Threshold-Free Cluster Enhancement

Luigi Giugliano[1], Marco Mecchia[1]

[1]Università degli studi di Salerno

23 maggio 2016

Overview

Overview

Sogliatura delle immagini statistiche

Sogliatura statistica

Generalmente, in statistica la sogliatura é un processo che permette di visualizzare i risultati di un esperimento maggiori di una soglia scelta.

Una soglia ben scelta consente di visualizzare solo i risultati piú significativi di un esperimento, eliminando in parte il rumore.

Spatial information enhancing

Lo spatial information enhancing é una tecnica che:

- utilizza di informazioni spaziali per aumentare l'autenticità di estese aree di segnale.
- le regioni del segnale sono infatti più estese del rumore e quindi trovare tali zone aumenta la possibilità che esse siano segnale e non artefatti.

Cluster-based Thresholding

Il cluster-based thresholding é l'approccio piú comune in neuroimaging:

- Consiste nel visualizzare solo i voxel che fanno parte di aree la cui estensione é $\geq$ di una soglia fissata.

Problemi:

- Necessitá di definire una soglia di clustering.
- Sogliatura di tipo *hard*.
- Difficoltá nel riconoscimento di eventuali *subcluster*.

Overview

TFCE

TFCE tenta di superare i problemi degli approcci precedenti.

- Input: Un immagine statistica non processata.
- Output: Un immagine statistica in cui il valore di ogni voxel é un punteggio che rappresenta il contributo spaziale del cluster di cui fa parte.

- Clustering dell'immagine intrinseco.

Assegnazione dei punteggi (1/2)

Il punteggio del voxel p viene stabilito dalla seguente formula:

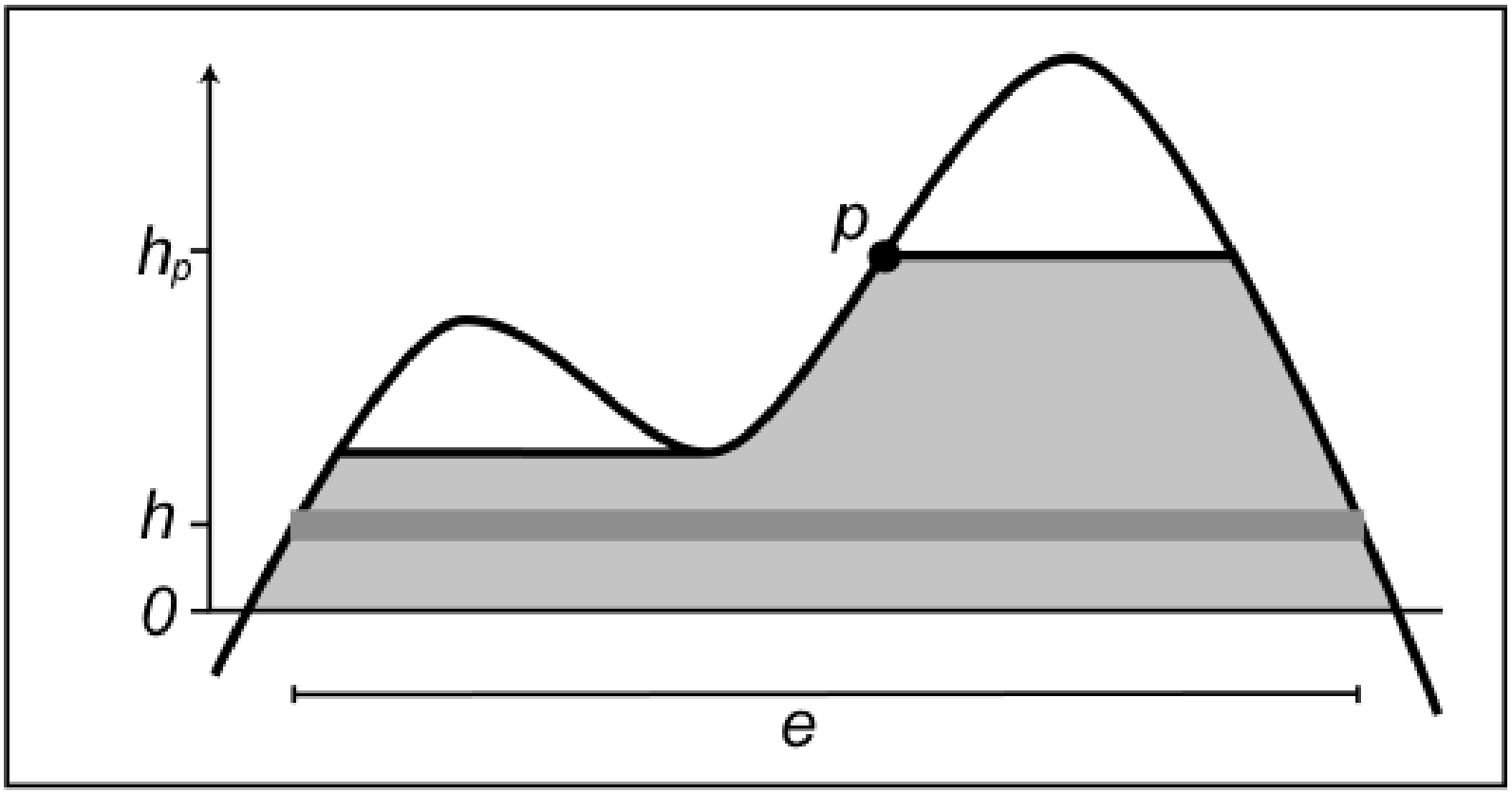
$$TFCE(p) = \int_{h=h_0}^{h_p} e(h)^E h^H dh$$

dove:

- h_p é il valore statistico del voxel p.
- $e(h)$ é l'area del cluster ad altezza h.
- E ed H sono costanti.

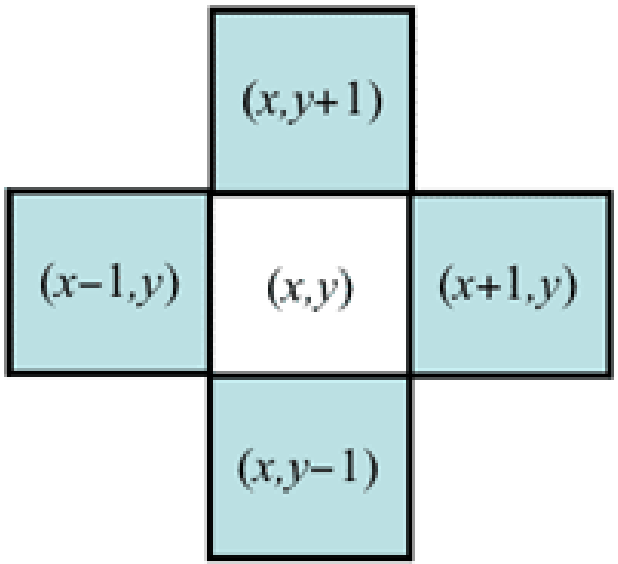
Questo integrale viene calcolato approssimandolo con una sommatoria ponendo $dh = 0.1$.

Assegnazione dei punteggi (2/2)

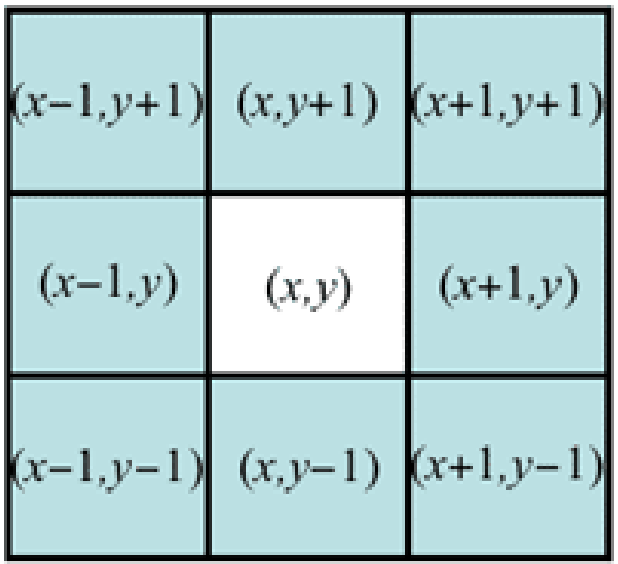


Calcolo estensione cluster (1/2)

- Il calcolo dell'estensione del cluster nel caso di immagini tridimensionali risulta essere piú complesso.
- Occorre controllare il vicinato di ogni voxel in base alla 26 connectivity.



4-neighbourhood



8-neighbourhood

Calcolo estensione cluster (2/2)

La nostra implementazione consiste in un semplice algoritmo:

1. Viene generata, a partire dall'immagine statistica, una mappa binaria in base alla soglia h corrente.
2. Una visita in ampiezza della mappa binaria etichetta tutti i cluster.

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 1 & 0 & 3 & 3 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 2 & 2 \end{bmatrix}$$

3. Per ogni voxel della mappa, il valore $e(h)$ é il numero di elementi presenti nel cluster di cui fa parte.

$$\begin{bmatrix} 1 & 1 & 0 & 3 & 3 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 2 & 2 \end{bmatrix} \rightarrow \begin{bmatrix} 5 & 5 & 0 & 2 & 2 \\ 5 & 5 & 0 & 0 & 0 \\ 5 & 0 & 0 & 2 & 2 \end{bmatrix}$$

Scelta dei parametri E ed H (1/2)

Ricordando la formula di assegnazione dei punteggi:

$$TFCE(p) = \int_{h=h_0}^{h_p} e(h)^E h^H dh$$

- La scelta dei parametri E ed H risulta essere cruciale per avere dei punteggi coerenti.
- Tali parametri sono stati scelti in modo da adattarsi ad un ampio set di segnali e rumore.

Scelta dei parametri E ed H (2/2)

La scelta finale é stata $H = 2$ ed $E = 0.5$ poiché:

- Scegliere $H > 1$ ha il risultato di far si che gli score scalino piú che linearmente con l' "altezza" dei cluster.
  - Ció é desirabile in quanto vengono favoriti cluster di intensitá molto alta rispetto a quelli con intensitá piú bassa.
- Scegliere $E < 1$ fa si che il risultati scali meno che linearmente con la "larghezza" dei cluster.
  - Ció é desirabile specie con h molto basso poiché é probabile che ci siano pochi cluster di dimensioni molto grandi, che forniscono poca informazione spaziale.

Sogliatura esplicita

L'algoritmo TFCE produce un'immagine con gli score calcolati sull'immagine originale.

Tuttavia, l'immagine prodotta manca di valenza stastica.

Per calcolare la valenza statistica dell'immagine degli score, occorre calcolare i p-value per ogni voxel.
Essendo TFCE una particolare statistica che sfrutta informazioni spaziali, é possibile calcolare i p-value effettuando il test delle permutazioni sull'esperimento originale.

Overview

Metodo di montecarlo per la sogliatura basata su cluster

Bonferroni

In statistica la **correzione di Bonferroni** viene utilizzata per contrastare il problema dei confronti multipli.

E' basata sull'idea che se in un esperimento si stanno testando m ipotesi, un modo per mantenere il **familywise error rate** (**FWER** = probabilità di effettuare errore di *Tipo 1* all'inverno delle ipotesi) è quello di testare ogni ipotesi individualmente con una significanza statistica di $1/m$ moltiplicato per il livello massimo desiderato.

Quindi se vogliamo un *p-value* totale di α, la correzione di Bonferroni testerà ogni singolo esperimento con un *p-value* di α/m è rifiuterà l'ipotesi nulla se il *p-value* di quell'esperimento è minore di tale valore.

False Discovery Rate

Il **False Discovery Rate** come la correzione di Bonferroni si prefigge l'obiettivo di contrastare il problema dei confronti multipli.

La procedura per il controllo FDR è stata creata per gestire la proporzione attesa di rifiuto dell'ipotesi nulla, che però sarebbe stato sbagliato rifiutare (false discoveries).

La procedura FDR fornisce un controllo meno stringente sugli errori di *Tipo 1* rispetto a Bonferroni.

Sia:

- V il numero di falsi positivi (Errori di Tipo 1)
- S il numero di veri positivi
- $R = V + S$

La FDR è definita:

$$FDR = E\left[\frac{V}{S+V}\right] = E\left[\frac{V}{S+V}\right]$$

dove $\frac{V}{R} = 0$ quando $R = 0$.

Avendo $H_1 \dots H_m$ test sull'ipotesi nulla e $p_1 \dots p_m$ *p-value* corrispondenti. Ordiniamo i *p-value* in ordine crescente; il *p-value* più piccolo corrisponde al test con valore statistico più alto.

La procedura Benjamini–Hochberg controlla il false discovery rate (al livello α) con i seguenti passi:

1. Per un dato α, trova il più grande k per cui: $p_k \leq \frac{k}{m}\alpha$
2. Rifiuta l'ipotesi nulla (accetta come discovery vere) tutte i test $H_i \dot{H}_k$

La procedura di Benjamini–Hochberg è valida quando i m test sono indipendenti e soddisfa anche la seguente equazione:

$$FDR \leq \frac{m_0}{m}\alpha \leq \alpha$$

Overview

Suddivisione del codice

I file principali che compongono il plugin sono:

- Tfce.cpp
- Utilities.cpp

Tfce è il core del plugin, dove avviene il calcolo dei punteggi.

Utilities contiene tutte le funzioni di supporto.

Funzioni pubbliche (1/3)

L'unica funzione che viene esposta dal file **Tfce.h** è:

```
float * tfce_score(float * map, int dim_x, int dim_y,
    int dim_z, float E, float H, float dh);
```

Funzioni pubbliche (2/3)

Le funzioni che espone **Utilities.h** sono:

```
void findMinMax(float *map, int n, float *min, float
    *max, float * range);

int * getBinaryVector(float * map, int n, int
    (*confront)(float, float), float value, int *
    numOfElementsMatching);
```

Funzioni pubbliche (3/3)

```
float * fromBinaryToRealVector(float * map, int n, int
    * binaryVector);

float * fill0(int n);

void apply_function(float * vector, int n, float (*
    operation) (float a, float b), float argument);

int linearIndexFromCoordinate(int x, int y, int z, int
    max_x, int max_y);

void coordinatesFromLinearIndex(int index, int max_x,
    int max_y, int * x, int * y, int * z);

float * copyAndConvertIntVector(int * vector, int n);
```

Funzione tfce score

```
float * tfce_score(float * map, int dim_x, int dim_y,
    int dim_z, float E, float H, float dh){
findMinMax(map, n, &minData, &maxData, &rangeData);
precision = rangeData/dh;
if (precision > 200) {
increment = rangeData/200;
} else{
increment = rangeData/precision;
}
steps = ceil((maxData - minData) / (increment));
#pragma omp parallel for
for (i = 0; i < steps; i++) {
 computeTfceIteration(minData + i*increment, map, n,
     dim_x, dim_y, dim_z, E, H, dh, toReturn);
}
return toReturn;
}
```

Funzione computeTfceIteration (1/3)

```
void computeTfceIteration(float h, float * map, int n,
    int dim_x, int dim_y, int dim_z, float E, float H,
    float dh, float * toReturn){
int * indexMatchingData = getBinaryVector(map, n,
    moreThan, h, &numOfElementsMatching);
clustered_map = find_clusters_3D(indexMatchingData,
    dim_x, dim_y, dim_z, n, &num_clusters);
extent_map = new int[n];
for (j = 0; j < n; ++j){
extent_map[j] = 0;
}
delete [] indexMatchingData;
```

Funzione computeTfceIteration (2/3)

```
for (i = 1; i <= num_clusters; ++i) {
numOfElementsMatching = 0;
for (j = 0; j < n; ++j){
    if(clustered_map[j] == i)
        numOfElementsMatching++;
}
for (j = 0; j < n; ++j) {
   if(clustered_map[j] == i)
       extent_map[j] = numOfElementsMatching;
}
}
```

Funzione computeTfceIteration (3/3)

```
clustered_map_float =
    copyAndConvertIntVector(extent_map, n);
apply_function(clustered_map_float, n, elevate, E);
apply_function(clustered_map_float, n, multiply,
    pow(h, H));
apply_function(clustered_map_float, n, multiply, dh);
for (i = 0; i < n; ++i) {
#pragma omp atomic
    toReturn[i] += (clustered_map_float[i]);
}
delete[] clustered_map_float;
delete[] clustered_map;
delete[] extent_map;
```

Funzione getBinaryVector

Questa funzione emula il risultato del costrutto Matlab (matrice <condizione> valore).

```
int * getBinaryVector(float * map, int n, int
    (*confront)(float, float), float value, int *
    numOfElementsMatching){
    int * binaryVector = new int [n];
    (*numOfElementsMatching) = 0;
    int i;
    for (i = 0; i < n; ++i) {
        if(confront(map[i],value)){
            binaryVector[i] = 1;
            (*numOfElementsMatching)++;
        }
        else
            binaryVector[i] = 0;
    }
    return binaryVector;
}
```

Calcolo dell'estensione dei cluster

La funzione **find_cluster_3D**:

```
int * find_clusters_3D(int * binaryVector, int dim_x,
    int dim_y, int dim_z, int n, int * num_clusters)
```

restituisce la mappa dei cluster trovati utilizzando la **26-connectivity** nell'immagine binaria fornita in input.

E' stato utilizzata la specifica OpenMP per rendere il calcolo degli score più veloce.

OpenMP (Open Multiprocessing) è un API multipiattaforma per la creazione di applicazioni parallele su sistemi a memoria condivisa.

Il comando:
#pragma omp parallel for
viene utilizzato per rendere un for parallelo.

Il comando:
#pragma omp atomic
invece viene utilizzato per rendere un istruzione atomica.

Abbiamo deciso di utilizzare, OMP perché l'effort per utilizzarlo é praticamente nullo, e le prestazioni sono ottime.

Inoltre essendo che l'implementazione dei *Thread* in *C* cambia tra Windows e Linux, si sarebbe reso necessario modificare il codice per renderlo funzionante su entrambe le piattaforme.

Overview