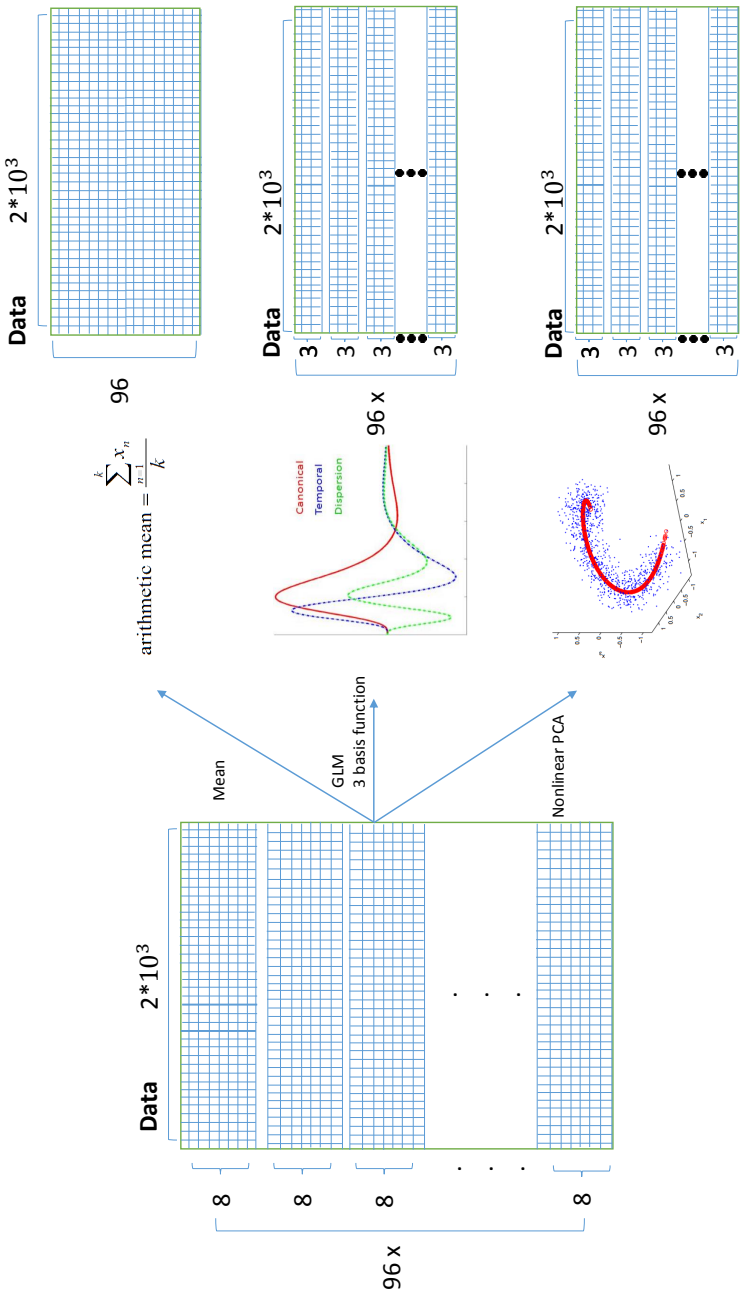

Figura 3.2.1: Compressione delle serie temporali con le tre metodologie implementate nel framework.

Compressione con media

Partendo da dataset mostrato in figura 3.1.2, ogni serie temporale (8 samples) è compressa effettuando una media aritmetica dei valori. A questo punto il nuovo dataset avrà dimensionalità $96 \times (2 \times 10^3)$.

Compressione con GLM

Il GLM può essere utilizzato per comprimere una serie temporale, sostituendo il time course con il valore di uno o più β opportunamente stimati. Nel caso specifico, ogni time course (8 punti) è compresso andando ad effettuare il fitting del segnale con tre diversi segnali:

- HRF canonica
- derivata temporale dell'HRF
- derivata dispersiva dell'HRF

In questo modo si ottengono tre β che riassumono il segnale originale (otto valori). In particolare il β relativo alla derivata temporale dell'HRF canonica contiene informazioni sulla latenza del segnale a raggiungere il picco; il β invece relativo alla derivata dispersiva dell'HRF contiene informazioni sulla durata del picco. Insieme dunque, i tre β forniscono un buon riassunto del segnale di partenza.

Compressione con PCA non lineare

Ulteriore possibilità offerta dal framework è quella di comprimere le serie temporali utilizzando la PCA non lineare (vedi 2.1.2). In questo

modo, ciascuna matrice $8 \times (2 \times 10^3)$ è proiettata in uno spazio di $k \times (2 \times 10^3)$, dove il k è specificato dall'utente (vedi 2). Per gli esperimenti effettuati il k è stato fissato a 3.

3.2.2 Clustering

La compressione delle serie temporali è la prima fase della riduzione delle dimensionalità. Effettuato questo step si riducono le dimensioni di ogni osservazione partendo da 8 samples per osservazione ed arrivando ad 1 sample nel caso della media o a 3 samples nel caso di compressione con GLM su tre basi o PCA non lineare. La figura 3.2.1 mostra le tre matrici a cui ci riconduciamo con le diverse metodologie di compressione.

A questo punto, per ridurre il numero di colonne (voxel), sono stati utilizzati due diversi algoritmi di clustering, clustering gerarchico e PAM, utilizzando due diverse metriche di distanza (Pearson e Spearman) (vedi 2.3). I valori di k utilizzati sono $5, 6, 7, 8, 9, 10, 20, 30, 40, 50$.

Come descritto nella sezione 2.3, l'algoritmo PAM ricava, per ogni cluster, un medoid (punto appartenente al dataset) che può essere visto come il *rappresentante* del cluster. Pertanto, per ogni valore di k, si ottiene in output una matrice $n \times k$, dove n è il numero di osservazioni (che dipende dalla tipologia di compressione temporale utilizzata) e k è il numero di cluster. Il clustering gerarchico,invece, restituisce un vettore di lunghezza pari al numero di elementi da raggruppare contenente. Tale vettore contiene, alla posizione i-esima, l'indice del cluster a cui l'elemento i è assegnato. Dovendo nel contesto specifico generare una matrice $n \times k$, è stata implementata una euristica per creare tale

matrice a partire dal vettore output del clustering gerarchico. Tale euristica lavora come descritto in nell'algoritmo 3.

foreach k **do**
  el_k = elements in cluster k;
  sub_mat_dist = distances between el_k;
  $centroid\_k$ = element that maximizes the sum of distances in sub_mat_dist;
end

Algorithm 3: L'algoritmo descritto seleziona, a partire dal vettore di assegnamento dei punti ai vari cluster, k punti (se k è il numero di cluster) come rappresentanti dei vari cluster.

Alla fine di questo step si ottengono tante nuove matrici, ciascuna derivata da una particolare scelta di k, algoritmo di clustering (linkage o PAM) e metrica di distanza. La struttura di tali matrici è mostrata in figura 3.2.2.

3.2.3 Valutazione dei cluster

Il processo di clustering descritto in 3.2.2 ha il ruolo di features selection. Quello che accade è che gruppi di voxel vengono associati ad una stessa categoria, rappresentata da un punto particolare del cluster (medoid). La fase di classificazione che segue lavora dunque sui rappresentanti dei cluster.

È opportuno quindi trovare un modo per validare i cluster formati. Per tale scopo, ricordando che:

- conosciamo i voxel appartenenti ad ogni cluster

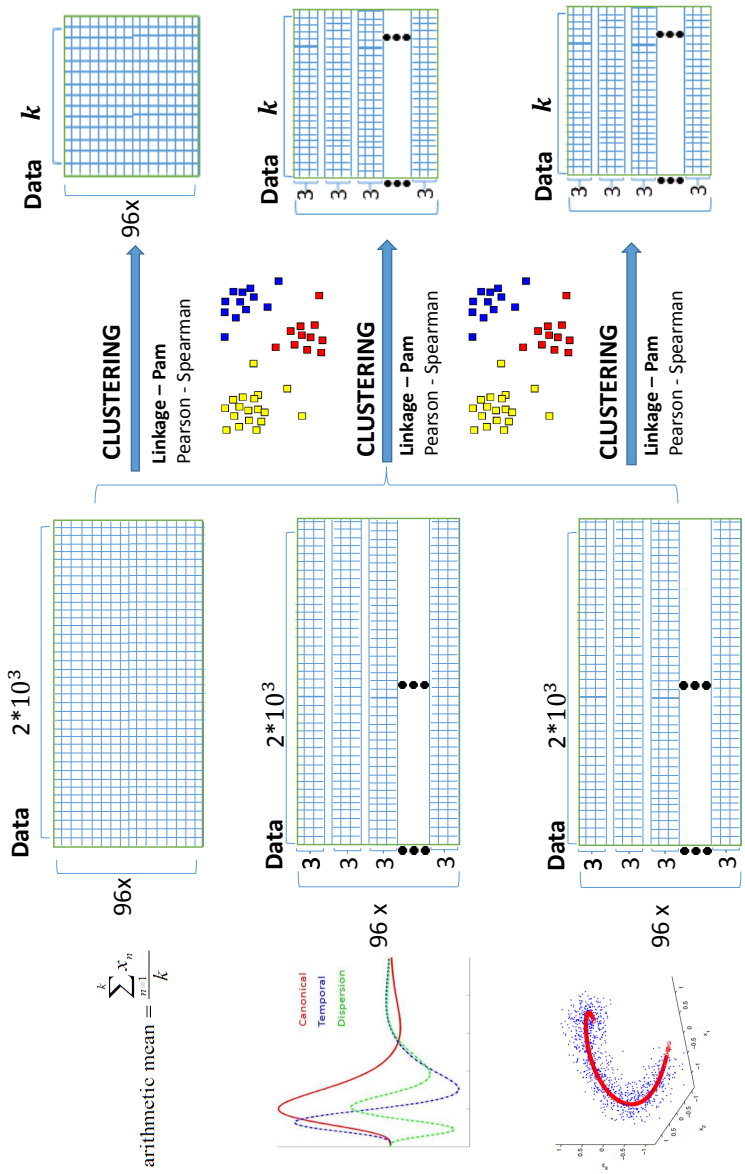

Figura 3.2.2: Matrici output del clustering.

- conosciamo le informazioni spaziali di ogni voxel in uno spazio di riferimento ben definito

è possibile realizzare, per ogni cluster, delle mappe dei voxel da visualizzare su una immagine anatomica di riferimento. Questa fase è fondamentale per capire, nella fase successiva di classificazione, se i voxel selezionati ed utilizzati per la classificazione hanno un senso anche dal punto di vista morfologico.

Per ogni output delle diverse tipologie di clustering utilizzate, è stata realizzato un set di mappe, una per ogni cluster. Ad esempio, per l'output del clustering ottenuto con:

- Algoritmo Linkage
- Metrica di distanza di Pearson
- $k = 10$

sono state realizzate 10 mappe, una per ogni cluster, contenente ciascuna i voxel appartenenti a quel cluster. Il colore assegnato ai voxel della mappa è il *degree of clustering,* metrica che premia cluster che hanno maggior numero di elementi contigui da un punto di vista morfologico. Il valore del degree of clustering è compreso tra 0 ed 1, dove 1 indica che gli elementi del cluster sono tutti contigui e 0 indica che gli elementi del cluster sono sparsi. Un esempio è riportato in figura 1.3.1.

Altra valutazione interessante è la stima della risposta emodinamica per il task in questione. La risposta emodinamica va stimata a partire dai segnali di ogni sample (pressione del pulsante) che è racchiusa

nella matrice di partenza $(96 \times 8) \times (2 \times 10^3)$. La terza dimensione (numero di voxel) è stata ridotta a k (con $k \in (5, 6, 7, 8, 9, 10, 20, 30, 40, 50)$) con il clustering; di conseguenza una stima della risposta emodinamica è stata ottenuta effettuando una media di tutti i segnali composti da 8 samples delle varie pressioni, utilizzando solo le colonne relative ai medoids output del cluster. Un esempio di risposta emodinamica è riportato in figura 1.3.2.

3.3 Classificazione

Tutte le operazioni di riduzione delle dimensionalità effettuate sui dati rendono dunque ragionevole l'utilizzo di modelli di classificazione. L'algoritmo di classificazione utilizzato è l'SVM (vedi 2.5.2), lineare e non lineare (kernel rbf). Per evitare problematiche legate all'overfitting, è stato implementato uno schema di cross-validation che include la stima dei parametri migliori per il classificatore e la selezione delle migliori k features. Si nota che, per parametri e features migliori, intendiamo i parametri e le features che minimizzano l'errore di classificazione.

3.3.1 Cross-validation

Consideriamo, per semplicità, il caso in cui si sceglie di comprimere le serie temporali effettuando una media dei campioni. In tale caso, come si vede in figura 3.2.1, il numero di pattern da classificare è 96 ed abbiamo una osservazione per ogni pattern. I passi fondamentali dell'algoritmo di cross-validation sono i seguenti:

- Divisione del dataset in 6 fold da 16
- Per ogni gruppo di 16 si utilizzano, a turno:
  - 16 elementi per il testing one shot
  - 80 elementi rimanenti per le fasi di training e validation

  Consideriamo dunque una singola iterazione di cross-validation dove abbiamo fissato una partizione dei dati come appena descritto. A questo punto il primo passo effettuato prevede una fase di training che prevede:

- divisione degli 80 elementi in 10 fold, di cui:
  - 8 per il training
  - 2 per la validation

Ora, per tutte le possibile combinazioni di training e validation ($\binom{8}{2}$) sono stati stimati i migliori parametri del classificatore (vedi 3.3.2), effettuando un t-test tra per valutare la differenza tra le diverse distribuzioni di errore ottenute con i vari parametri.

Il tutto è ripetuto partendo da una sola feature e aggiungendo di volta in volta una nuova feature (se l'aggiunta porta miglioramenti), fino a selezionare una percentuale di features scelte dall'utente. Si ricorda che, a questo punto della pipeline, il dataset sarà composto da k features, dove k è il numero di cluster che si è scelto di creare. Ogni feature (colonna della matrice) porta le informazioni di un cluster.

Di seguito è mostrato lo script.

```
function [res] = crossValidation(data,labels,FEATURES_
    PERCENTAGE,kernel,data_test, data_training)
  %author Simone Romano 0522500294
  %s.romano1992@gmail.com
  %kernel could be: 'linear' or 'nonlinear'
  tic;
  %leave one out
  if size(labels,1) ~= size(data,1)
    disp('Error! Labels size and data should be equals!')
    return
  end
  res.data_training = data_training;

  %folds generations
  nFolds = 8;
  res.folds = cvpartition(labels(data_training),'KFold',10);
  dataTrainingValidation = data(data_training,:);
  C = combnk(1:nFolds,2); %28 different combinations
  cp = classperf(labels);
  res.cRange = [10^-3 10^-2 10^-1 10^0 10^1];
  res.gammaRange = [10^-3 10^-2 10^-1 10^0];
  FEATURES_NUM = round(size(data,2)/100*FEATURES_PERCENTAGE);
  featuresVector = [1:size(data,2)];
  tmpRes.selectedFeatures = [];
  tmpRes.bestErrorRate = 100; %initialization
  res.bestResult.bestErrorRate = tmpRes.bestErrorRate;

```

```
if (strcmp(kernel,'linear')==1)
  tTestPair = [1:2;2:3;3:4];
  errorResults = zeros(size(C,1),size(res.cRange,2));
  for f=1:FEATURES_NUM
    fprintf('FEATURES_NUM: %d / %d\n',f,FEATURES_NUM);
    allSetsOfFeatures = combnk(featuresVector,f);
    validSetsOfFeatures = removeNotValidSets(
        allSetsOfFeatures, tmpRes.selectedFeatures);
    for featuresSetNum=1:size(validSetsOfFeatures,1) %best
        subset of featuresNum features
      fprintf('FEATURES_SUBSET_NUM: %d / %d\n',featuresSetNum
          ,size(validSetsOfFeatures,1));
      for cIndex=1:size(res.cRange,2)
        fprintf('BEST_C: %d / %d\n',cIndex,size(res.cRange,2)
            );
        for i=1:size(C,1)
          %fprintf('Training/Validation: %d / %d\n',i,size(C
              ,1));
          testingIndex = [test(res.folds,C(i,1)) + test(res.
              folds,C(i,2))];
          if (size(find(testingIndex==2),1))>0
            disp('error!')
            return
          end
          trainIndex = 1-testingIndex;
          testingIndex = find(testingIndex==1);
          trainIndex = find(trainIndex==1);
          options = optimset('maxiter',10^8);
```

```
        svmStruct = svmtrain(dataTrainingValidation(
            trainIndex,validSetsOfFeatures(featuresSetNum,:)
            ),labels(trainIndex),'kernel_function','linear',
            'boxconstraint',res.cRange(cIndex),'tolkkt',1e
            -5,'options',options);
        result = svmclassify(svmStruct,
            dataTrainingValidation(testingIndex,
            validSetsOfFeatures(featuresSetNum,:)));
        classperf(cp,result,testingIndex);
        errorResults(i,cIndex) = cp.lastErrorRate;
        svmStruct_collection(cIndex)= svmStruct; %one
            structure for each value of C
      end
    end
    %t-test
    best_c = res.cRange(tTestPair(1,1));
    col_best_c = tTestPair(1,1);
    tmpRes.error = mean(errorResults(:,col_best_c));
    for t=1:size(tTestPair,1)
      [h,p,ci,stats] = ttest2(errorResults(:,tTestPair(t,1)
          ),errorResults(:,tTestPair(t,2)));
      if (h==1)
        if( mean(errorResults(:,tTestPair(t,1)))<tmpRes.
            error || mean(errorResults(:,tTestPair(t,2)))<
            tmpRes.error)
          if mean(errorResults(:,tTestPair(t,1)))<mean(
              errorResults(:,tTestPair(t,2)))
           best_c = res.cRange(tTestPair(t,1));
```

```
                col_best_c = tTestPair(t,1);
               tmpRes.error = mean(errorResults(:,col_best_c));
             else
                best_c = res.cRange(tTestPair(t,2));
                col_best_c = tTestPair(t,2);
                tmpRes.error = mean(errorResults(:,col_best_c))
                    ;
             end
           end
         end
       end
       tmpRes.c = best_c;
       tmpRes.featuresSubset = validSetsOfFeatures(
           featuresSetNum,:);
       tmpRes.error = mean(errorResults(:,col_best_c));
       fprintf('last best error = %d; current bestError = %d
           with c=%d\n',res.bestResult.bestErrorRate,tmpRes.
           error,tmpRes.c);
       if(tmpRes.error<res.bestResult.bestErrorRate)
         tmpRes.bestErrorRate = tmpRes.error;
         tmpRes.selectedFeatures = validSetsOfFeatures(
             featuresSetNum,:);
         tmpRes.svmStruct = svmStruct_collection(col_best_c);
         res.bestResult = tmpRes;
         fprintf('Updated error! Last best error = %d - best
             features = %d\n',res.bestResult.bestErrorRate,
             tmpRes.selectedFeatures);
       end
```

```
    end
    if (size(res.bestResult.selectedFeatures,2)<f) %no
        improvement adding new features! Exit
      fprintf('Maximum number of features = %d; expected %d\n
          ',f-1,FEATURES_NUM);
      %one shot testing here
      res.oneShotSvmStruct = svmtrain(data(data_training,res.
          bestResult.selectedFeatures),labels(data_training),'
          kernel_function','linear','boxconstraint',res.
          bestResult.c,'tolkkt',1e-5,'options',options);
      res.oneShotResult = svmclassify(res.oneShotSvmStruct,
          data(data_test,res.bestResult.selectedFeatures));
      classperf(cp,res.oneShotResult,data_test);
      res.oneShotResultError = cp.lastErrorRate;
      res.kernel = kernel;
      res.permutationTest = permutationTest(res.
          oneShotSvmStruct,data(:,res.bestResult.
          selectedFeatures),labels,data_test,cp.lastErrorRate)
          ;
      res.executionTime = toc;
      return;
    end
  end
  %one shot testing here
  res.oneShotSvmStruct = svmtrain(data(data_training,res.
      bestResult.selectedFeatures),labels(data_training),'
      kernel_function','linear','boxconstraint',res.
      bestResult.c,'tolkkt',1e-5,'options',options);
```

```
    res.oneShotResult = svmclassify(res.oneShotSvmStruct,data(
        data_test,res.bestResult.selectedFeatures));
    classperf(cp,res.oneShotResult,data_test);
    res.oneShotResultError = cp.lastErrorRate;
    res.kernel = kernel;
    res.permutationTest = permutationTest(res.oneShotSvmStruct,
        data(:,res.bestResult.selectedFeatures),labels,data_
        test,cp.lastErrorRate);
    res.executionTime = toc;
    %----------------%
  else
    %nonlinear ...
  end
end


function [toReturn] = permutationTest(svmStruct,data,labels,
    testIndex,currentError)
    fprintf('Permutation test...');
    toReturn = struct();
    PERMUTATION_NUMBER = 1000;
    sizeTest = size(find(testIndex==1),1);
    v = labels(testIndex);
    for i=1:PERMUTATION_NUMBER
        permutations(i,:) = zeros(1,length(v));
        permutations(i,:) = v(randperm(length(v)));
    end
    success = 0;
```

```
    for p=1:PERMUTATION_NUMBER
        tmpLabels = labels;
        curr_labels = permutations(p,:);
        tmpLabels(testIndex) = curr_labels;
131     cp_perm = classperf(tmpLabels);
        res = svmclassify(svmStruct,data(testIndex,:));
        classperf(cp_perm,res,testIndex);
        if (cp_perm.lastErrorRate<currentError)
            success = success + 1;
136     end
        toReturn.error(p) = cp_perm.lastErrorRate;
    end
    toReturn.permutationTestResult = success/PERMUTATION_NUMBER
        ;
    fprintf('Permutation test error = %d\n', toReturn.
        permutationTestResult);
141 end
```

La figura 3.3.1 mostra una iterazione della cross-validation appena descritta.

Il modello addestrato sul training set è stato riutilizzato sul test set ed è stato ottenuto un errore one-shot. A questo punto è stato effettuato un test di permutazione sulle label del test set (1000 volte) ed è stato riutilizzaro il modello addestrato sui dati con le label permutate. È stato considerato dunque il numero di volte in cui, con le label permutate, si ottiene un errore di classificazione più basso. Se questa condizione si verifica k volte, $\frac{k}{1000}$ darà una indicazione su quanto la bontà del modello è dipendente dal caso. Quanto appena descritto è

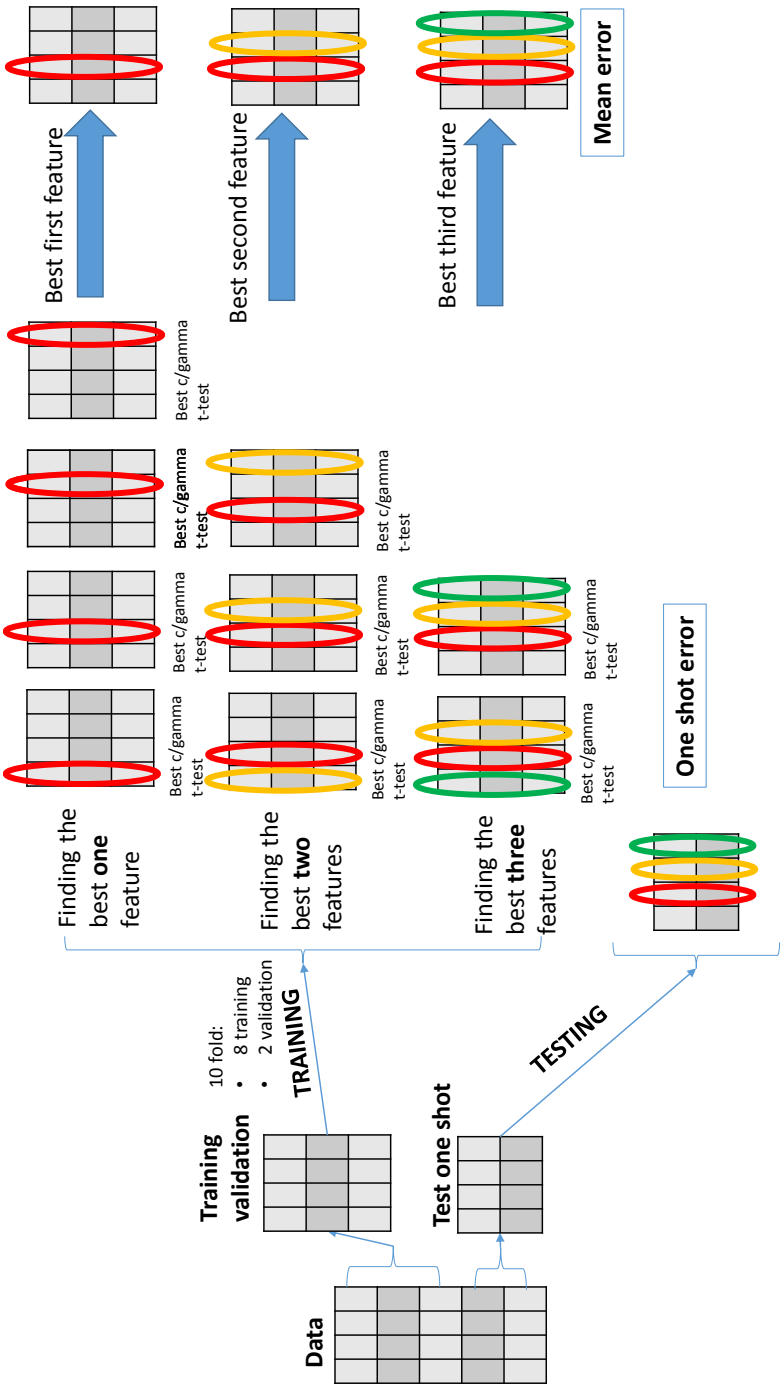

Figura 3.3.1: La figura mostra una singola iterazione della cross-validation implementata.

mostrato in figura 3.3.2).

È opportuno notare che, nel caso in cui l'utente selezioni una metodologia di compressione delle serie temporali diversa dalla media aritmetica (es. PCA non lineare o GLM su tre basi), ogni osservazione sarà composta da tre pattern. Di conseguenza è stato implementato, per quei casi specifici, un sistema che assegna una osservazione ad una determinata classe secondo uno schema a votazione: una osservazione viene etichettata con la classe a cui è assegnata due volte su tre (vedi figura3.3.3).

3.3.2 Parametri SVM

Quando si utilizza un classificatore SVM, lineare o con particolari kernel, è possibile specificare alcuni parametri per regolare alcune caratteristiche del classificatore. L'algoritmo di cross-validation implementato nel framework AfL, testa il classificatore su due parametri: c e γ. La figura 3.3.4 mostra la differenza nella scelta di diversi valori di questi due parametri. Guardando le immagini per riga, osserviamo la differenza dovuta al variare del parametro c: al crescere di c tendiamo ad essere meno tolleranti agli outlier. Per colonna vediamo la differenza legata al variare dei γ (parametro legato al kernel rbf): il modello tende, al crescere di γ, ad adattarsi maggiormente ai dati.

3.3.3 Output classificazione

Lo schema di cross-validation descritto è applicato ad ogni nuovo dataset ottenuto a seguito della compressione delle serie temporali e del clu-

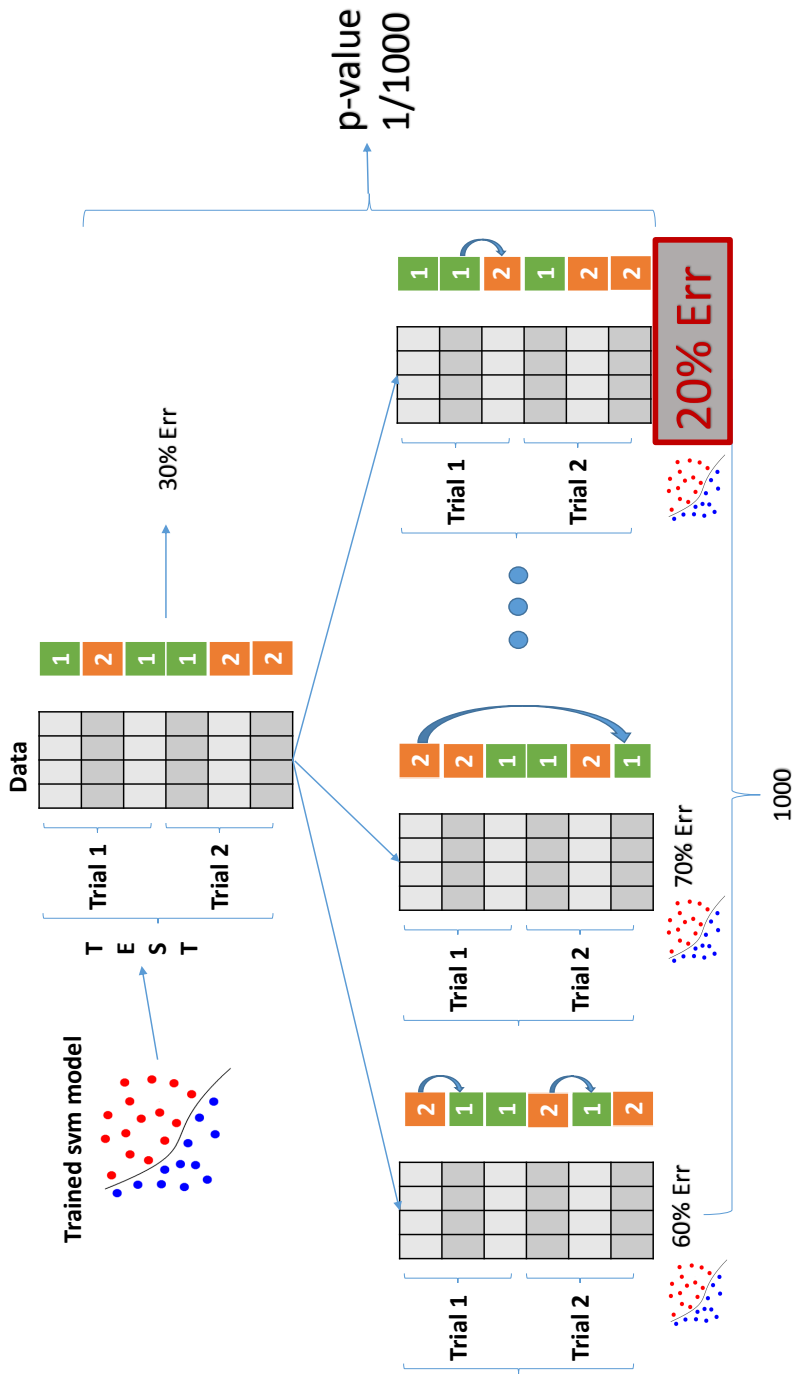

Figura 3.3.2: Test delle permutazioni sulle label del test set.

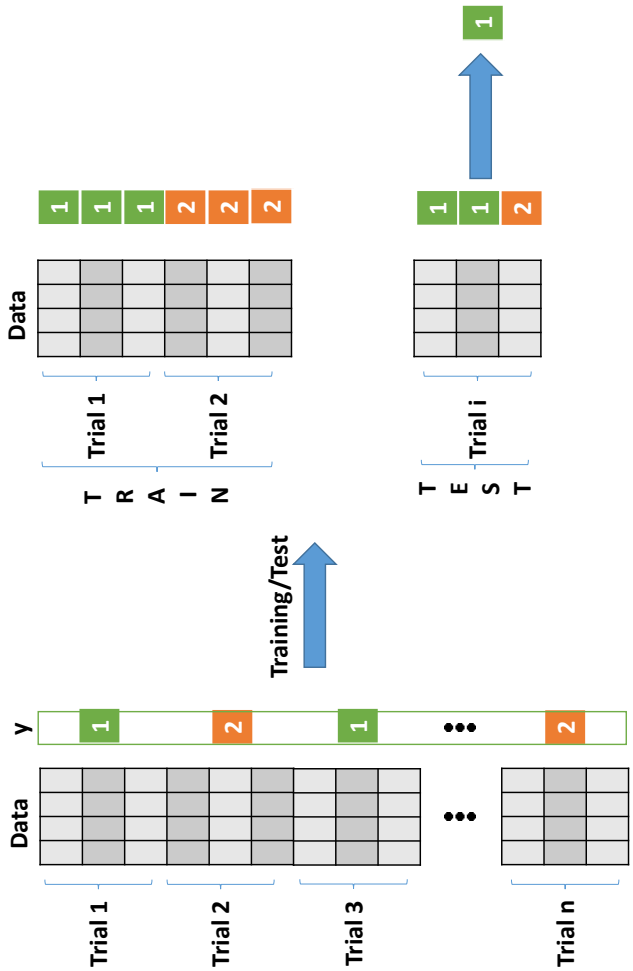

Figura 3.3.3: Schema di classificazione a votazione.

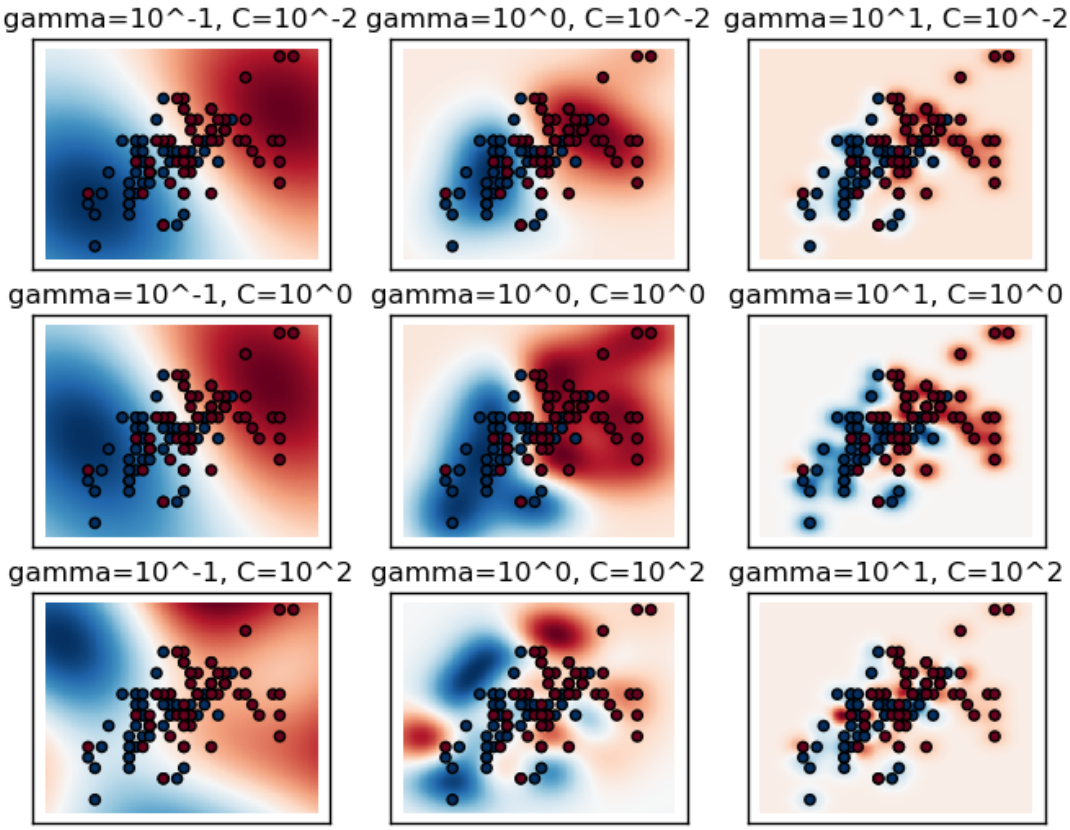

Figura 3.3.4: Influenza dei parametri c e γ sul classificatore SVM.

stering. Un singolo output dello step di classificazione è rappresentato da una serie di informazioni per ogni iterazione della cross-validation (6 sono le iterazioni totali, una per ogni fold). In particolare, *per ogni iterazione*, le seguenti sono le informazioni più importanti:

- modello SVM addestrato
- errore in cross-validation (training/validation)
- errore one shot
- informazioni sul fold utilizzato
- fature selezionate
- p-value del test delle permutazioni

Considerando le diverse scelte per:

- il numero di cluster
- due differenti tipologie di algoritmi di clustering utilizzate
- due differenti metriche di dissimilarità
- due tipi di SVM utilizzati (lineare ed rbf)

il numero di risultati generati da questo step rende difficile la ricerca del miglior risultato in maniera manuale. La ricerca del risultato ottimale tra tutti quelli generati è stata quindi automatizzata con uno script matlab.

Lo script filtra tra i risultati quelli con:

- media dei 6 errori one-shot più bassa
- almeno 4/6 p-value relativi al test delle permutazione significativi (soglia di significatività fissata a 0.05)

3.3.4 Analisi cluster migliori

Ultima tipologia di analisi implementata nel framework fornisce la possibilità di analizzare ulteriormente gli output migliori (filtrati con il metodo descritto in 3.3.3). Tra le informazioni dell'output migliore abbiamo la history delle features selezionate per ogni partizionamento train/test.

Ogni feature, come descritto in precedenza, è il medoid di un cluster di voxel. Il numero di voxel selezionati dipende dal numero di cluster e dalla percentuale di features selezionate dall'algoritmo di cross-validation (di default il 20% delle features). Anche se i voxel scelti rappresentano un insieme più grande di voxel, spesso si preferisce ottenere un modello che lavora su più voxel o sulla media di un gruppo più grande di voxel.

Un ulteriore step implementato nel framework lavora sul risultato ottimale, applicando i seguenti step:

- A partire dal risultato migliore:
  - ricerca il cluster selezionato più volte nei 6 errori one-shot
  - suddividi il cluster in regioni contigue (dal punto di vista spaziale)

- ricerca la regione contigua *mediamente più correlata* al centroide del cluster iniziale
- utilizzo della media degli elementi della regione per una nuova cross-validation

Purtroppo, come verrà mostrato nei risultati, questa ulteriore considerazione non ha portato miglioramenti con i tre soggetti in esame.

Capitolo 4

Risultati

Introduzione

In questo capitolo sono riportati e descritti i risultati ottenuti utilizzando il framework su dataset appartenenti a tre soggetti, etichettati con *AZ, GV* e *JE*. Per ogni soggetto sono state testate tutte le metodologie di compressione delle serie temporali (media aritmetica, GLM su tre basi e PCA non lineare).

Verranno mostrati gli errori di classificazione, le mappe dei voxel selezionati dalla feature selection e la stima della risposta emodinamica relativa ai medoids dei cluster selezionati.

4.1 Risultati classificazione

I risultati migliori di classificazione, filtrati secondo i criteri descritti nella sottosezione 3.3.3, sono riportati in figura 4.1.1. Per ogni soggetto sono riportati due errori: il primo (barra verde) è la media dei sei errori one-shot (uno per ogni fold); l'altro (barra gialla) è l'errore medio dei soli errori one-shot con p-value (relativo al test delle permutazioni) significativo. È facile notare che, eliminando dai 6 errori one-shot quelli con p-value non significativo, l'errore di classificazione si abbassa. Questo perché laddove il modello generato non supera il test delle permutazioni, l'errore one-shot è sicuramente più alto. Un p-value non significativo indica che il modello generato (per un particolare fold) riesce a fare meglio con dati casuali.

Come si evince dalla figura 4.1.1, per 2 soggetti la metodologia di compressione delle serie temporali con la media aritmetica ha portato all'errore di classificazione più basso. Per tutti i soggetti invece è stato sempre migliore l'SVM non lineare (kernel rbf).

La distribuzione degli errori sui 6 fold è riportata in figura 4.1.2.

4.2 Mappe ed emodinamica

Per ogni soggetto si ha quindi a disposizione il risultato migliore filtrato in maniera automatica dal framework. Avendo per tale risultato l'informazione sulle features selezionate ad ogni iterazione della cross-validation (medoid di un particolare cluster) e sulla tipologia

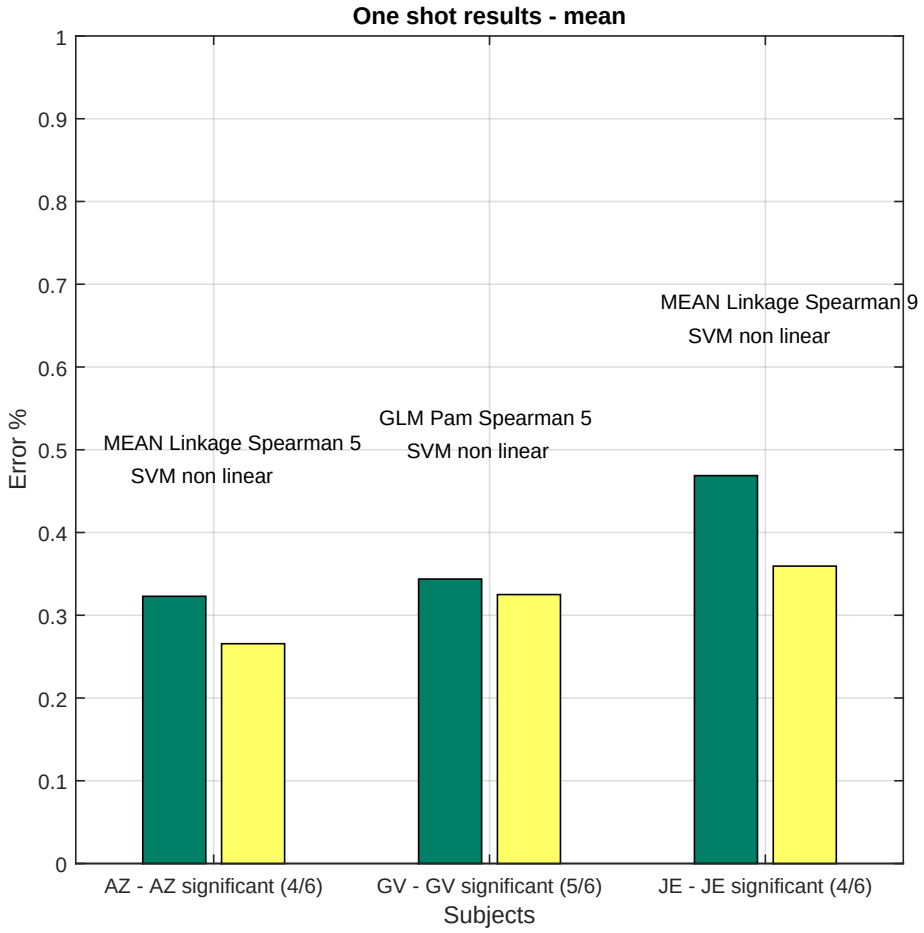

Figura 4.1.1: Errore medio one-shot su 6 fold per i vari soggetti.

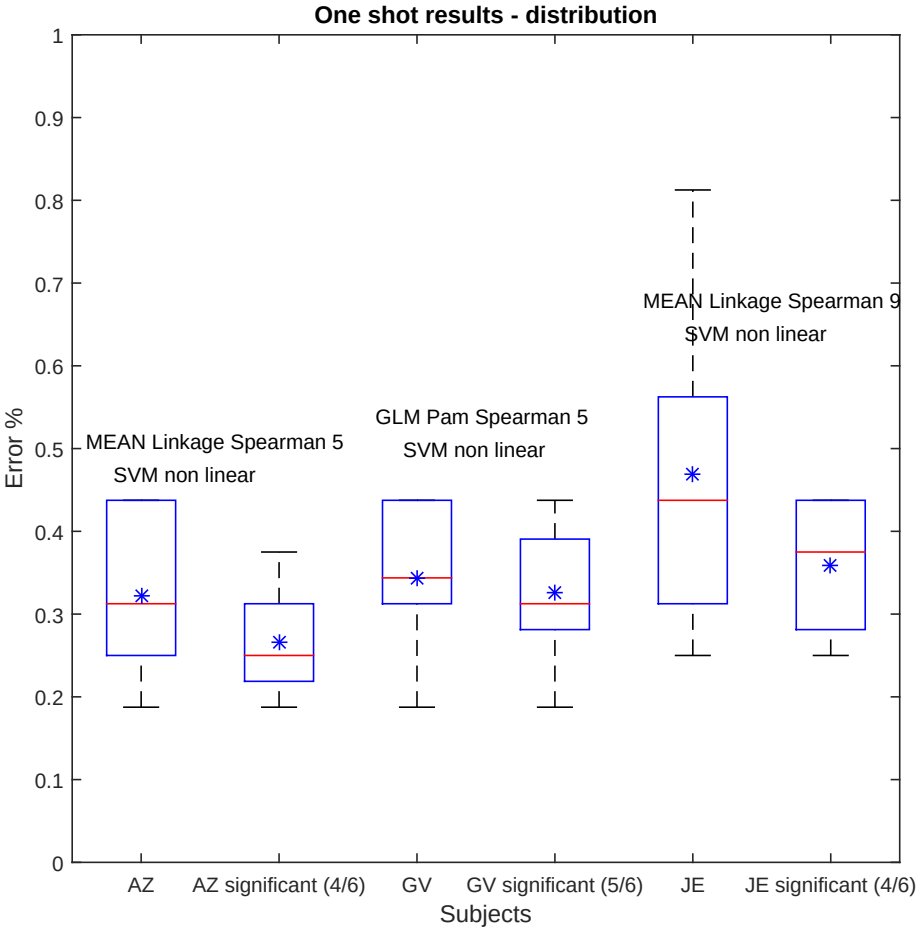

Figura 4.1.2: Distribuzione errori one-shot su 6 fold per i vari soggetti.

di clustering utilizzata, è possibile generare una mappa con i voxel appartenenti al cluster.

Avendo 6 iterazioni di cross-validation, ciò che accade è che per ogni fold l'algoritmo di cross-validation può selezionare delle features diverse. Nei casi migliori dei soggetti AZ, GV e JE si verifica (nella maggior parte dei casi) che una feature selezionata in un fold viene riselezionata nel successivo. Di conseguenza ha senso visualizzare la mappa relativa alla feature (e quindi al cluster) selezionato più volte in cross-validation.

Per il soggetto AZ il risultato migliore è stato ottenuto con dataset così processati:

- compressione serie temporali con media aritmetica
- algoritmo di clustering gerarchico
- metrica di distanza di Spearman
- numero di cluster pari a 5
- SVM rbf

Per tale soggetto, in 4 iterazioni su 6 l'algoritmo di cross-validation ha selezionato come feature migliore il medoid del cluster 2. I voxel contenuti nel cluster 2 sono mostrati in figura 4.2.1. La risposta emodinamica, per ogni medoid del cluster, è mostrata in figura 4.2.2.

Per il soggetto GV il risultato migliore è stato ottenuto con dataset così processati:

- compressione serie temporali con GLM su tre basi

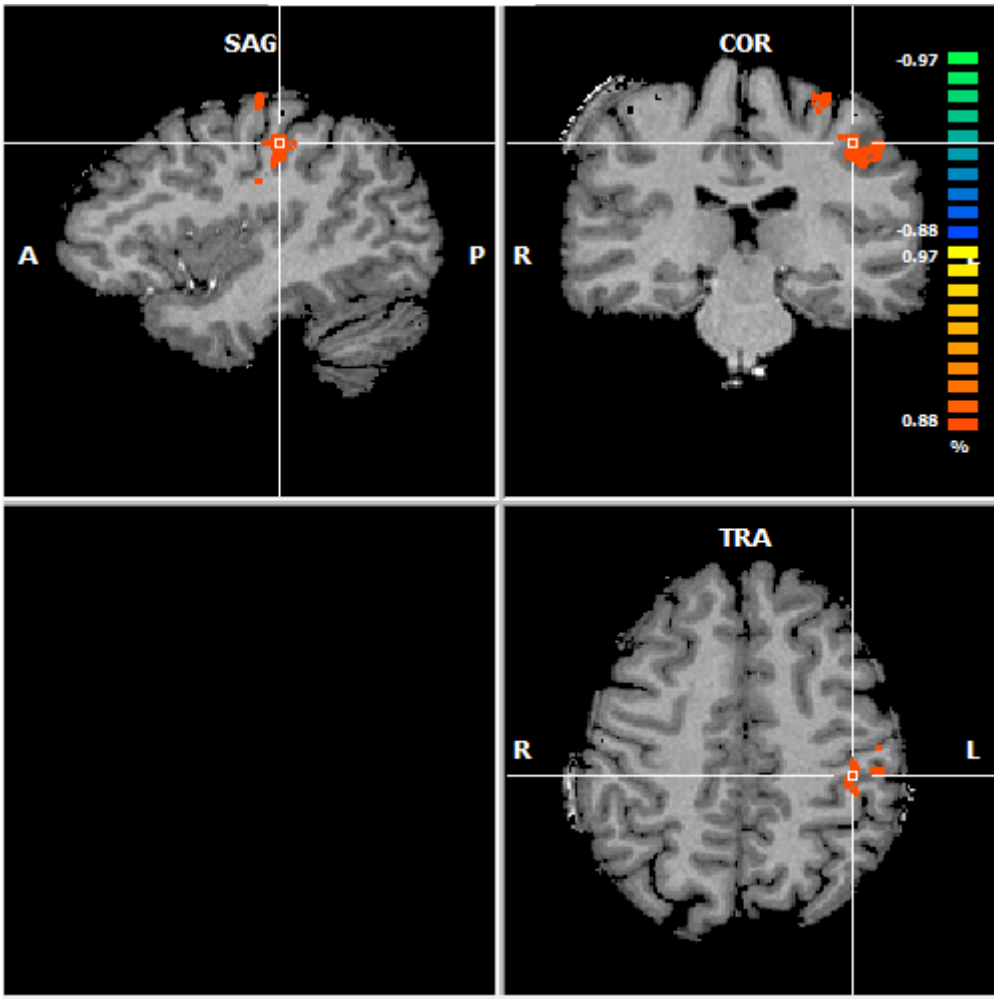

Figura 4.2.1: Cluster selezionato in fase di classificazione per il soggetto AZ.

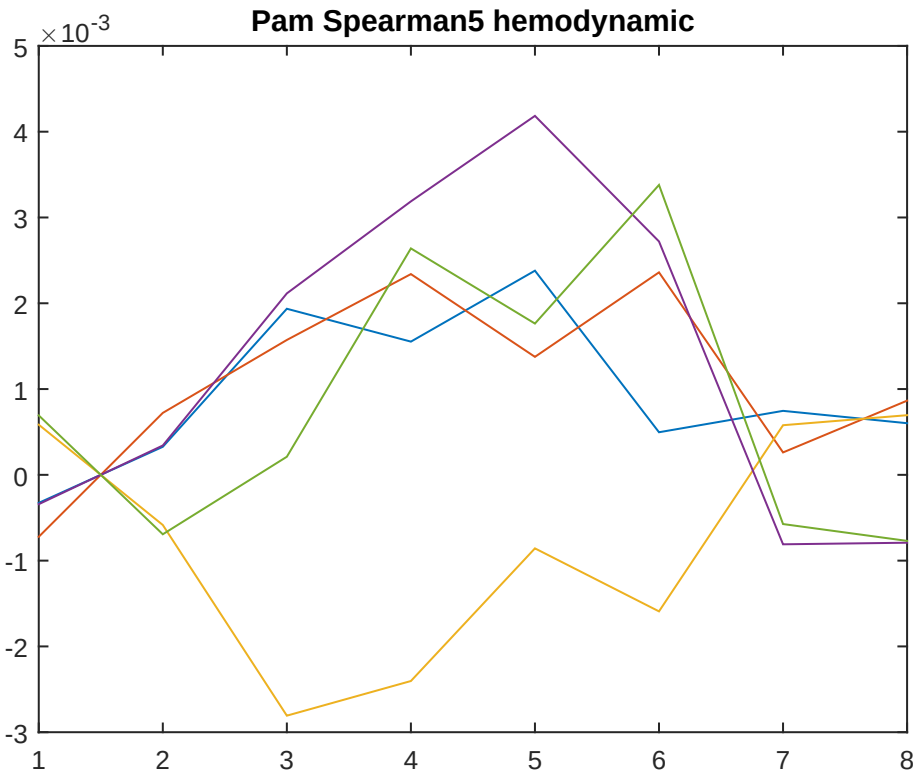

Figura 4.2.2: Risposte emodinamiche relative al clustering migliore del soggetto AZ.

- algoritmo di clustering PAM
- metrica di distanza di Spearman
- numero di cluster pari a 5
- SVM rbf

Anche per il soggetto GV, in 4 iterazioni su 6 l'algoritmo di cross-validation ha selezionato come feature migliore il medoid del cluster 3. I voxel contenuti nel cluster 3 sono mostrati in figura 4.2.3. La risposta emodinamica, per ogni medoid del cluster, è mostrata in figura 4.2.2.

Infine, per il soggetto JE il risultato migliore è stato ottenuto con dataset così processati:

- compressione serie temporali con media aritmetica

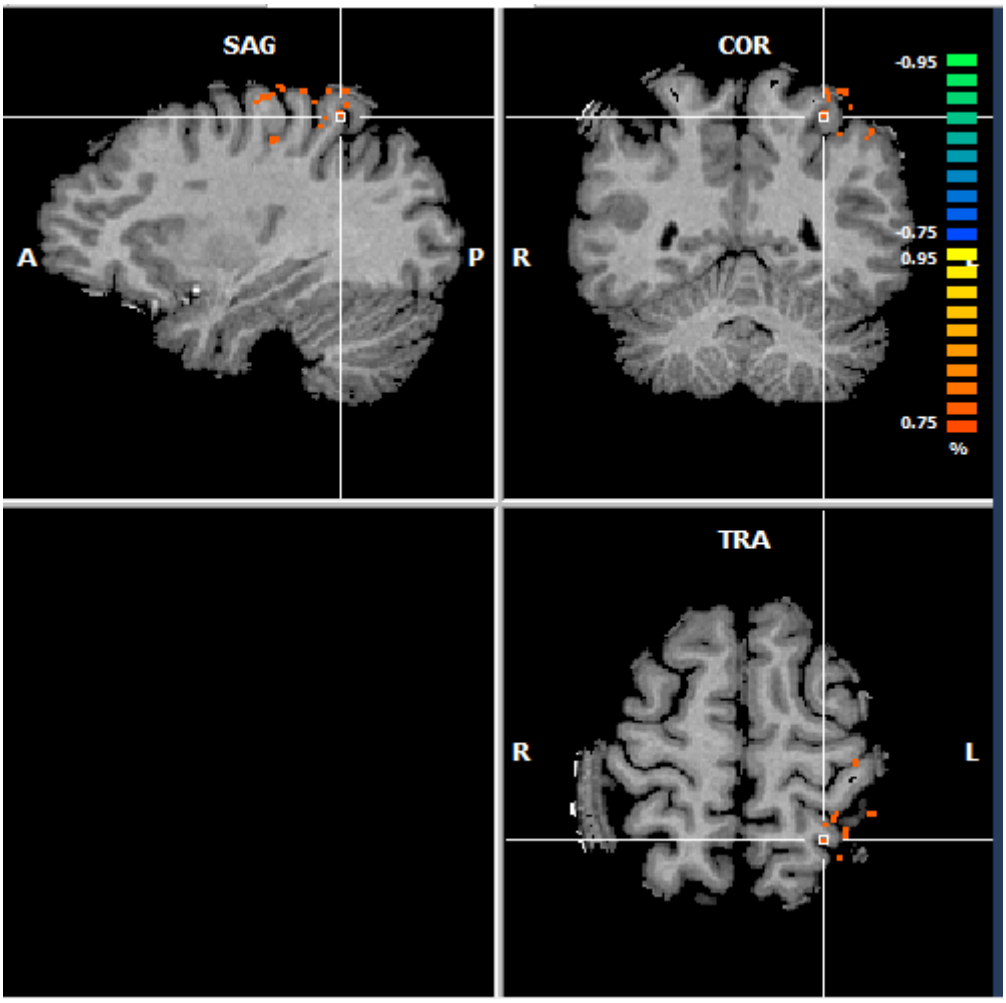

Figura 4.2.3: Cluster selezionato in fase di classificazione per il soggetto GV.

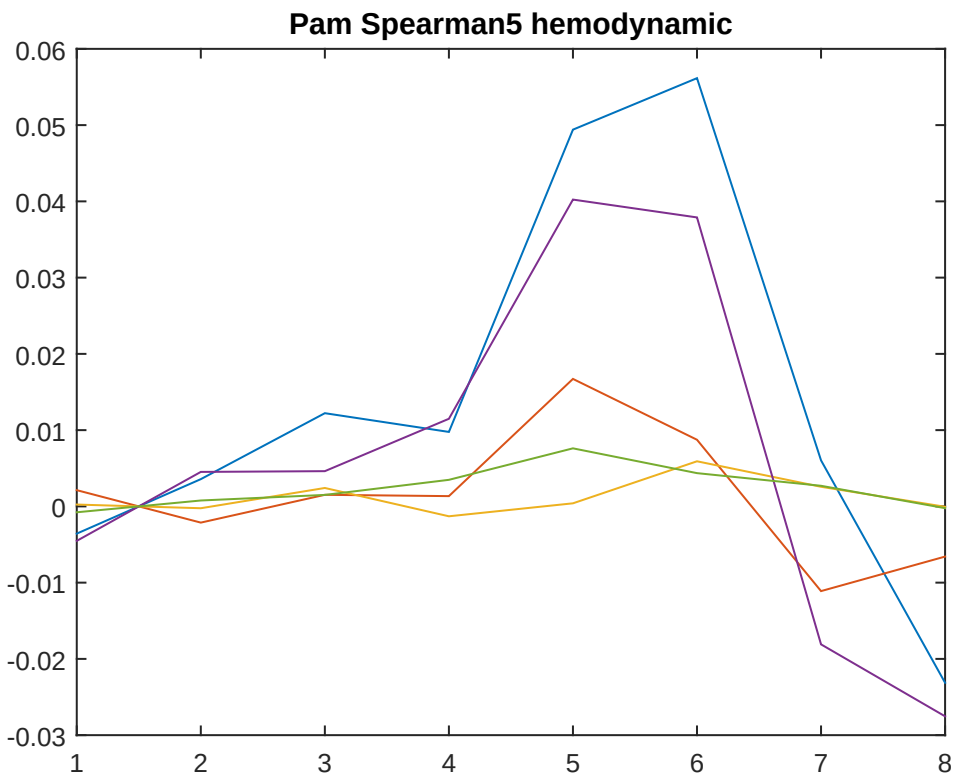

Figura 4.2.4: Risposte emodinamiche relative al clustering migliore del soggetto GV.

- algoritmo di clustering gerarchico
- metrica di distanza di Spearman
- numero di cluster pari a 9
- SVM rbf

Per tale soggetto, in 2 iterazioni su 6 l'algoritmo di cross-validation ha selezionato come feature migliore il medoid del cluster 4. I voxel contenuti nel cluster 4 sono mostrati in figura 4.2.5. La risposta emodinamica, per ogni medoid del cluster, è mostrata in figura 4.2.6.

4.2.1 Analisi dei cluster migliori

I risultati migliori per ogni soggetto mostrati nella sezione 4.2 sono stati ulteriormente analizzati. L'ultimo step implementato nel framework (come descritto nella sottosezione 3.3.4) parte proprio dai risultati migliori e costruisce una nuova feature con la quale riaddestra un modello e rivaluta l'errore di classificazione.

Più che cercare di ridurre ulteriormente gli errori di classificazione (già soddisfacenti), quest'ultimo step vuole rendere i risultati più appetibili da un punto di vista concettuale; si ricorda infatti che gli errori ottenuti fino a questo step sono stati raggiunti utilizzando come features i medoids dei vari cluster. Avendo effettuato una features selection sui medoids con lo schema descritto nella sottosezione 3.3.1, spesso vengono selezionate poche features. Spesso si tende a valorizzare modelli che raggiungono una buona accuratezza valutando un set

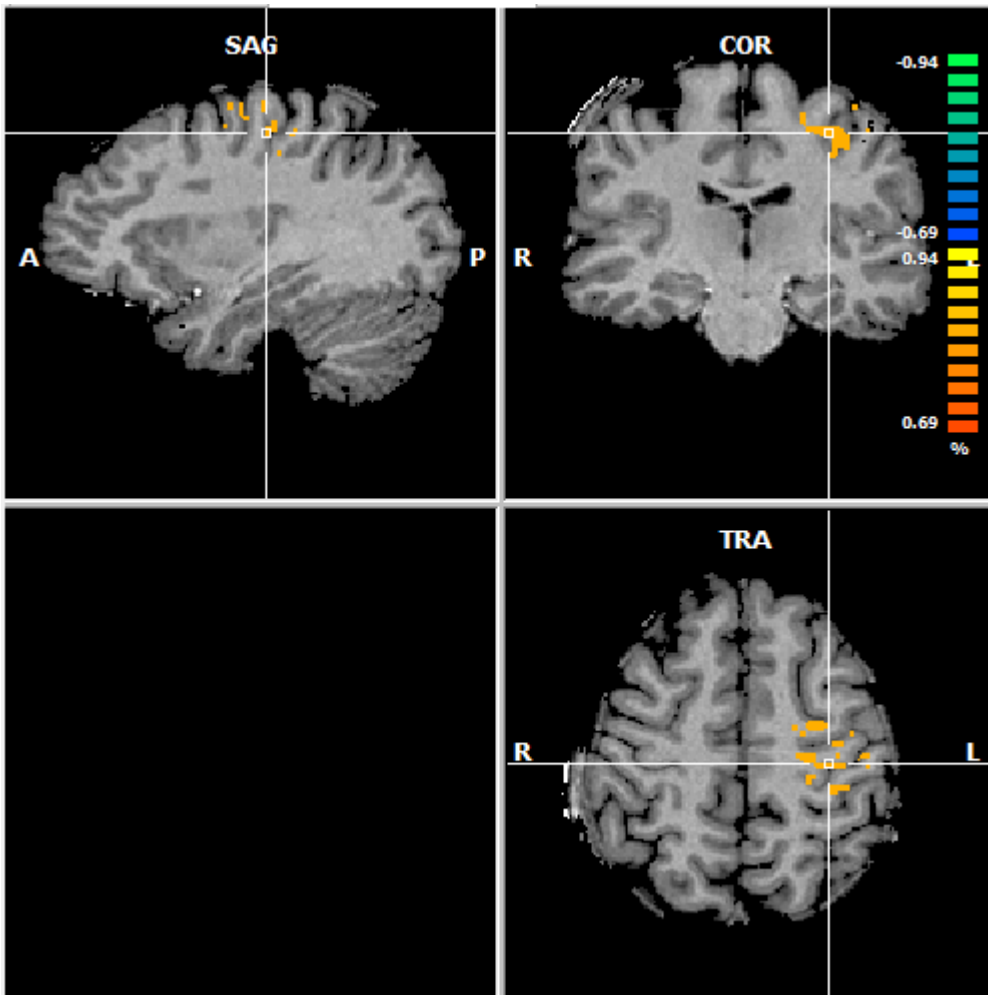

Figura 4.2.5: Cluster selezionato in fase di classificazione per il soggetto JE.

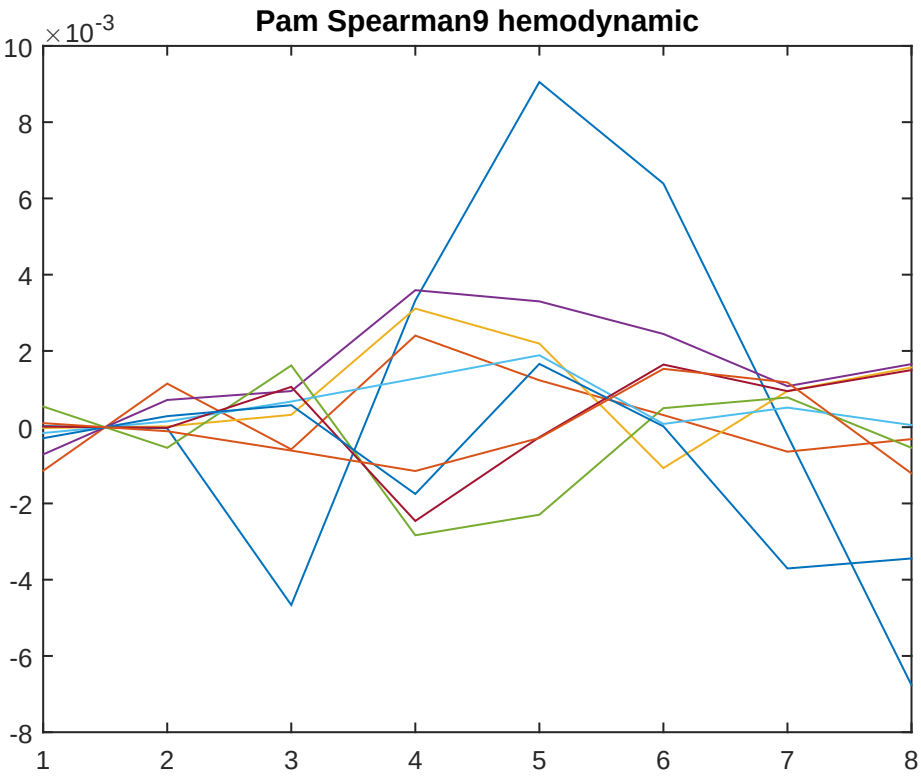

Figura 4.2.6: Risposte emodinamiche relative al clustering migliore del soggetto JE.

più grande di voxel. A partire dalle features migliori (e quindi dai *cluster migliori*), è stato applicato lo schema descritto in 3.3.4.

I risultati ottenuti sono più interessanti da un punto di vista concettuale ma, purtroppo, portano un lieve peggioramento degli errori di classificazione (vedi figura4.2.7). La distribuzione di tali errori è mostrata in figura 4.2.8.

I cluster ottenuti, come ci si aspetta, sono più compatti da un punto di vista morfologico (vedi figure 4.2.9, 4.2.10 e 4.2.11).

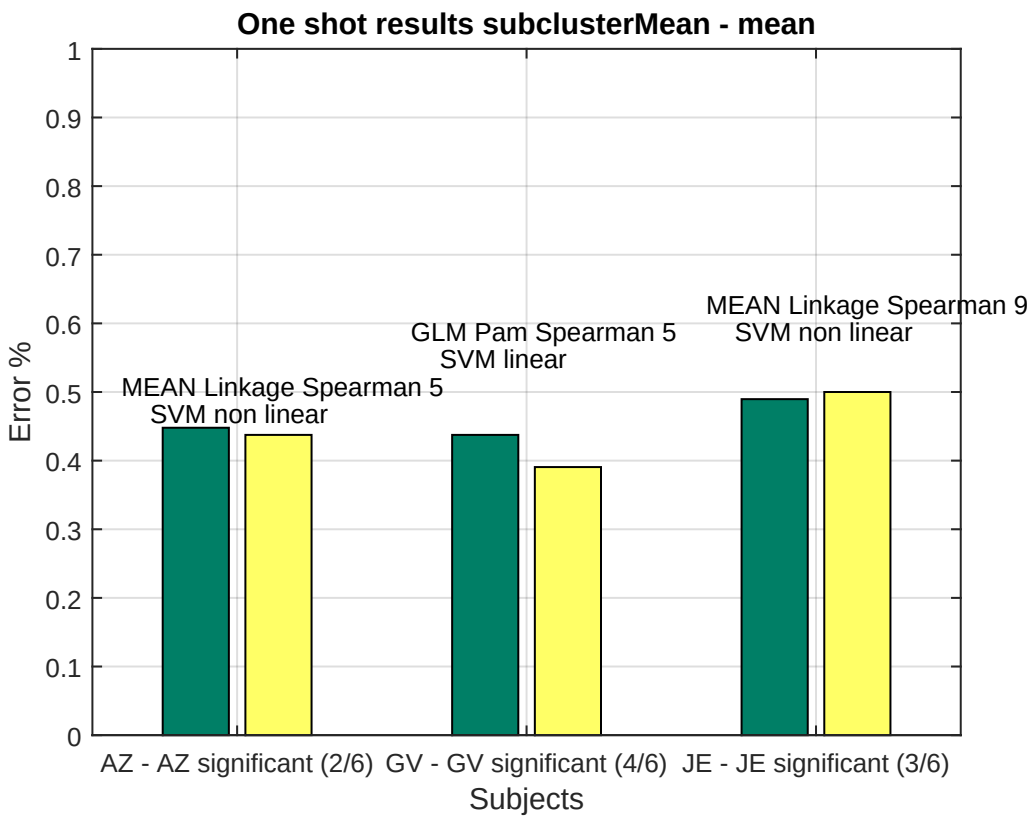

Figura 4.2.7: Errore medio one-shot su 6 fold per i vari soggetti.

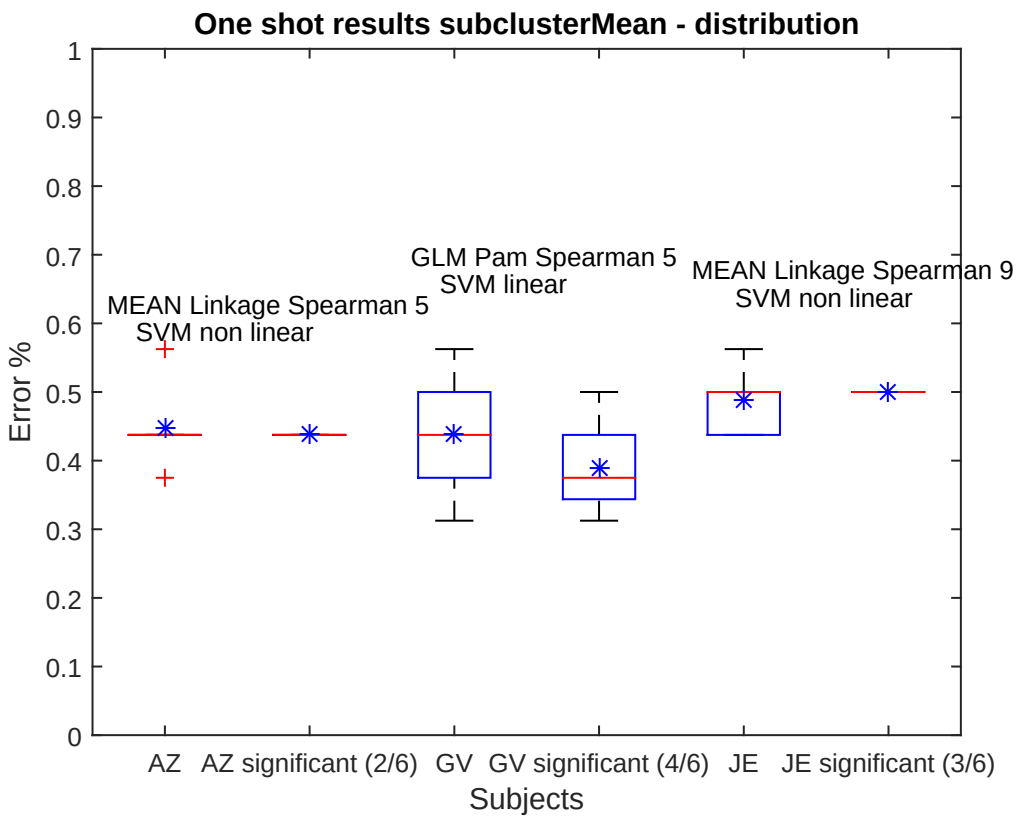

Figura 4.2.8: Distribuzione errori one-shot su 6 fold per i vari soggetti.

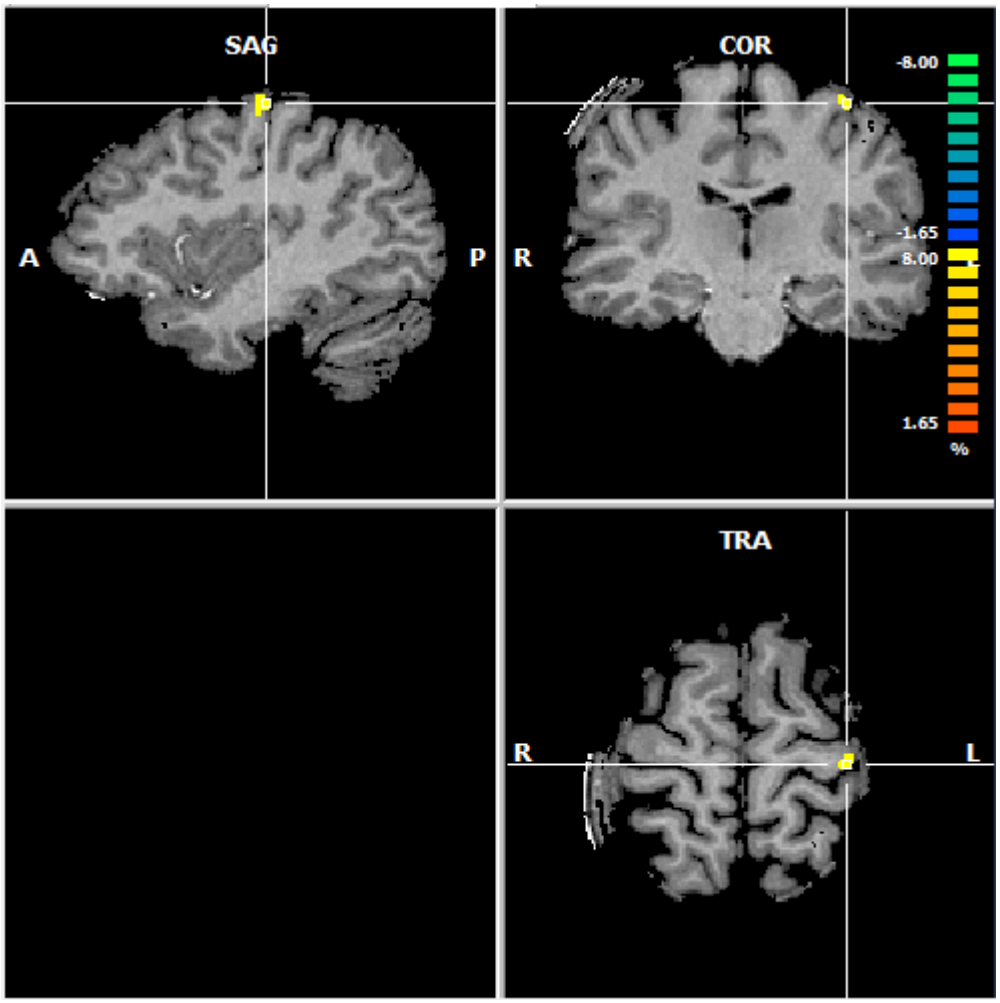

Figura 4.2.9: Subcluster soggetto AZ.

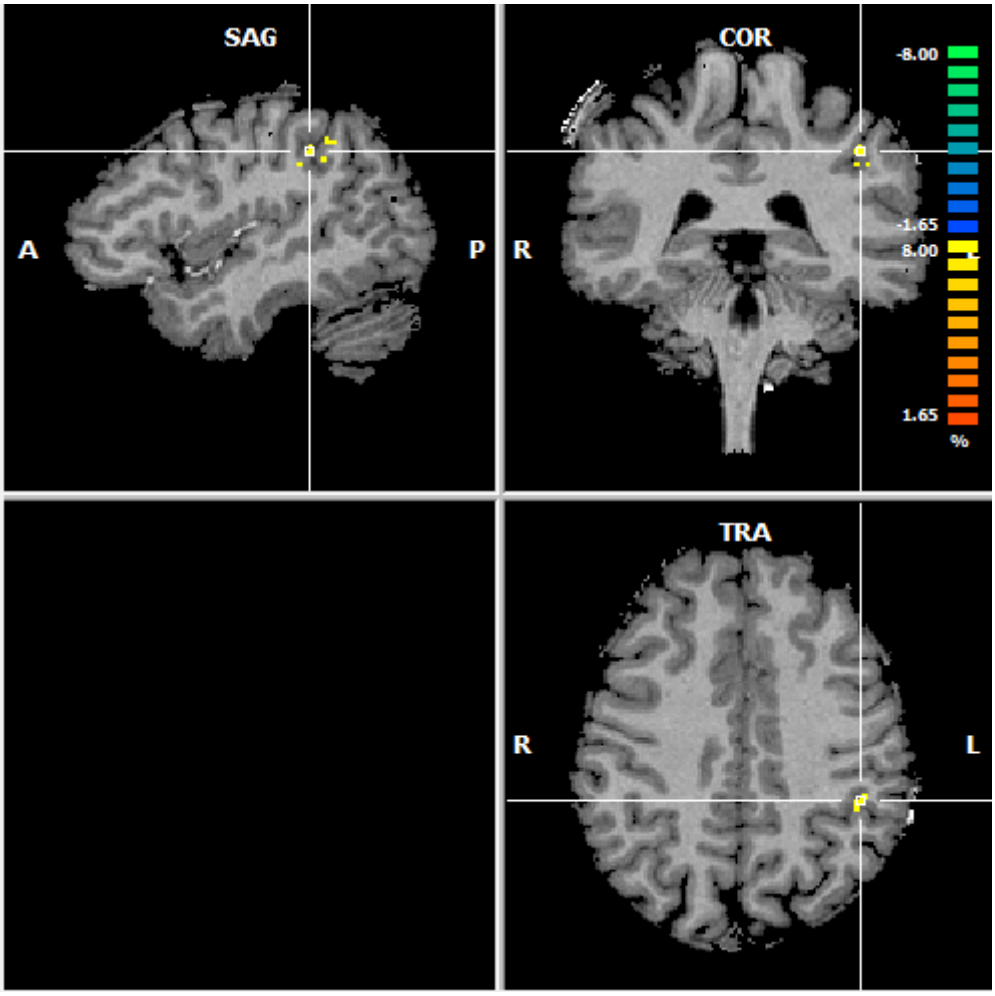

Figura 4.2.10: Subcluster soggetto AZ.

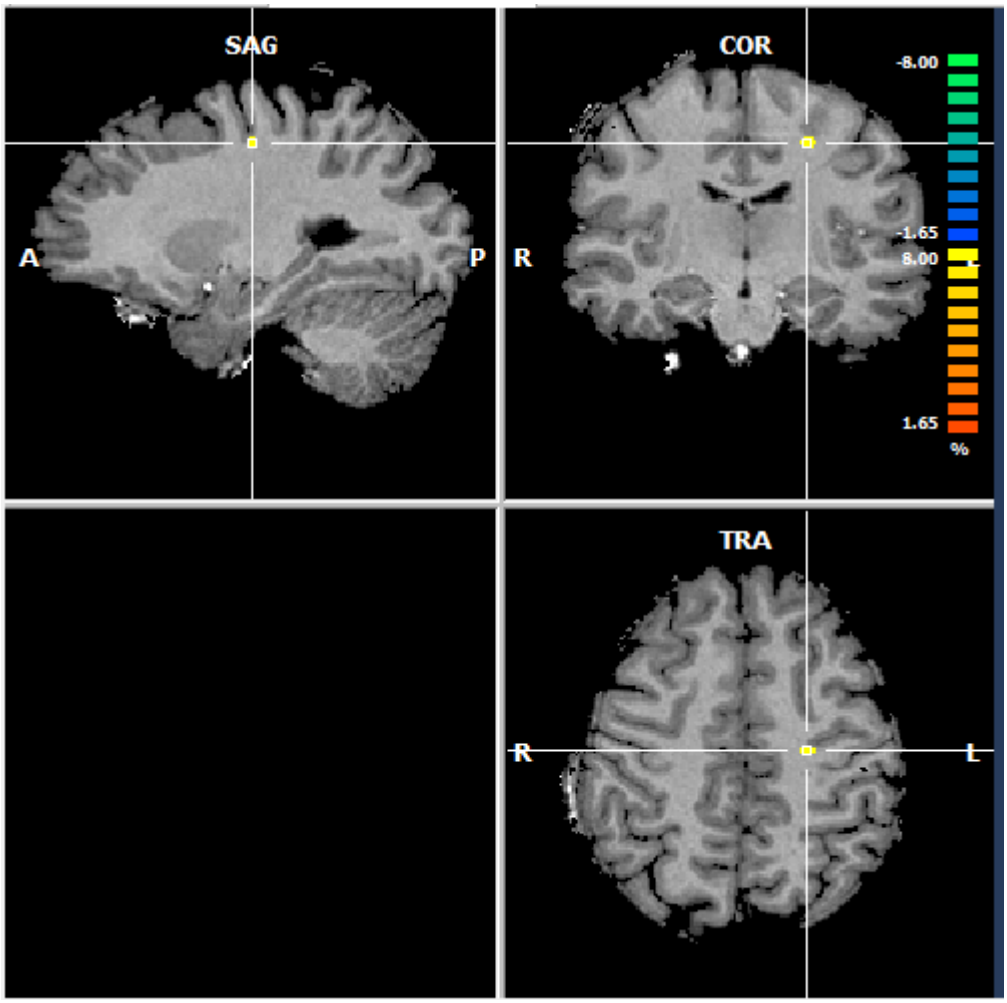

Figura 4.2.11: Subcluster soggetto AZ.

Capitolo 5

Conclusioni

Il lavoro di tesi propone una pipeline generale per trattare dati di risonanza magnetica funzionale. La pipeline formalizza una serie di elaborazioni da applicare ai dati di input in un ordine ben definito, dando all'utente possibilità di scelta su alcune decisioni.

Partendo da un dataset fMRI relativo ad un particolare esperimento e conoscendo il design dell'esperimento (label), è possibile utilizzare la pipeline proposta ed ottenere:

- Raggruppamento dei voxel in cluster
- Mappe dei cluster
- Stima della risposta emodinamica
- Modello addestrato
- Errore di classificazione

I punti di forza della pipeline sono sicuramente le differenti modalità proposte per la compressione delle serie temporali (media aritmetica in un intervallo definito dall'utente, PCA non lineare, GLM su tre basi), lo schema di cross-validation e l'analisi dei cluster migliori (vedi sottosezione 3.3.4).

La riduzione delle dimensionalità iniziale, che va dalla compressione delle serie temporali al raggruppamento dei voxel, rende possibile l'utilizzo di uno schema di cross-validation abbastanza oneroso da un punto di vista computazionale. Tale schema, infatti, ottimizza i parametri del classificatore SVM e ricerca il miglior sottoinsieme di features utilizzando una forward selection. I risultati di classificazione ottenuti sono soddisfacenti e riescono a generare, per due dei tre soggetti analizzati, errori di classificazione inferiori al 35%.

Bibliografia

[1] S M SMITH, «Overview of fMRI analysis», The British Journal of Radiology 2004

[2] Kenneth A. Norman et al., Beyond mind-reading: multi-voxel pattern analysis of fMRI data, TRENDS in Cognitive Sciences (2006), doi:10.1016/j.tics.2006.07.005

[3] Bishop Christopher, Pattern Recognition and Machine Learning, ISBN 978-0-387-31073-2, pp. 561-563

[4] Bishop Christopher, Pattern Recognition and Machine Learning, ISBN 978-0-387-31073-2, pp. 179-180

[5] Matthias Scholz, Martin Fraunholz and Joachim Selbig., Principal Manifolds for Data Visualization and Dimension Reduction, Volume 58 of LNCSE, pages 44-67. Springer Berlin Heidelberg, 2007.