CONTINUOUS HMM SOLVER

Luca Banzato
Università degli Studi del Piemonte Orientale

Sommario

1. Introduzione

Un Hidden Markov Model (in seguito abbreviato come HMM) è un modello statistico di Markov nel quale il sistema modellato è assunto essere un processo Markoviano con stati non osservati (nascosti)[1][2].
I modelli di Markov nascosti sono conosciuti specialmente per le loro applicazioni in apprendimento per rinforzo e riconoscimento di pattern temporali come il parlato, la scrittura a mano e gesti, analisi grammaticale e bioinformatica.

1.1 - Terminologia di un HMM

Un modello di Markov nascosto è definito dai seguenti elementi:

T: lunghezza della sequenza di osservazioni
N: numero di stati del modello
M: numero di simboli presenti nelle osservazioni
Q: $\{q_0, q_1, \ldots, q_{N-1}\}$ stati distinti del processo di Markov
V: $\{0, 1, \ldots, N-1\}$ insieme delle possibili osservazioni
A: matrice delle probabilità di transizione da uno stato all'altro
B: matrice delle probabilità di emissione di un simbolo in un dato stato
Π: distribuzione dello stato iniziale
O: $(O_0, O_1, \ldots, O_{N-1})$ sequenza di osservazioni

Per **osservazione** si intende un'informazione conosciuta e osservata.

Per "**hidden**" si intende il processo di Markov del primo ordine dietro le osservazioni, non direttamente osservabile e perciò "**nascosto**".

Un modello completo con la matrice delle probabilità di transizione degli stati, la matrice delle probabilità delle osservazioni e la distribuzione dello stato iniziale, è indicato come:

$$\lambda = (A, B, \Pi)$$

Un esempio dell'applicazione dei concetti presentati è dato in seguito:

"Si supponga di volere modellare un sistema di attività di una persona legate alle condizioni metereologiche. Si è a conoscenza del fatto che le attività compiute sono limitate a "Walk", "Shop" e "Clean", e che le condizioni metereologiche possono essere state "Rainy" e "Sunny"."

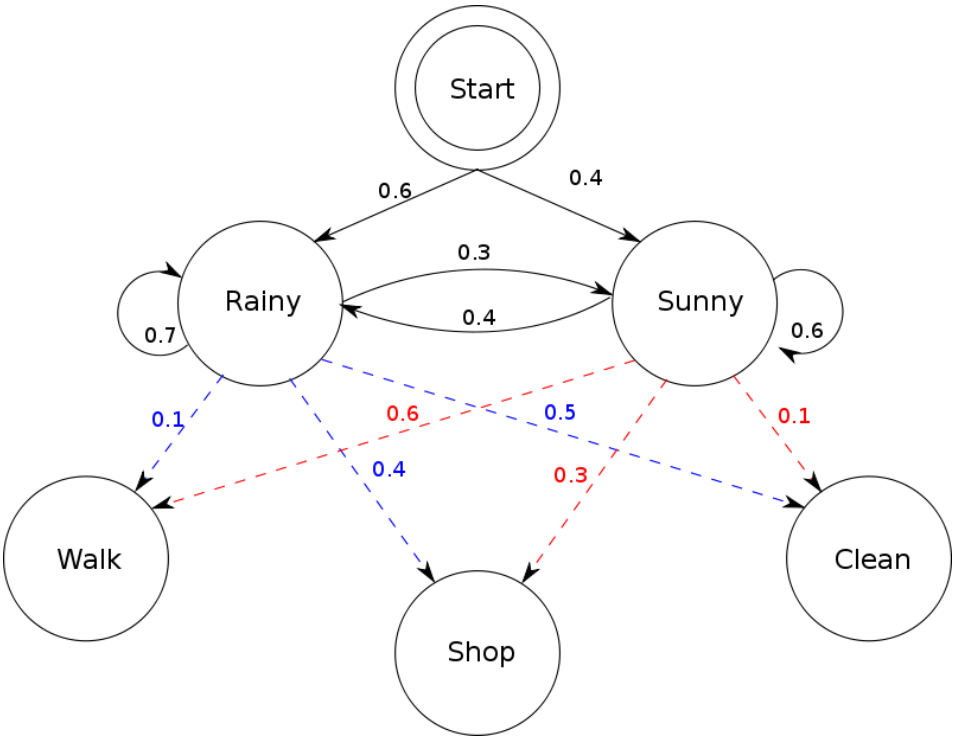

Figura 1 - Esempio di HMM

Nell'esempio specifico, si ha un Hidden Markov Model definito dai seguenti parametri:

N = 2
M = 3
Q = {"Rainy", "Sunny"}
V = {"Walk", "Shop", "Clean"}
Π: = {"Start: 1.0", "Rainy: 0.0", "Sunny: 0.0"}

Le **probabilità di transizione (A)** degli stati sono le frecce che puntano a ciascun stato nascosto.

Le **probabilità di emissione (B)** di un simbolo in un dato stato sono rappresentate invece dalle frecce blu e rosse che puntano a ciascuna osservazione da ciascun stato nascosto.

I collegamenti fra stati e stati o stati ed osservazioni sono rappresentabili tramite una matrice bidimensionale, la quale illustra le probabilità di spostarsi da uno stato all'altro o di produrre uno specifico simbolo durante il permanere in uno stato.

A: S/S'	Rainy	Sunny
Rainy	0.7	0.3
Sunny	0.6	0.4

B: S/O	Walk	Shop	Clean
Rainy	0.1	0.4	0.5
Sunny	0.6	0.3	0.1

Si noti inoltre che il processo di Markov è mostrato dall'interazione tra "Rainy" e "Sunny" nel diagramma e ciascuno di essi è uno **stato nascosto**.

Le **osservazioni** sono informazioni conosciute e si riferiscono a "Walk", "Shop" e "Clean" nel diagramma 1. In machine learning, un'osservazione è un dato per il training e il numero di stati è un parametro del modello in considerazione.

Un esempio di sequenza di osservazioni prodotta dal modello può essere la seguente:

{"Walk", "Walk", "Walk", "Shop", "Walk", "Clean", "Clean"}

In questo caso, la lunghezza della sequenza **T** è uguale a 7

1.2 – I tre problemi fondamentali di un HMM

Gli Hidden Markov Model possono essere utilizzati in applicazioni reali in seguito alla risoluzione dei tre seguenti problemi [3]:

1. Data la sequenza di osservazioni $\boldsymbol{O} = \boldsymbol{O_1}\ \boldsymbol{O_2}\ \ldots\ \boldsymbol{O_T}$, e un modello $\boldsymbol{\lambda = (A, B, \Pi)}$, calcolare efficientemente $\boldsymbol{P(O \mid \lambda)}$, la probabilità della sequenza di osservazioni, dato il modello.
   Si tratta di un problema di valutazione che può essere anche visto come di punteggio, ovvero quanto bene a un dato modello corrisponde una data sequenza di osservazioni.
2. Data la sequenza di osservazioni $\boldsymbol{O} = \boldsymbol{O_1}\ \boldsymbol{O_2}\ \ldots\ \boldsymbol{O_T}$, e un modello $\boldsymbol{\lambda = (A, B, \Pi)}$, determinare una sequenza di stati corrispondenti $\boldsymbol{q} = \boldsymbol{q_1}\ \boldsymbol{q_2}\ \ldots\ \boldsymbol{q_T}$ ottimale, ovvero in grado di spiegare al meglio le osservazioni.
   È bene tenere presente che, salvo il caso di un modello degenere (composto da un solo stato), non esiste una sequenza "corretta", ma una serie di criteri di ottimalità che possono portare alla scoperta di sequenze di stati più probabili.
3. Modificare i parametri del modello $\boldsymbol{\lambda = (A, B, \Pi)}$ per massimizzare $\boldsymbol{P(O \mid \lambda)}$. Si tratta di un problema di ottimizzazione, dove si utilizza una sequenza di osservazioni (denominata **sequenza di "training"**) per adattare i parametri del modello ai dati di training che meglio rappresentano un fenomeno reale.

1.2.1 – Soluzione al primo problema: Procedura Forward-Backward

Il metodo più diretto per calcolare la probabilità della sequenza di osservazioni dato il modello è attraverso l'enumerazione di ogni possibile sequenza di stati di lunghezza T (corrispondente al numero di osservazioni presenti nella sequenza), equivalente alla sommatoria della probabilità congiunta su tutte le possibili sequenze di stati nella forma $\boldsymbol{Q} = \boldsymbol{q_1}\ \boldsymbol{q_2}\ \ldots\ \boldsymbol{q_T}$.

$$P(O|\lambda) = \sum_{tutte\ le\ Q} P(O|Q,\lambda)\ P(Q|\lambda)$$

Il costo richiesto per il calcolo di $P(O|\lambda)$ è di $2T * N^T$, in quanto ad ogni istante temporale t = 1, 2, …, T, ci sono N possibili stati che possono essere raggiunti e per ogni sequenza di stati si ha un costo di 2T per ogni termine della sommatoria. Questo calcolo risulta essere computazionalmente irrealizzabile: basti pensare che per un numero di stati N = 3 e una

sequenza composta da T = 70 osservazioni, il costo computazionale è dell'ordine di $2 * 70 * 3^{70} \approx 10^{35}$.

Al fine di ovviare al problema, si utilizza la procedura di **Forward-Backward**, che sfrutta la tecnica della Programmazione Dinamica per ridurre notevolmente il costo computazionale.

Per la parte **Forward** della procedura, si definisce la variabile forward $\alpha_t(i)$ come

$$\alpha_t(i) = P(O_1\ O_2\ \dots\ O_t, q_t\ =\ S_i|\lambda)$$

ovvero la probabilità della sequenza di osservazioni parziale $O_1\ O_2\ \dots\ O_t$ (fino al tempo t) e la presenza nello stato S_i al tempo t, dato il modello λ. È possibile calcolare il valore di questa variabile mediante un procedimento induttivo:

1. Inizializzazione delle probabilità **forward** come la probabilità congiunta dello stato S_i e l'osservazione iniziale O_1

$$\alpha_1(i) = \pi_i b_i(O_1) \qquad con\ 1\ \leq\ i\ \leq\ N$$

2. Passo induttivo

$$\alpha_t(j) = \left[\sum_{i=1}^{N} \alpha_{t-1}(j) a_{ij}\right] b_j(O_t) \qquad con\ 1\ \leq\ j\ \leq\ N\ e\ 1\ \leq\ t\ \leq\ T$$

Illustrato nella seguente figura è una struttura a lattice con evidenziato il metodo con cui è possibile raggiungere lo stato S_j al tempo t+1 da tutti gli N possibili stati S_i, con $1\ \leq\ i\ \leq\ N$, al tempo t.

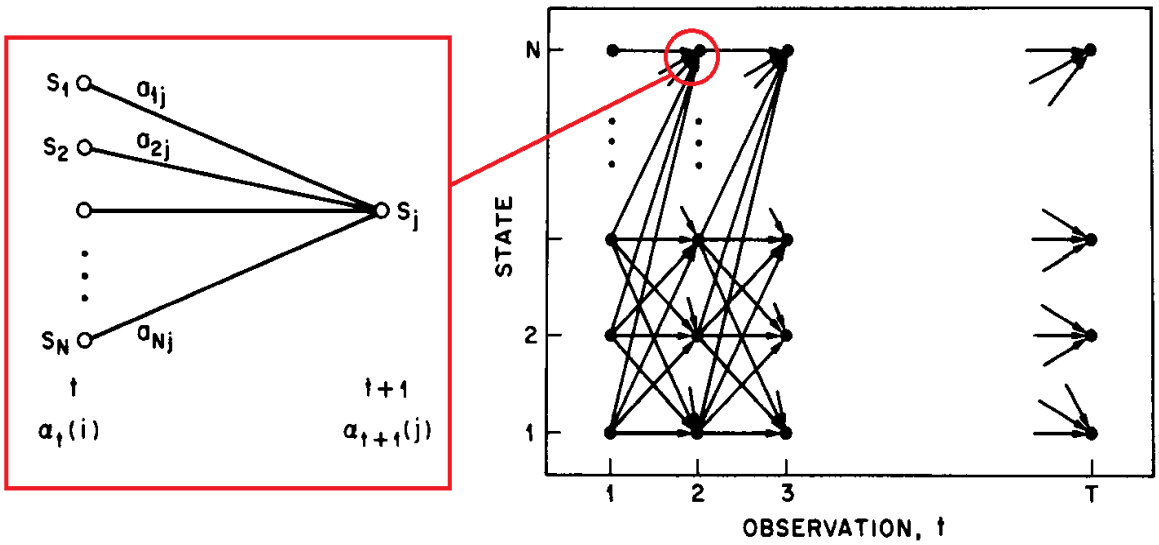

Figura 2 - Illustrazione della sequenza di operazioni per il calcolo della variabile forward $\alpha_t(j)$

3. Terminazione

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

Questo calcolo risulta essere computazionalmente più gestibile rispetto alla precedente procedura, in quanto il costo richiesto per il calcolo di P(O|λ) è di $T * N^2$: per lo stesso numero di stati N = 3 e una sequenza composta da T = 70 osservazioni, il costo computazionale è dell'ordine di $70 * 3^2 \approx 10^2$.

Per la parte **Backward** della procedura, si definisce la variabile backward $\beta_t(i)$ come

$$\beta_t(i) = P(O_{t+1}\, O_{t+2} \dots O_T \mid q_t = S_i, \lambda)$$

ovvero la probabilità della sequenza di osservazioni parziale dall'istante t+1 a T, dato lo stato S_i al tempo t e il modello λ.

Anche in questo caso è possibile calcolare il valore di questa variabile mediante un procedimento induttivo:

1. Inizializzazione delle probabilità **backward**, definendo ciascun $\beta_T(i)$ come

$$\beta_T(i) = 1 \qquad con\ 1 \le i \le N$$

2. Passo induttivo

$$\beta_t(j) = \sum_{j=1}^{N} a_{ij}\, b_j(O_{t+1})\beta_{t+1}(j) \qquad con\ 1 \le i \le N\ e\ t = T-1, T-2, \dots, 1$$

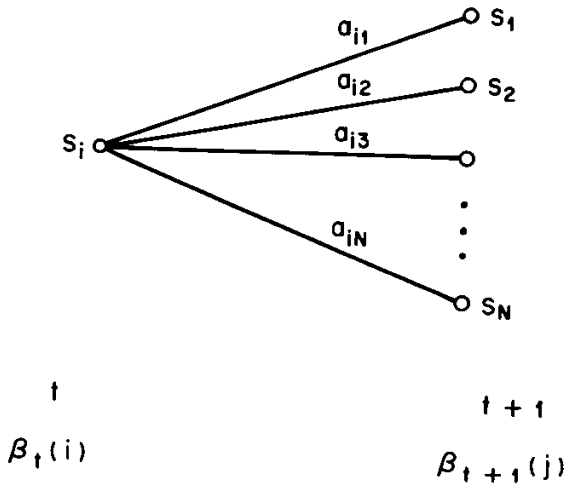

Figura 3 - Sequenza di operazioni richieste per il calcolo della variabile backward $\beta_t(i)$

Nuovamente, il costo richiesto per il calcolo di $\beta_T(i)$ è di $T * N^2$ e può essere calcolato utilizzando una struttura a lattice come quella illustrata in **Figura 3**.

Per la risoluzione del problema è necessario utilizzare solamente la procedura **forward**, ma è stata presentata anche la **backward** per completezza e per il suo utilizzo nella risoluzione del terzo problema, presentato in seguito.

1.2.2 – Soluzione al secondo problema: la variabile Gamma e Viterbi

Per la soluzione al secondo problema, ovvero la ricerca della sequenza di stati ottimale data una sequenza di osservazioni, si adotta un criterio di ottimalità che massimizza il numero atteso di stati individuali corretti. Per implementare questa soluzione, si definisce la variabile

$$\gamma_t(i) = P(q_t = S_i \mid O, \lambda)$$

ovvero la probabilità di essere in uno stato S_i al tempo t, data la sequenza di osservazioni O e il modello λ. Questa equazione può essere espressa in termini delle variabili **forward-backward** come segue:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)}$$

dove $\alpha_t(i)$ prende in considerazione la sequenza di osservazioni parziale $O_1\ O_2 \dots O_T$ e stato S_i al tempo t e $\beta_t(i)$ considera la rimanenza della sequenza di osservazioni parziale $O_{t+1}\ O_{t+2} \dots O_T$ dato lo stato S_i al tempo t. Si noti che il numeratore è $P(q_t = S_i \mid O, \lambda)$, a cui viene applicata una normalizzazione per restituire un valore probabilistico compreso fra 0 e 1.

Tuttavia, nonostante la formula massimizzi il numero atteso di stati corretti, ci possono essere alcuni problemi con la sequenza di stati risultante, come ad esempio una sequenza di stati ottimale in cui esiste una transizione con probabilità 0. Questo è dovuto al fatto che la soluzione adottata determina la sequenza di stati più probabile ad ogni istante, senza considerare la probabilità di occorrenza delle sequenze di stati

Al fine di ovviare a questo problema, è necessario modificare il criterio di ottimalità presentato in precedenza: si vuole ottenere la sequenza di stati singola (percorso), ovvero massimizzare $P(Q \mid O, \lambda)$, equivalente a $P(Q, O \mid \lambda)$. L'**Algoritmo di Viterbi** permette di risolvere questo problema utilizzando la tecnica della Programmazione Dinamica.

Occorre definire due quantità per calcolare la sequenza mediante Viterbi:

$$\delta_t(i) = max_{q_1\, q_2 \dots q_{t-1}} P(q_1\, q_2 \dots q_t = i, O_1\, O_2 \dots O_t \mid \lambda)$$

Ovvero la probabilità lungo un singolo percorso, al tempo t, che tiene conto delle prime t osservazioni, e termina nello stato S_i.

In aggiunta, si definisce il vettore $\psi_t(i)$, che tiene traccia dell'argomento che ha massimizzato $\delta_t(i)$, per ogni t e j.

L'algoritmo di Viterbi può essere calcolato mediante un procedimento induttivo con una struttura a lattice:

1. Inizializzazione

$$\delta_1(i) = \alpha_1(i) = \pi_i b_i(O_1) \qquad con\ 1 \leq i \leq N$$

$$\psi_1(i) = 0$$

2. Ricorsione

$$\delta_t(j) = max_{1 \leq i \leq N} \left[\delta_{t-1}(i) a_{ij}\right] b_j(O_{t+1}) \qquad con\ 1 \leq j \leq N\ e\ 2 \leq t \leq T$$

$$\psi_1(j) = argmax_{1 \leq i \leq N} \left[\delta_{t-1}(i) a_{ij}\right] \qquad con\ 1 \leq j \leq N\ e\ 2 \leq t \leq T$$

3. Terminazione

$$P* = max_{1 \leq i \leq N} \left[\delta_T(i)\right]$$

$$q_T* = argmax_{1 \leq i \leq N} \left[\delta_T(i)\right]$$

4. Backtracking

$$q_t* = \psi_{t+1}(q_{t+1}*) \quad con\ t = T-1, T-2, \ldots, 1$$

1.2.3 – Soluzione al terzo problema: l'algoritmo di Baum-Welch

Il terzo problema prevede la determinazione di un metodo per modificare i parametri del modello per massimizzare la probabilità della sequenza di osservazioni dato il modello stesso.

Per la risoluzione del problema si utilizza una procedura iterativa, quale l'algoritmo di Baum-Welch, che sceglie un modello $\lambda = (A, B, \Pi)$ tale per cui $P(O \mid \lambda)$ è massimizzata localmente.

Per descrivere la procedura di ristima dei parametri, si definisce la variabile $\xi_t(i,j)$ come

$$\xi_t(i,j) = P\left(q_t = S_i, q_{t+1} = S_j \mid O, \lambda\right)$$

ovvero la probabilità di essere nello stato S_i al tempo t e nello stato S_j al tempo t+1, dato il modello e la sequenza di osservazioni. Anche questa equazione può essere espressa in termini di variabili **forward-backward** descritte in precedenza:

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1})\ \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j(O_{t+1})\ \beta_{t+1}(j)}$$

dove anche in questo caso al numeratore equivalente a $P\left(q_t = S_i, q_{t+1} = S_j \mid O, \lambda\right)$ è applicata la normalizzazione al fine di avere un valore compreso fra 0 e 1.

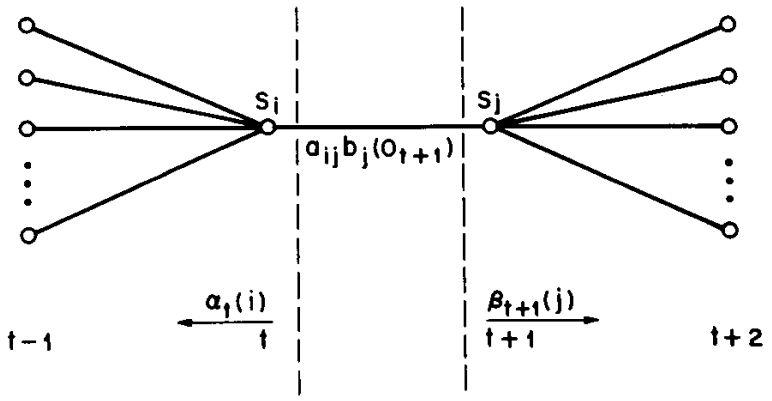

Figura 4 - Sequenza di operazioni richieste per il calcolo di $\xi_t(i,j)$

Utilizzando le formule presentate in precedenza, vengono definite le formule di ristima per i parametri π, A e B:

$$\bar{\pi}_i = \text{numero atteso di volte nello stato } S_i \text{ al tempo } (t=1) = \gamma_1(i)$$

$$\bar{a}_{ij} = \frac{\text{numero ateso di transizioni dallo stato } S_i \text{ allo stato } S_j}{\text{numero atteso di transizioni dallo stato } S_i} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

$$b_j(k) = \frac{\text{numero atteso di volte nello stato } j \text{ e osservando il simbolo } v_k}{\text{numero atteso di volte nello stato } j} = \frac{\sum_{t=1}^{T-1} \gamma_t(j), \text{con } O_t = V_k}{\sum_{t=1}^{T-1} \gamma_t(j)}$$

1.3 – Modifiche per l'utilizzo con osservazioni continue e sequenze multiple

Al fine di utilizzare le formule presentate in precedenza con osservazioni continue, risulta necessario introdurre le seguenti variazioni:

1. Densità di osservazioni continue
2. Sequenze multiple
3. Scaling

1.3.1 – Osservazioni continue

I concetti presentati consentono di utilizzare un HMM con osservazioni caratterizzate da simboli discreti scelti da un alfabeto finito e quindi si utilizza una densità di probabilità discreta per ogni stato del modello.

Il problema di questo approccio, riguardante alcune applicazioni, è che le osservazioni possono essere di natura continua. Nonostante sia possibile eseguire una quantizzazione del segnale, possono verificarsi delle notevoli perdite di informazioni associate alla quantizzazione e risulta quindi essere opportuno utilizzare un HMM con osservazioni continue [2], limitandosi al caso in cui uno stato possa generare osservazioni utilizzando una mistura di gaussiane.

Le modifiche da apportare all'Hidden Markov Model riguardano la sostituzione dell'alfabeto di simboli che possono essere emessi dal modello in uno stato con una mistura di Gaussiane della forma

$$b_j(O) = \sum_{m=1}^{M} c_{jm}\mathrm{M}(O, \mu_{jm}, \sigma_{jm})$$

dove **O** è il vettore di osservazioni modellato, $\boldsymbol{c_{jm}}$ è il coefficiente della mistura per la m-esima mistura nello stato **j** e **M** è una densità log-concava o ellitticamente simmetrica (ad esempio una Gaussiana), con media $\boldsymbol{\mu_{jm}}$ e deviazione standard $\boldsymbol{\sigma_{jm}}$. Graficamente, la nuova matrice delle probabilità di emissione ospiterà **k** distribuzioni Gaussiane Normali e relativo coefficiente per ogni stato, che costituiscono la mistura di Gaussiane.

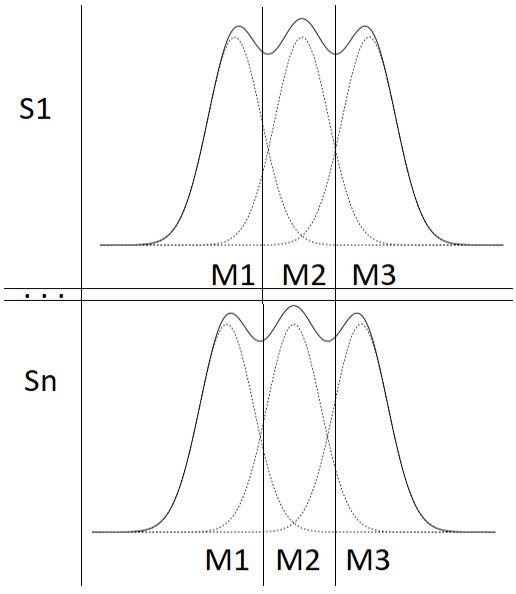

Figura 5 - Matrice B modificata per osservazioni continue

In termini di formule, è necessario apportare modifiche all'equazione della variabile $\gamma_t(j)$, in quanto si dovrà tenere conto del fatto che sono presenti k differenti distribuzioni della mistura che possono produrre un'osservazione con una certa probabilità e peso. Inoltre, dovranno essere fornite le formule per la ristima dei parametri $\boldsymbol{c_{jm}}$, $\boldsymbol{\mu_{jm}}$ e $\boldsymbol{\sigma_{jm}}$ di ogni elemento della mistura.

Per la prima modifica, si definisce la variabile $\gamma_t(j,k)$ come

$$\gamma_t(j,k) = \gamma_t(j)\left[\frac{c_{jm}\mathrm{M}(O_t,\mu_{jk},\sigma_{jk})}{\sum_{m=1}^{M} c_{jm}\mathrm{M}(O,\mu_{jm},\sigma_{jm})}\right]$$

ovvero la probabilità di essere nello stato j al tempo t con la k-esima componente della mistura associata a O_t. Nel caso in cui sia presente una sola distribuzione associata allo stato, il moltiplicatore assume valore 1 e $\gamma_t(j,k) = \gamma_t(j)$.

I parametri delle distribuzioni vengono invece modificati secondo le seguenti formule:

$$\bar{c}_{jk} = \frac{\sum_{t=1}^{T-1}\gamma_t(j,k)}{\sum_{t=1}^{T}\sum_{k=1}^{M}\gamma_t(j,k)}$$

$$\bar{\mu}_{jk} = \frac{\sum_{t=1}^{T}\gamma_t(j,k)O_t}{\sum_{t=1}^{T}\gamma_t(j,k)}$$

$$\bar{\sigma}_{jk} = \sqrt{\frac{\sum_{t=1}^{T}\gamma_t(j,k)(O_t - \mu_{jk})^2}{\sum_{t=1}^{T}\gamma_t(j,k)}}$$

La formula di ristima di $\bar{c}_{jk}$ è il rapporto fra il numero atteso di volte in cui il sistema è nello stato j utilizzando la k-esima componente della mistura e il numero atteso di volte in cui il sistema è nello stato j.

In modo simile, la forma di ristima di $\bar{\mu}_{jk}$ pesa ciascun termine del numeratore con l'osservazione, restituendo il valore atteso della porzione della sequenza di osservazioni utilizzando la k-esima componente della mistura.

Un'interpretazione simile può essere data anche per la formula della ristima della deviazione standard $\bar{\sigma}_{jk}$.

1.3.2 – Sequenze di osservazioni multiple

Al fine di avere dati sufficienti per ottenere stime affidabili di tutti i parametri del modello, è necessario utilizzare sequenze di osservazioni multiple. Supponendo di avere un insieme con un numero S di sequenze di osservazioni continue

$$O = [\, O^1, O^2 \ldots O^s \,], con\ O^s = [O_1^s, O_2^s, \ldots, O_{Ts}^s \,]$$

si assume che ciascuna sequenza di osservazioni sia indipendente da ciascuna altra sequenza. Le formule di ristima per $\bar{\pi}_i$, $\bar{a}_{ij}$, $\bar{c}_{jk}$, $\bar{\mu}_{jk}$ e $\bar{\sigma}_{jk}$ sono definite come:

$$\bar{\pi}_i = \gamma_1(i)\ (invariato, essendo\ \pi_1 = 1\ e\ \pi_i = 0, per\ i\ \neq 1)$$

$$\bar{a}_{ij} = \frac{\sum_{s=1}^{S}\sum_{t=1}^{T-1}\xi_{st}(i,j)}{\sum_{s=1}^{S}\sum_{t=1}^{T-1}\gamma_{st}(i)}$$

$$\bar{c}_{jk} = \frac{\sum_{s=1}^{S}\sum_{t=1}^{T-1}\gamma_{st}(j,k)}{\sum_{s=1}^{S}\sum_{t=1}^{T}\sum_{k=1}^{K}\gamma_{st}(j,k)}\frac{1}{S}$$

$$\bar{\mu}_{jk} = \frac{\sum_{s=1}^{S}\sum_{t=1}^{T}\gamma_t(j,k)O_t}{\sum_{s=1}^{S}\sum_{t=1}^{T}\gamma_t(j,k)}\frac{1}{S}$$

$$\bar{\sigma}_{jk} = \sqrt{\frac{\sum_{s=1}^{S}\sum_{t=1}^{T}\gamma_t(j,k)(O_t - \mu_{jk})^2}{\sum_{S=1}^{S}\sum_{t=1}^{T}\gamma_t(j,k)}}\frac{1}{S}$$

1.3.3 – Scaling

Si è osservato sperimentalmente che, all'aumentare del numero di osservazioni di una sequenza (ad esempio per T > 10), ogni termine di $\alpha_t(i)$ tende esponenzialmente a 0. Per un valore di T > 100, il calcolo di $\alpha_T(i)$ risulta produrre valori che eccedono l'intervallo dinamico di un moderno calcolatore, anche utilizzando valori a doppia precisione.

Risulta quindi necessario incorporare una procedura di scaling durante il calcolo dei termini delle variabili forward e backward, che sia indipendente dallo stato i e dipendente dal tempo t. Si definisce il coefficiente di scaling c_t come:

$$c_t = \frac{1}{\sum_{i=1}^{N}\alpha_t(i)}$$

Utilizzando questa definizione, è possibile ottenere i termini scalati di $\alpha_t(i)$, $\hat{\alpha}_t(i)$:

$$\hat{\alpha}_t(i) = c_t\,\alpha_t(i)$$

Per la computazione di $P(O|\lambda)$, corrispondente all'ultimo passo della procedura forward, non è però possibile eseguire la sommatoria di $\hat{\alpha}_T(i)$, in quanto termini scalati. Sfruttando la proprietà per la quale

$$\prod_{t=1}^{T} c_t \sum_{i=1}^{N} \alpha_T(i) = 1$$

Si può calcolare il logaritmo di P (e non P, in quanto al di fuori dell'intervallo dinamico del calcolatore) con la formula

$$\log[P(O|\lambda)] = -\sum_{t=1}^{T} \log c_t$$

Un ragionamento analogo si applica anche per i termini $\beta_t(i)$ della procedura backward, con l'utilizzo degli stessi coefficienti di scaling utilizzati in precedenza per $\alpha_t(i)$, in quanto permette di mantenere la computazione entro certi limiti ragionabili.

$$\hat{\beta}_t(i) = c_t\, \beta_t(i)$$

Lo scaling risulta essere necessario anche per il calcolo delle sequenze di stati ottimali mediante Viterbi. Fortunatamente, un utilizzo attento dei logaritmi evita la necessità di calcolare un coefficiente di scaling, sostituendo $\delta_t(i)$ con la variabile $\phi_t(i)$ definita come:

$$\phi_t(i) = max_{q_1\, q_2 \ldots\, q_{t-1}}\, log\, P(q_1\, q_2 \ldots\, q_t = i, O_1\, O_2 \ldots\, O_t \mid \lambda)$$

Le formule definite nei I passi necessari per computare l'algoritmo saranno modificate come segue:

1. Inizializzazione

$$\phi_1(i) = \log[\pi_i] + log\, [b_i(O_1)] \qquad con\ 1 \le i \le N$$

$$\psi_1(i) = 0 \qquad con\ 1 \le i \le N$$

2. Ricorsione

$$\phi_t(j) = max_{1 \le i \le N} \left[\phi_{t-1}(i) + \log(a_{ij})\right] + \log[b_j(O_{t+1})] \quad con\ 1 \le j \le N\ e\ 2 \le t \le T$$

$$\psi_1(j) = argmax_{1 \le i \le N} \left[\phi_{t-1}(i) a_{ij}\right] \quad con\ 1 \le j \le N\ e\ 2 \le t \le T$$

3. Terminazione

$$P* = max_{1 \le i \le N} [\phi_T(i)] \quad con\ 1 \le i \le N$$

$$q_T* = argmax_{1 \le i \le N} [\phi_T(i)] \quad con\ 1 \le i \le N$$

4. Il passo di backtracking rimane invece invariato.

Nel caso di sequenze di osservazioni multiple, è necessario apportare alcune modifiche alle formule per il calcolo dei termini di $\xi_t(i,j)$, che, come riportato nel procedimento illustrato nell'errata [4] del turorial di Rabiner, si semplifica in

$$\hat{\xi}_t(i,j) = \hat{\alpha}_t(i)a_{ij}b_j(O_{t+1})\hat{\beta}_{t+1}(j)$$

2 – Implementazione

L'implementazione è stata eseguita mediante il linguaggio di programmazione Java. La scelta è ricaduta su questo linguaggio in quanto meglio conosciuto dal programmatore al momento dell'implementazione e consente la creazione semplificata di applicazioni cross-platform.
Tuttavia si tratta di una scelta discutibile, in quanto il programma afferisce all'insieme degli strumenti utilizzati per il machine learning: questo si traduce in un numero elevato di calcoli da eseguire in un tempo idealmente breve. Utilizzare un framework che prevede l'esecuzione delle applicazioni in una virtual machine (**Java Virtual Machine** o **JVM**) aumenta il numero di risorse richieste al sistema, oltre al tempo notevolmente aumentato.
Un linguaggio di programmazione come C++ o C# sarebbe sicuramente stata una scelta migliore, ma per esigenze pratiche e di tempo si è deciso di procedere con Java.

In fase progettuale si sono stabilite le caratteristiche funzionali minime per considerare come completa la realizzazione del programma, indicate di seguito:

- Algoritmo di Baum-Welch (con un numero fisso di passi o learning e pruning set)
- Algoritmo di Viterbi
- Supporto allo scaling
- Supporto alle sequenze di osservazioni multiple
- Supporto alle osservazioni continue
- Test di una o più sequenze con due o più modelli (classificazione)
- Valutazione del classificatore con Cross-Validation a K-Fold
- Formato standardizzato per HMM e dataset
- Rescaling del dataset
- Interfaccia grafica

Gli obiettivi prefissati sono stati quasi tutti raggiunti, salvo incorrere in alcune limitazioni che verranno specificate in seguito.

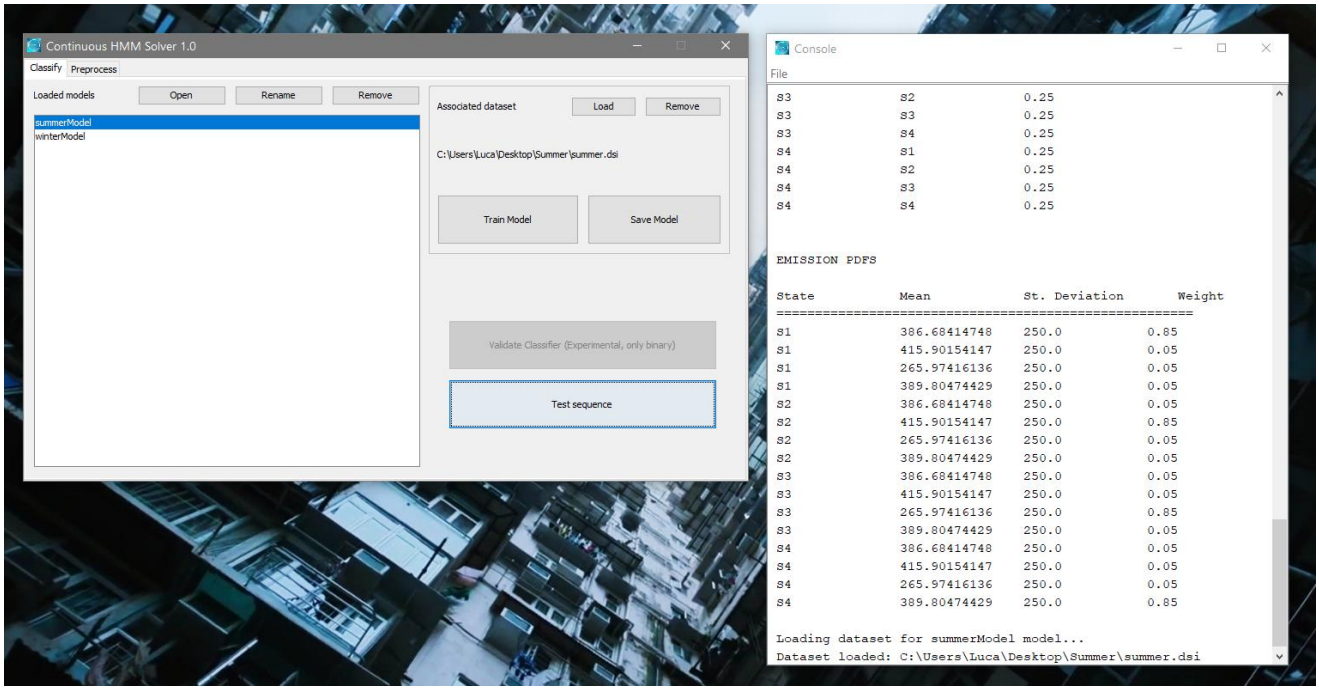

Figura 6 - Il risultato del progetto

2.1 – Librerie utilizzate

Per la realizzazione di questo progetto, ci si è appoggiati alle seguenti librerie:

- **Parallel Colt**: un insieme di librerie Open Source per computazioni scientifiche tecniche ad alte prestazioni con Java prodotta dal CERN di Ginevra.
  https://sites.google.com/site/piotrwendykier/software/parallelcolt
- **The Apache Commons Mathematics Library**: libreria di componenti matematiche e statistiche non presenti in Java.
  https://commons.apache.org/proper/commons-math/
- **Gson**: libreria di Google per serializzazione/deserializzazione di dati in formato JSON.
  https://github.com/google/gson

Una nota viene fatta per un ulteriore libreria: durante l'implementazione, ci si è inoltre accorti che sarebbe stato utile potere utilizzare un corrispettivo delle **struct** presente in linguaggi di programmazione come C e C++. Il progetto **JUnion** (https://github.com/TehLeo/junion) sembrava potere risolvere questo problema, ma il suo utilizzo con l'IDE Eclipse in ambiente Microsoft® Windows ha creato non poche difficoltà, legate ad alcuni bug della libreria e l'assenza del supporto di array di struct, richiesta imprescindibile del progetto.

Sono state spese alcune ore per provare ad integrare JUnion con Continuous HMM Solver, ma si è deciso infine di abbandonare l'idea in favore di soluzioni più semplici. È comunque stata l'occasione per individuare e segnalare all'autore la presenza di alcuni bug mediante il sistema di tracking di GitHub, contribuendo in qualche modo ad un altro progetto esistente.

2.2 – Nucleo del progetto: TrainableHMM

La classe fondamentale per il progetto è TrainableHMM, che vede implementate quasi tutte le funzionalità descritte in precedenza.

Per la matrice di transizione dell'HMM si è utilizzata una struttura sparsa, istanza della classe DoubleMatrix2D presente all'interno della libreria **Parallel Colt**, in quanto possono essere presenti probabilità di transizione nulle e risulta comodo avere a disposizione un modello a lattice che supporta anche la rimozione di valori nulli allocati utilizzando il metodo **trimToSize**.

Per la matrice di emissione si è scelto di adottare una matrice di distribuzioni normali pesate, implementate con la classe WeightedNormal, in quanto si è scelto di considerare come valore nullo una distribuzione Normale pesata con peso 0. La classe WeightedNormal è estensione di NormalDistribution, presente in **The Apache Commons Mathematics Library**. Il vettore contenente le probabilità di essere in uno stato al tempo t=0 è implementato mediante un vettore sparso SparseDoubleMatrix1D. Le motivazioni di questa scelta sono le medesime della matrice di transizione.

Per la classe sono forniti due costruttori:

```
TrainableHMM(String name, SparseDoubleMatrix2D A, WeightedNormal[][] B,
SparseDoubleMatrix1D PI, Enumerator enumeratedStates))

TrainableHMM(File hmmFile) throws IOException
```

Il primo prevede il passaggio di tutti i parametri fondamentali di un HMM, ovvero le matrici A e B, il vettore P, il nome assegnato al modello e un oggetto Enumerator, contenente l'enumerazione degli stati dell'Hidden Markov Model: quest'ultimo consente di associare un valore intero ad uno stato e di utilizzarlo in sostituzione di una stringa negli algoritmi implementati, portando un notevole vantaggio in termini di prestazioni, in quanto il confronto fra stringhe ha un costo computazionale maggiore rispetto a quello fra valori interi.
Il secondo costruttore prevede la creazione di un'istanza di TrainableHMM partendo da una definizione salvata in un file in formato JSON; la specifica del formato verrà presentata in seguito.

In aggiunta ai costruttori, la classe offre i seguenti metodi pubblici:

- `double likelihood(Observation[] sequence)`: data una sequenza di osservazioni, ritorna la probabilità (intesa come log-probability) che sia stata generata dall'HMM

- `Struct_ViterbiResults viterbi(Observation[] sequence)`: data una sequenza di osservazioni, ritorna una struttura dati contenente la sequenza di stati più probabile che possa avere generato le osservazioni e la probabilità che questa sequenza sia effettivamente quella meglio rappresentativa.

- `void viterbiToFile(Observation[] sequence, File outFile) throws IOException`: salva su file i risultati dell'algoritmo di Viterbi. Utilizza il metodo **viterbi** presentato sopra.

- `String getModelName()`: ritorna una stringa contenente il nome assegnato all'HMM. Se l'HMM è stato caricato da file, il nome sarà equivalente al nome del file, privato però della sua estensione, se presente.

- `void setModelName(String name)`: modifica il nome del modello con quello passato in input.

- `String toString()`: ritorna una stringa contenente la rappresentazione di un HMM (con A, B e PI)

- `void bw_fixed(int steps, Observation[][] learningSet, File... directory) throws IOException`: esegue un numero fissato di passi dell'algoritmo di Baum-Welch. Consultare il paragrafo **2.2.5** per dettagli.

- `void bw_adaptive(Observation[][] learningSet, Observation[][] pruningSet, File... directory) throws IOException`: esegue l'algoritmo di Baum-Welch con l'utilizzo del pruning set. Consultare il paragrafo **2.2.5** per dettagli.

- `void writeToFile(File hmmFile) throws IOException`: salva su file tutte le informazioni riguardanti l'HMM (A, B, PI) in formato JSON.

2.2.1 – Il problema dei risultati intermedi

Per l'implementazione di Baum-Welch e Viterbi, è inizialmente risultata problematica la creazione di una o più strutture dati che permettessero il salvataggio dei risultati intermedi delle variabili legate alle computazioni.
Fortunatamente si ha avuto modo di discutere il problema durante le lezioni del corso e, su suggerimento del docente, l'idea è stata quella di considerare la creazione di una struttura dati (qui denominata **TimeSlot**) che contenesse al suo interno i valori delle variabili computazionali utilizzate da Baum-Welch / Viterbi, tutte dipendenti dall'istante temporale.

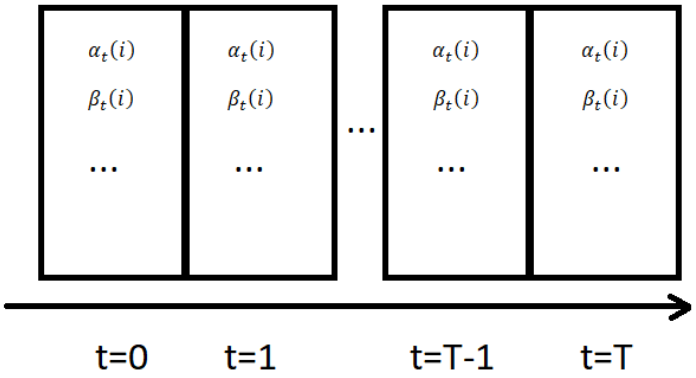

Figura 7 - Rappresentazione grafica dei TimeSlot

All'interno della struttura **TimeSlot** sono contenute le seguenti variabili:

- Alpha: un vettore sparso contenente i valori ottenuti dalla procedura forward al tempo t per ogni stato i.
- Beta: vettore sparso contenente i valori ottenuti dalla procedura backward al tempo t per ogni stato i.
- Gamma: vettore sparso contenente le probabilità di essere nello stato i al tempo t.
- Gamma_K: matrice sparsa contenente le probabilità di essere nello stato i al tempo t, con la k-esima componente della mistura associata all'osservazione O_t.
- Delta: vettore contenente le probabilità calcolate mediante Viterbi.
- State: vettore contenente il prossimo indice dello stato più probabile (Viterbi).
- Xi: matrice contenente la probabilità di transizione da uno stato all'altro al tempo t.
- C: fattore di scaling utilizzato nella procedura Forward-Backward.

Il contenuto della struttura deve essere inizializzato richiamando il metodo **init**, illustrato nel dettaglio nel paragrafo 2.2.5.

2.2.2 – Procedura Forward-Backward

Per l'implementazione dell'algoritmo ci si è limitati alla realizzazione di due metodi:

`double alpha(Observation[] observations)`: implementazione parte forward.

`double beta(Observation[] observations)`: implementazione parte backward.

Per la procedura **forward** si sono implementate le formule che prevedono lo scaling dei valori:

1. Inizializzazione

$$\hat{\alpha}_1(i) = \frac{\pi_i b_i(O_1)}{c_1} \qquad con\ 1 \leq i \leq N$$

2. Passo induttivo

$$\hat{\alpha}_t(j) = \frac{[\sum_{i=1}^{N} \alpha_{t-1}(j) a_{ij}] b_j(O_t)}{c_t} \qquad con\ 1 \leq j \leq N\ e\ 1 \leq t \leq T$$

3. Terminazione

$$\log[P(O|\lambda)] = -\sum_{t=1}^{T} \log c_t$$

Inizialmente si è provato a verificare il funzionamento dell'algoritmo senza l'utilizzo delle formule con i valori scalati, ma i risultati sono caduti in underflow.

Un discorso analogo viene fatto per la procedura **backward**:

1. Inizializzazione

$$\hat{\beta}_T(i) = \frac{1}{c_T} \qquad con\ 1 \leq i \leq N$$

2. Passo induttivo

$$\hat{\beta}_t(j) = \sum_{j=1}^{N} a_{ij}\, b_j(O_{t+1})\hat{\beta}_{t+1}(j) \qquad con\ 1 \leq j \leq N\ e\ t = T-1, T-2, \dots, 1$$

Nel caso della procedura **forward** viene ritornato il valore $\log[P(O|\lambda)]$, mentre non è necessario restituire alcun valore per **backward**, in quanto non utile ai fini di Baum-Welch.

2.2.3 – Gamma e Xi

In analogia a quanto visto nel precedente paragrafo, sono implementate tre funzioni per il calcolo del valore assunto dalle variabili Gamma e Xi:

- **`void gamma(int T)`**: implementa il calcolo della funzione

$$\hat{\gamma}_t(i) = \frac{\hat{\alpha}_t(i)\hat{\beta}_t(i)}{\sum_{i=1}^{N} \hat{\alpha}_t(i)\hat{\beta}_t(i)} \qquad con\ 1 \leq i \leq N$$

- **`void gamma_k(Observation[] sequence)`**: implementa il calcolo della funzione

$$\hat{\gamma}_t(j,k) = \hat{\gamma}_t(j)\left[\frac{c_{jm}\mathrm{M}(O_t, \mu_{jk}, \sigma_{jk})}{\sum_{m=1}^{M} c_{jm}\mathrm{M}(O, \mu_{jm}, \sigma_{jm})}\right] \qquad con\ 1 \leq j \leq N\ e\ 1 \leq k \leq M$$

- **`void xi(Observation[] sequence)`**: implementa il calcolo della funzione

$$\hat{\xi}_t(i,j) = \hat{\alpha}_t(i)a_{ij}b_j(O_{t+1})\hat{\beta}_{t+1}(j) \qquad con\ 1 \leq i \leq N\ e\ 1 \leq j \leq N$$

2.2.4 - Viterbi

La funzione fondamentale che si occupa della computazione dell'algoritmo di Viterbi è

`Struct_ViterbiResults delta(Observation[] sequence)`

che vede implementate le seguenti formule con l'uso dei logaritmi, in accordo a quanto descritto nei paragrafi **1.2.2** e **1.3.3**.

1. Inizializzazione

$$\phi_1(i) = \log[\pi_i] + log\ [b_i(O_1)] \qquad con\ 1 \leq i \leq N$$

$$\psi_1(i) = 0 \qquad con\ 1 \leq i \leq N$$

2. Ricorsione

$$\phi_t(j) = max_{1 \leq i \leq N} \left[\phi_{t-1}(i) + \log(a_{ij})\right] + \log[b_j(O_{t+1})] \quad con\ 1 \leq j \leq N\ e\ 2 \leq t \leq T$$

$$\psi_1(j) = argmax_{1 \leq i \leq N} \left[\phi_{t-1}(i) a_{ij}\right] \quad con\ 1 \leq j \leq N\ e\ 2 \leq t \leq T$$

3. Terminazione

$$P* = max_{1 \leq i \leq N} [\phi_T(i)] \qquad con\ 1 \leq i \leq N$$

$$q_T* = argmax_{1 \leq i \leq N} [\phi_T(i)] \qquad con\ 1 \leq i \leq N$$

4. Backtracking

$$q_t* = \psi_{t+1}(q_{t+1}*) \quad con\ t = T-1, T-2, \dots, 1$$

Per l'implementazione della funzione **max** (utilizzata nel passo 2) è necessario inizialmente specificare una log-probabilità (**maxWeight**) pari a -∞ prima di procedere con i confronti, in quanto i valori ottenuti saranno negativi per via dell'utilizzo dei logaritmi con valori compresi fra 0 e 1. Nel caso del passo 3, questo discorso non è necessario, in quanto i valori di $\phi_T(i)$ sono già stati calcolati in precedenza e si può considerare come valore iniziale di confronto $\phi_T(0)$. Il risultato migliore ottenibile mediante **max** è il valore più prossimo a 0.

Per il backtracking invece è necessario iterare dall'istante T a 1 sul vettore **State** presente in ciascun **TimeSlot**, in quanto contenente le probabilità dei singoli stati ad ogni istante temporale.

2.2.5 – L'algoritmo di Baum-Welch

Per Baum-Welch si è adottato l'utilizzo di tre funzioni, due considerabili come **wrapper** e una terza che risulta implementare il cuore dell'algoritmo.

- `void bw_fixed(int steps, Observation[][] learningSet, File... directory) throws IOException`

  Esegue un numero predefinito di passi determinato dal valore **steps** dell'algoritmo di Baum-Welch, utilizzando il dataset **learningSet**. Eventualmente, è possibile indicare una directory dove salvare i risultati parziali prodotti da ogni passo dell'algoritmo.

- `void bw_adaptive(Observation[][] learningSet, Observation[][] pruningSet, File... directory)throws IOException`

  Esegue un numero variabile di passi utilizzando un **learningSet** in combinazione con un **pruningSet**.

  Questa tecnica prevede di continuare ad eseguire uno step di Baum-Welch utilizzando il learning set; in seguito, si calcola $\log[P(O|\lambda)]$ per ogni sequenza del pruning set e si determina il valore medio dividendo il risultato cumulativo per il numero di sequenze del pruning set.
  Se durante l'iterazione *step*, la media risulta essere **maggiore** (e non minore, in quanto si sta operando con i logaritmi) di quella ottenuta all'iterazione precedente, il modello prodotto dall'ultimo step di Baum-Welch risulta essere peggiore di quello dello step precedente ed è necessario interrompere l'algoritmo e restituire in output il precedente modello.
  Altrimenti si provvede ad aggiornare il valore della media migliore con quello calcolato durante l'ultimo passo.

  Al fine di restituire in tempi accettabili un risultato, si è scelto di limitare il numero di passi eseguibili in questa modalità a 200. Il valore può essere aumentato o rimosso eliminando la condizione all'interno del costrutto **while**.

- `void bw_step(Observation[][] sequences)`

  Il metodo fondamentale richiamato dalle due funzioni **wrapper** presentate in precedenza, che prevede la computazione di un singolo passo di Baum-Welch.

  Per ogni sequenza del dataset

  1. Inizializzazione delle strutture dati mediante il metodo **init**, che si occupa di istanziare le variabili utilizzate dai **TimeSlot**. In aggiunta, per motivi di efficienza, si provvede a calcolare la probabilità totale di emissione di un valore da parte di uno stato, data dalla somma delle probabilità di ogni singola componente del **mixture model** associata allo stato:

$$b_j(O_t) = \sum_{m=1}^{M} c_{jm} \mathrm{M}(O_t, \mu_{jm}, \sigma_{jm})$$

2. Calcolo di **alpha**, **beta**, **gamma** e **xi** e salvataggio del **TimeSlot** all'interno di un vettore (Vedere figura 7).

Si procede infine con la fase di ristima dei parametri secondo le formule presentate al termine del paragrafo **1.3.2**.

2.2.6 – Lettura e scrittura di un HMM

Per gestire la lettura/scrittura da/su disco per i modelli si è deciso di utilizzare lo standard JSON. Un modello può essere definito su file utilizzando la seguente sintassi:

```
{
    "PI": [
      {"state":"Nome_Stato_1", "probability":0.5},
      … { … },
      {"state":"Nome_Stato_N", "probability":0.1}
    ],
    "A": [
      {"state":"Nome_Stato_N", "to":" Nome_Stato_2", "probability":0.2},
      … { … },
      {"state":"Nome_Stato_N", "to":" Nome_Stato_1", "probability":0.8}
    ],
    "B": [
      {"state":"Nome_Stato_1", "distributions":[{"mu":327.795243, "sigma":250, "weight":0.8},
      …, {"mu":316.679794, "sigma":250, "weight":0.04]},
      … { … },
      {"state":"Nome_Stato_N", "distributions":[{"mu":327.795243, "sigma":250, "weight":0.8},
      …, {"mu":319.232111, "sigma":250, "weight":0.04]}
    ]
}
```

Un file di esempio pronto alla modifica è fornito insieme a **Continuous HMM Solver**.

I metodi che si occupano della lettura e scrittura sono

```
void readFromFile(File hmmFile) throws JsonParseException, IOException

void writeToFile(File hmmFile) throws IOException
```

Il primo è utilizzato dal costruttore della classe quando si invoca con un parametro di tipo **File** e non è utilizzabile direttamente da nessuna istanza della classe. All'interno di questo metodo si esegue la deserializzazione del file e vengono eseguiti alcuni controlli di consistenza per accertarsi che il modello sia stato correttamente definito, come ad esempio la somma a 1.0 delle probabilità iniziali degli stati.

Il secondo metodo, presentato già in precedenza, prevede la serializzazione e scrittura su file del modello. Al suo interno sono presenti alcuni controlli per assicurarsi che l'HMM non presenti valori **NaN** (Not a Number), dovuti quasi sempre a risultati di Baum-Welch con modelli aventi parametri non idonei rispetto al dataset.

Nel caso in cui si verifichi questa situazione, è necessario convertire il valore **NaN** in una stringa, in quanto la specifica JSON non prevede valori **NaN**. Il modello salvato risultante **non sarà apribile da Continuous HMM Solver**, ma servirà all'utente per comprendere che si è verificato un problema, ad esempio, durante la computazione di un Baum-Welch adattivo.

2.3 – Gli altri componenti del sistema

In aggiunta alla classe TrainableHMM, sono presenti una serie di componenti di supporto, utili alla manipolazione dei dataset o per la gestione dei risultati prodotti dalla classe principale. Di questi verrà offerta una panoramica più generale, in quanto ritenuti secondari.

2.3.1 – DatasetUtils

La classe include alcuni metodi per la gestione dei dataset, fra cui il rimescolamento e separazione in due parti separate (in rapporto 80/20) di un dataset in Learning e Pruning set. Alla lettura dei dataset da file sono dedicati due metodi: il primo prevede la consultazione di un file contenente l'indice dei files che compongono il dataset, nel formato

File1
File2
…
FileN

mentre il secondo prevede la lettura dei singoli file delle sequenze di osservazione. L'indice può essere compilato a mano o in maniera automatizzata, utilizzando ad esempio il comando **ls > index.dsi** in ambiente GNU/Linux o UNIX e rimuovendo il nome del file dell'indice dall'elenco creato.

Per la manipolazione del dataset è presente la funzione **meanNRescale**, un'implementazione in Java del rescaler scritto in C e fornito insieme al dataset come materiale base per il progetto.

2.3.2 – Classificatore e Interfaccia Grafica

Per rendere più semplice e fruibile l'utilizzazione di un classificatore che sfrutta gli Hidden Markov Model, si è deciso per l'implementazione di questo sfruttando le librerie grafiche Swing.

Il package **classifier** include al suo interno una serie di subpackage per meglio suddividerne i componenti:

- **core:** contiene tutte le classi che compongono la base per l'esecuzione dell'applicazione e della sua interfaccia grafica.
- **actions**: comprende tutte le classi che gestiscono le azioni prodotte dall'iterazione dell'utente con i componenti dell'interfaccia grafica.
- **workers**: comprende tutte le classi che si occupano di eseguire **un'azione richiesta dall'utente** in background, **prima fra tutte la classificazione di sequenze**, per poi passare, ad esempio, all'allenamento di un HMM o la lettura di un dataset da file.
- **validation**: include le classi che compongono il modulo di validazione di un HMM. Al momento è implementata la tecnica di Validazione Incrociata per un **massimo di due modelli**. La procedura permette di mantenere in considerazione alcuni indici di prestazione, come **Accuracy, Error Rate, Recall e Precision**.

3 – Utilizzo di Continuous HMM Solver

L'interfaccia del programma si compone di due finestre: **programma principale e console**. Il programma principale integra tutte le funzionalità per potere eseguire il caricamento di modelli e dataset, la console invece presenta un'area testuale dove saranno presentati all'utente tutte le informazioni riguardanti i risultati di una sessione di training di un HMM o messaggi di sistema.

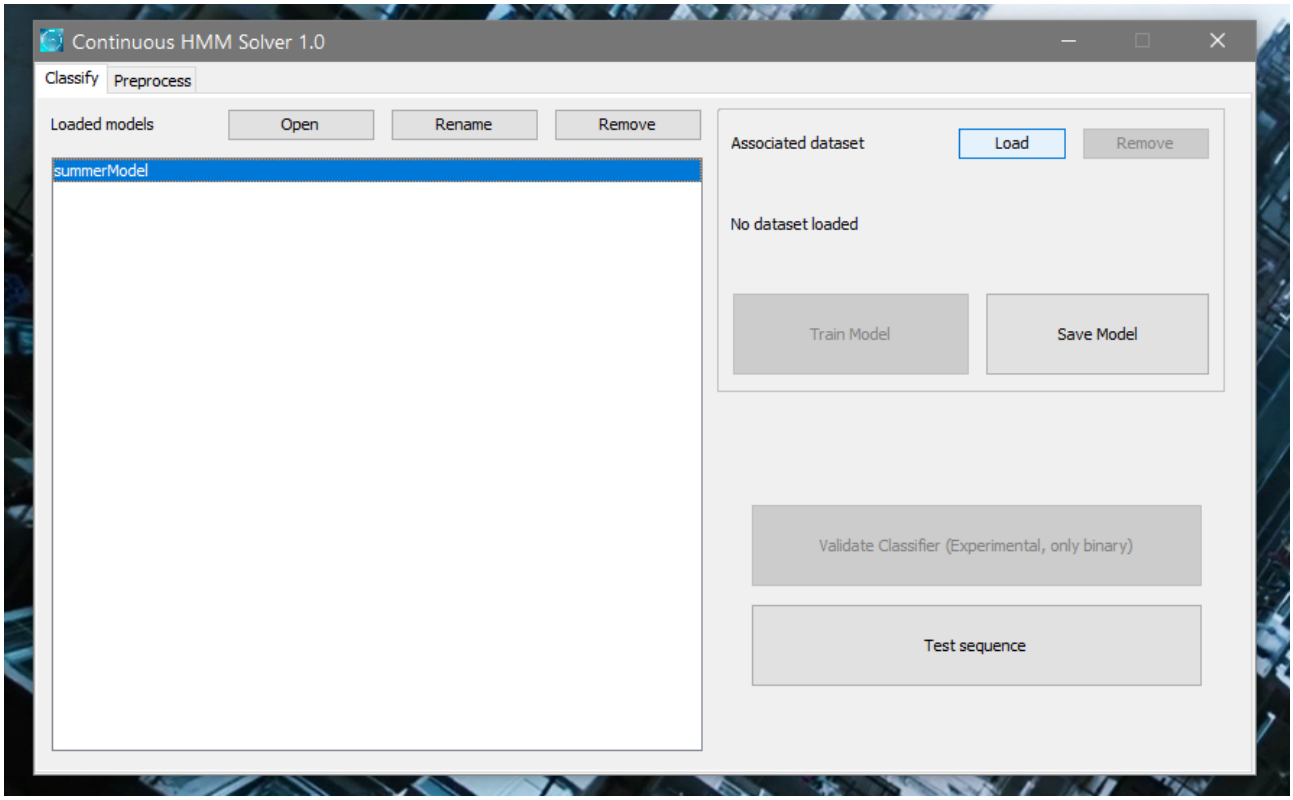

Figura 8 - Interfaccia principale del classificatore

La prima operazione richiesta all'utente è il caricamento di un modello da file mediante l'unico pulsante abilitato, "**Open**". Caricato il modello, all'utente viene concessa la possibilità di salvarlo su un nuovo file, provare direttamente la likelihood di un insieme di sequenze, oppure di associare un dataset specifico al modello mediante prima la selezione del nome del modello scelto dalla lista sulla sinistra, e in seguito la pressione del pulsante "**Load**" sulla destra, che permetterà la selezione di un indice del dataset nel formato indicato nel paragrafo **2.3.1**.

Il caricamento del dataset abiliterà l'opzione per l'allenamento del modello mediante Baum-Welch (pulsante **Train Model**), per la quale sarà possibile impostare alcuni settaggi, come ad esempio il numero di passi dell'algoritmo e se si vogliono salvare su file i risultati.

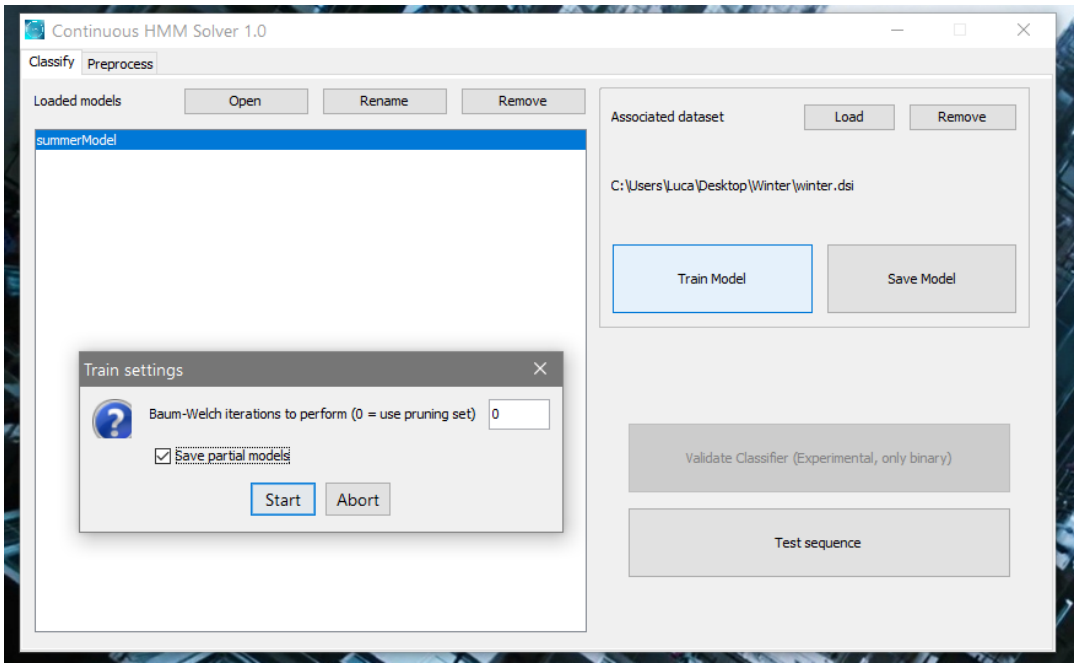

Figura 9 - Impostazioni per l'allenamento del modello

Per abilitare l'opzione di Cross-Validazione del classificatore, occorre caricare esattamente due modelli (per il limite imposto in precedenza, vedere paragrafo **2.3.2**) ed associare un corrispondente dataset. I modelli caricati per la validazione incrociata non devono essere allenati mediante la funzione **Train Model** e devono essere modelli di partenza per l'allenamento con Baum-Welch.

Per il **test delle sequenze** è possibile selezionare uno o più file dalla finestra di caricamento (per la selezione multipla è necessario in ambiente Windows utilizzare il tasto CTRL e click sinistro del mouse, discorso analogo per i sistemi basati su GNU/Linux e UNIX). Terminata la selezione, l'utente può decidere o meno di salvare nella stessa directory delle sequenze i file contenenti tutti i risultati dell'**algoritmo di Viterbi** di entrambi i modelli.

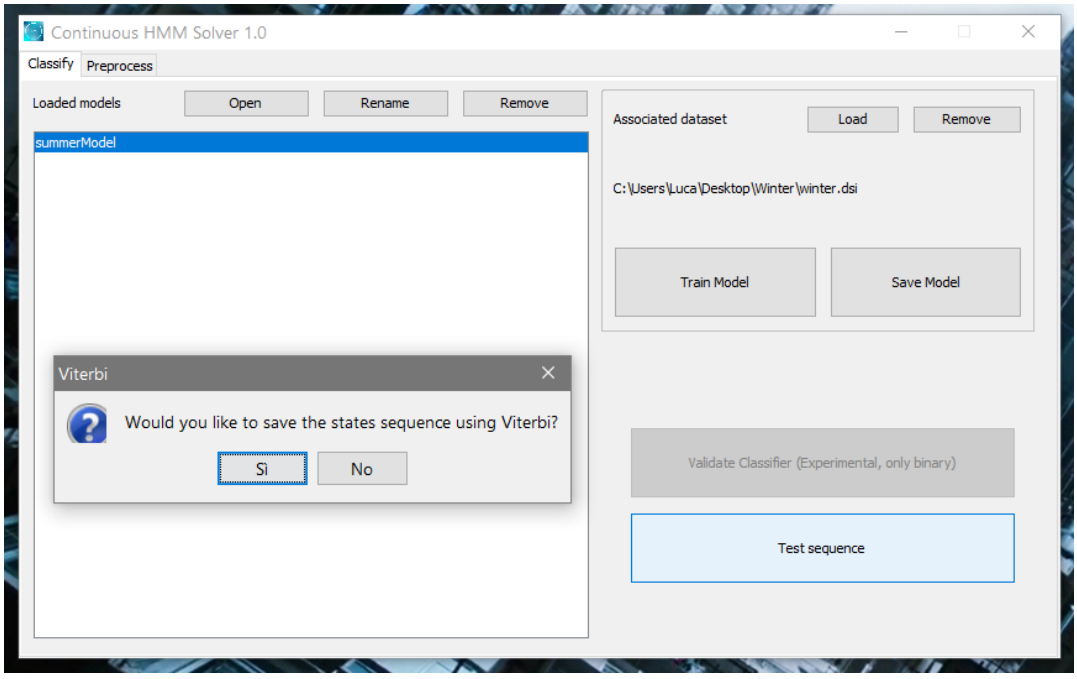

Figura 10 - Finestra per l'uso dell'algoritmo di Viterbi

La scheda **Preprocess** offre invece un'interfaccia grafica per il rescaler di cui si è discusso nel paragrafo **2.3.1**, fornito insieme al dataset di partenza del progetto.

Per il suo utilizzo, occorre selezionare l'indice di un dataset, una cartella di destinazione ed il **fattore di unione** utilizzato dal rescaler per determinare quanti valori devono essere aggregati in uno solo. Al termine della fase di preprocessing sarà creato il dataset riscalato e il relativo file di indice nella directory specificata in precedenza.

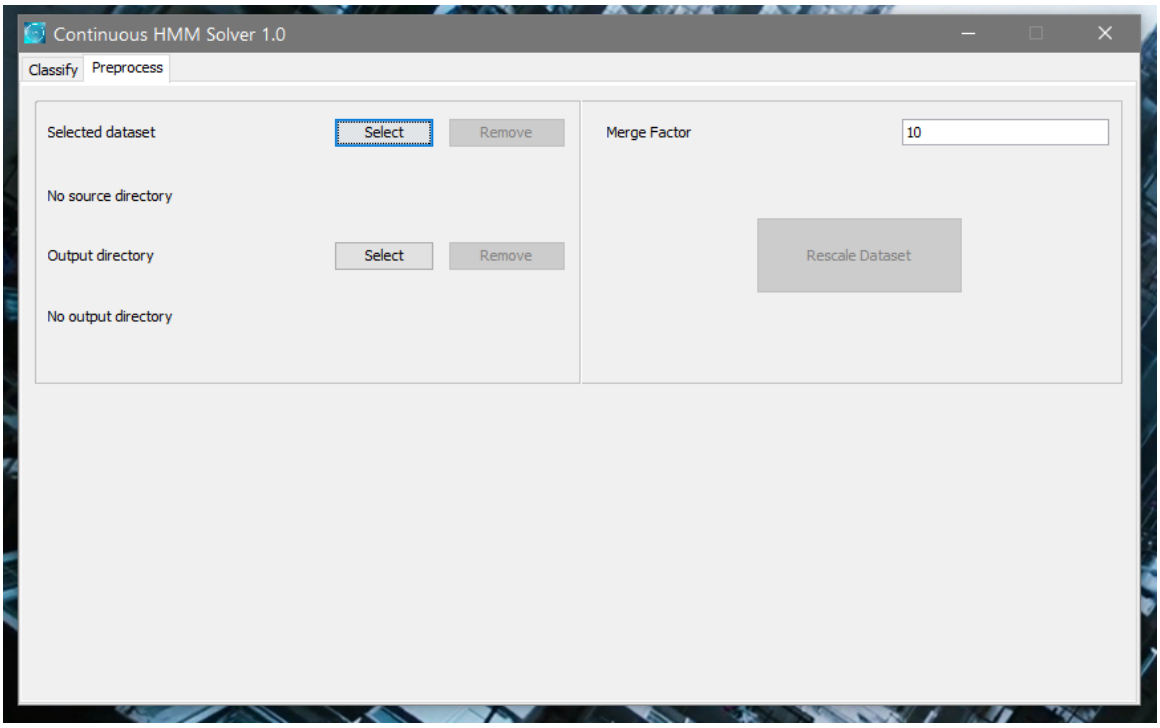

Figura 11 - Finestra del rescaler

La console mette a disposizione dell'utente un menu di funzionalità, attraverso il quale l'utente può procedere alla cancellazione o al salvataggio di tutte le informazioni presenti all'interno di questa (utile in particolare per salvare i risultati di classificazione/validazione dei modelli).

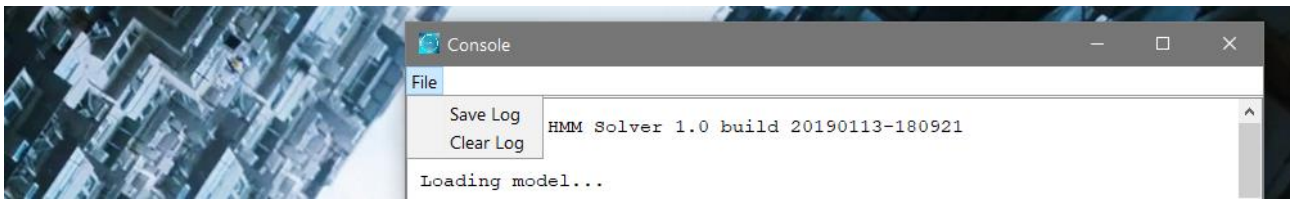

Figura 12 - Menu della console

4 – Un caso pratico: classificazione di sequenze di una pompa di calore

Terminata l'implementazione di **Continuous HMM Solver**, si è deciso di utilizzarlo per trattare un caso reale di utilizzo, ovvero la classificazione di sequenze prodotte da una **pompa di calore** (o **termopompa**) operante in due modalità differenti: riscaldamento e raffreddamento.

Insieme alle richieste del progetto viene infatti fornito un dataset di valori prodotti da una BeagleBone collegata alla termopompa, durante l'arco temporale di un anno.

Il dataset è composto da 341 file di **coppie di valori**, il primo indicante i **minuti relativi trascorsi dalla prima misurazione** della giornata, il secondo il **consumo in Wh** della termopompa. In aggiunta a questi sono presenti i corrispondenti grafici in formato **pdf** ottenuti mediante lo strumento **GNUPlot**.

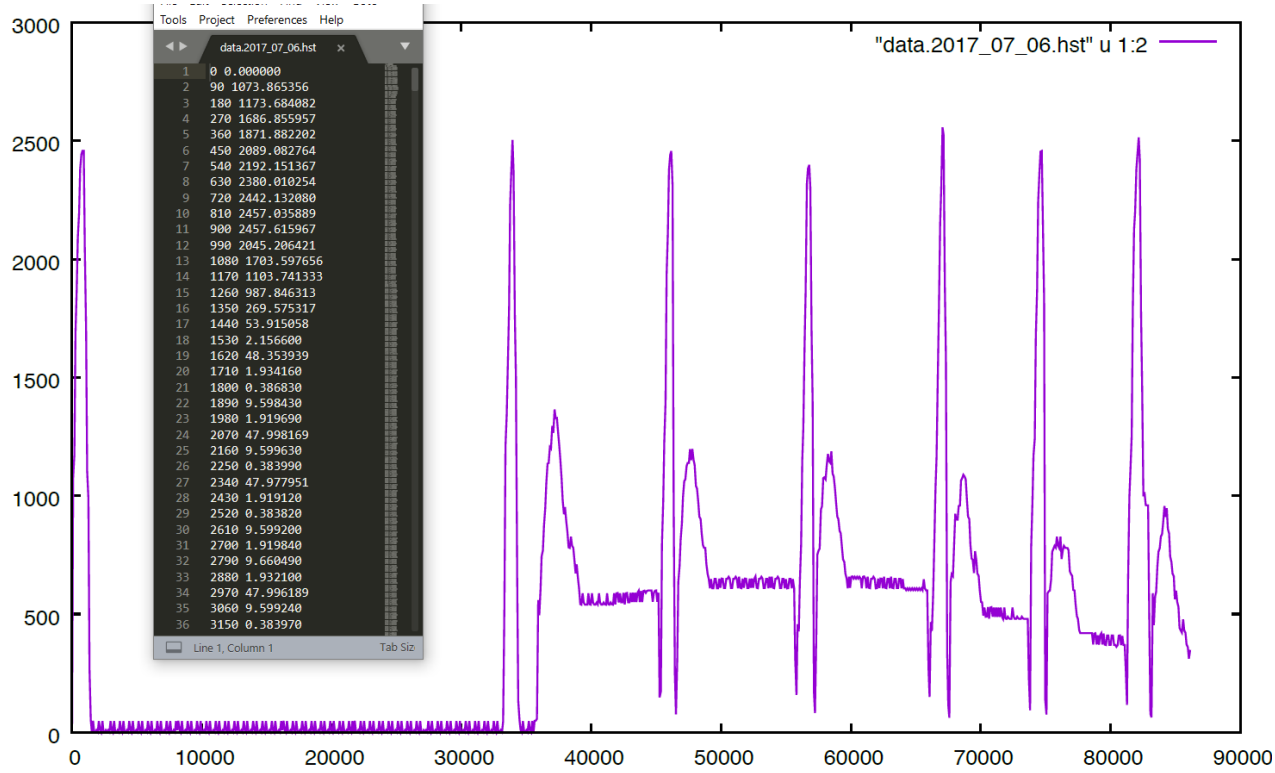

Figura 13 - File del dataset e relativa rappresentazione grafica

Prima di potere utilizzare il dataset è stato necessario eseguire un controllo sui singoli file, in quanto a lezione è stata segnalata la presenza di alcune sequenze composte da valori di consumo nulli da parte della pompa.

Per automatizzare il processo, si è utilizzato un piccolo script in **PowerShell** [5], che determina dapprima l'impronta hash di un file, in seguito ricerca altri file nella stessa directory con lo stesso hash e infine li elimina. Il motivo è che le sequenze composte da valori nulli presentano quasi tutte la stessa lunghezza temporale, comportando di fatto la generazione di alcuni file identici fra loro. Nonostante questo, un controllo manuale dei file ha riscontrato alcune sequenze

nulle “anomale” non eliminate dallo script, per cui il controllo da parte di una persona si rende lo stesso necessario. Al termine del controllo, 218 file sono stati considerati idonei per il processo di apprendimento.

4.1 – Creazione dei modelli

L’andamento del consumo della pompa, visibile dai grafici forniti insieme al dataset, evidenzia delle differenze sostanziali durante la modalità di raffreddamento e riscaldamento. Risulta quindi necessario creare due HMM differenti, uno per la pompa in riscaldamento, l’altro per il raffreddamento.

Per la creazione dei modelli è necessario superare due problemi differenti:

1. La determinazione del numero di stati per ciascun modello.
2. I valori iniziali per le matrici di transizione, emissione e il vettore delle probabilità iniziali

Il primo quesito si risolve osservando i grafici: è infatti possibile notare alcune ridondanze nel loro andamento, segno che la pompa opera in un numero finito di stati.

Nel caso del modello per il riscaldamento, prendendo una sequenza di osservazioni prodotta in un giorno del mese di gennaio, si possono mettere in evidenza i possibili stati differenti che la pompa di calore ha assunto durante le rilevazioni.

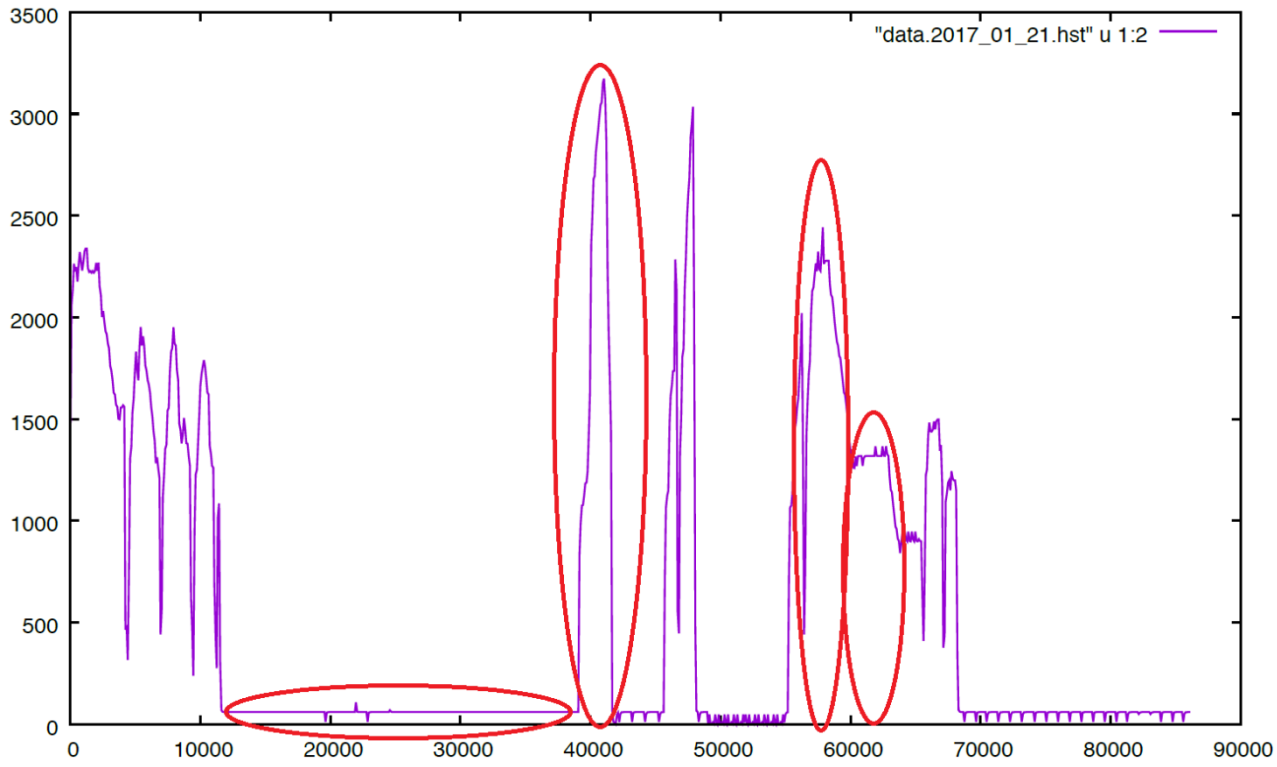

Figura 14 - Individuazione stati differenti per il modello della pompa in riscaldamento

In analogia, prendendo una sequenza prodotta durante il mese di luglio, è possibile individuare gli stati differenti ridondanti.

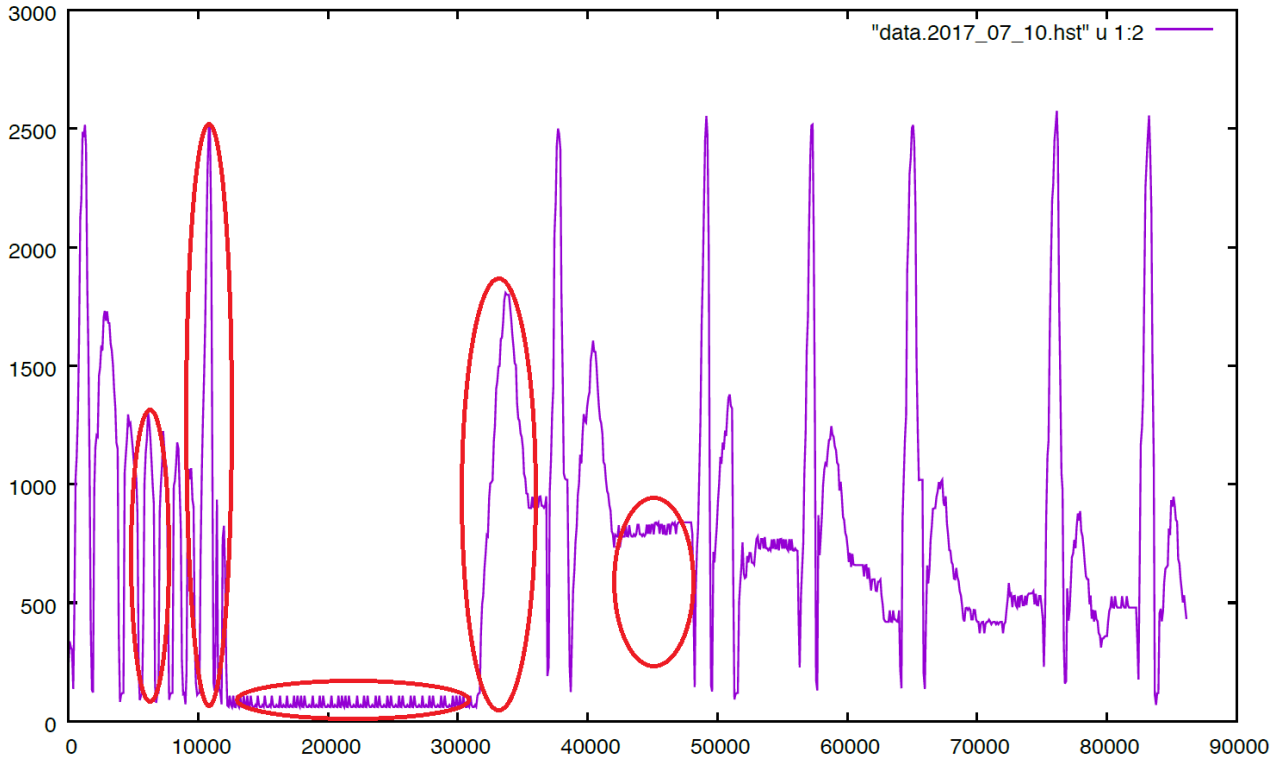

Figura 15 - Individuazione stati differenti per il modello della pompa in raffreddamento

Si assume quindi che:

- Il modello per la pompa in riscaldamento abbia quattro stati differenti
- Il modello per la pompa in raffreddamento abbia cinque stati differenti

Non è tuttavia scontato che gli stati effettivi possano essere in numero maggiore rispetto a quelli determinati.

Per la risoluzione del secondo problema, si possono stabilire le seguenti regole:

- Ogni elemento del vettore delle probabilità di essere in uno stato al tempo t=0 ha probabilità equa $(\frac{1}{numero\_stati})$
- Ogni stato ha una probabilità equa di transire verso sé stesso od un altro stato $(\frac{1}{numero\_stati})$
- Ogni stato presenta una mistura di gaussiane in cui è presente una componente dominante (con peso 0.8 circa) differente per ogni stato e $numero\_stati - 1$ altre componenti con peso equo $(\frac{1.0-0.8}{numero\_stati-1})$
- Per i parametri iniziali delle Gaussiane, risulta non essere scontata la soluzione, in quanto è necessario eseguire una "polarizzazione" di queste rispetto al dataset. La

mancanza o cattiva esecuzione di questa procedura può produrre modelli con una pessima, se non nulla, qualità.
Le soluzioni vagliate hanno previsto tecniche di Clustering per determinare il valore iniziale di queste: su suggerimento del docente, si è adottata la tecnica del K-Means mediante uno script in Python [6] sull'intero dataset del riscaldamento e raffreddamento per determinare i centroidi, utilizzati poi come media dei singoli stati. Per la varianza si è impostato un valore arbitrario di 250, in quanto il K-Means non fornisce direttamente la varianza (occorrerebbe calcolare la massima distanza fra il centroide del cluster e il punto più lontano appartenente al cluster).
Esistono comunque tecniche più efficienti e precise per determinare i parametri delle gaussiane, come ad esempio l'algoritmo di Expectation Maximization [7], che preso in input un dataset, restituisce una mistura di gaussiane con media, varianza e peso di ciascuna componente.

4.2 – Associazione dei dataset ai modelli

Determinati i parametri fondamentali per i modelli, si deve stabilire il dataset da associare a ciascuno di questi.

Il metodo di definizione di un dataset presentato nel **paragrafo 2.3.1** prevede di potere definire differenti indici di file che costituiscono il dataset, senza dovere necessariamente spostare o modificare i singoli file da una directory all'altra. Questo rende particolarmente semplice creare in pochi secondi dataset differenti da provare a sottoporre all'algoritmo di Baum-Welch durante l'allenamento dei modelli.

I dataset creati vertono sullo spostamento di più sequenze da associare a un modello all'altro, in particolare delle sequenze registrate nella stagione **autunnale**, dove si riscontrano giorni in cui la **termopompa** opera in regime di riscaldamento alternati a quelli in cui si registra un'attività per il raffreddamento. Il motivo per cui non si considera anche la stagione primaverile è il fatto che per tutta la stagione non è stata registrata alcuna attività della pompa e le corrispondenti sequenze sono state rimosse dallo script presentato nell'introduzione al capitolo **4**.

Vengono in seguito presentate le suddivisioni dei dati, su cui sono stati eseguiti gli esperimenti:

- **Suddivisione A / Riscaldamento (winterA.dsi)**: 05/01/17 – 26/03/17 e 15/09/17 – 16/12/17
- **Suddivisione A / Raffreddamento (summerA.dsi)**: 28/06/17 – 14/09/17
- **Suddivisione B / Riscaldamento (winterB.dsi)**: 05/01/17 – 26/03/17 e 01/10/17 – 16/12/17
- **Suddivisione B / Raffreddamento (summerB.dsi)**: 28/06/17 – 30/09/17

4.3 – Validazione del classificatore

Per la validazione del classificatore, **Continuous HMM Solver** utilizza la tecnica di Validazione Incrociata (Cross-Validation) a K-Fold [8], utilizzata in ambito di machine learning per stimare la qualità di un modello su dati non osservati. Si utilizza un campione ridotto per stimare quanto possa essere la qualità del modello quando utilizzato per realizzare predizioni su dati non utilizzati per l'allenamento del modello.

I passi della Validazione Incrociata possono essere riassunti in seguito:

1. Rimescolamento del dataset
2. Suddivisione del dataset in K gruppi
3. Per ogni gruppo
    a. Estrazione del gruppo dal dataset per utilizzo come test set
    b. Considerazione dei gruppi rimanenti come learning set
    c. Allenamento del modello mediante Baum-Welch con il learning set e valutazione di questo con il test set
    d. Salvataggio del punteggio di valutazione e scarto del modello
4. Valutazione della qualità del modello usando la media dei punteggi ottenuti da ogni fold

La fase di Validazione Incrociata implementata nell'applicazione prevede di calcolare il punteggio di un modello secondo la **matrice di confusione**, composta da due righe e due colonne e riportante il numero di **veri positivi**, **falsi positivi**, **veri negativi** e **falsi negativi**.

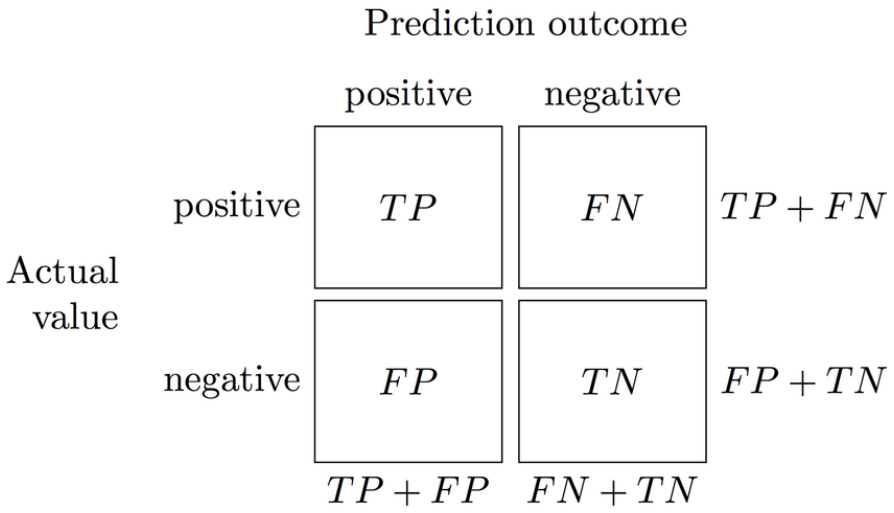

Figura 16 - Tabella di confusione

Da queste misurazioni è possibile determinare alcuni indici di prestazione:

- **Accuratezza**: rapporto fra il numero di osservazioni corrette e il numero totale di osservazioni

$$accuracy = \frac{true\ positive + true\ negative}{number\ of\ instances}$$

- **Frequenza di errori**: rapporto fra il numero di osservazioni errate e il numero totale di osservazioni

$$error\ rate = \frac{false\ positive + false\ negative}{number\ of\ instances}$$

- **Precisione**: rapporto fra il numero di osservazioni classificate correttamente e il numero totale di osservazioni etichettate come positive

$$precision = \frac{true\ positive}{true\ positive + false\ positive}$$

- **Recall**: rapporto fra il numero di osservazioni classificate correttamente e il numero totale di osservazioni che dovevano essere classificate come positive

$$recall = \frac{true\ positive}{true\ positive + false\ negative}$$

4.4 – Risultati della validazione e classificazione

Per il numero di Fold della Validazione Incrociata si è utilizzato il valore 10, suggerito da molti studiosi di machine learning in quanto si tratta di un valore che sperimentalmente produce una stima della qualità del modello con bias ridotto e varianza modesta.

Sono in seguito riportati i risultati ottenuti durante la fase di validazione del classificatore con l'utilizzo dei modelli illustrati in precedenza e delle due suddivisioni del dataset per confrontarne i risultati:

- **K = 10, Suddivisione A, Vero Positivo = sequenza prodotta dalla pompa in raffreddamento, Vero Negativo = sequenza prodotta dalla pompa in riscaldamento, 21 elementi del test set per fold.**

Fold	Accuratezza	Errore	Precisione	Recall	TP	FP	TN	FN
1	61.90%	38.10%	100.0%	46.67%	7	0	6	8
2	90.48%	9.52%	71.43%	100.0%	5	2	14	0
3	90.48%	9.52%	71.43%	100.0%	5	2	14	0
4	95.24%	4.76%	85.71%	100.0%	6	1	14	0
5	85.71%	14.29%	100.0%	70.0%	7	0	11	3
6	85.71%	14.29%	57.14%	100.0%	4	3	14	0
7	61.90%	38.10%	100.0%	46.67%	7	0	6	8
8	42.86%	57.14%	100.0%	36.84%	7	0	2	12
9	71.42%	28.58%	100.0%	53.85%	7	0	8	6
10	100.0%	0.00%	100.0%	100.0%	7	0	14	0
Media	**78.57%**	**21.43%**	**88.57%**	**75.40%**	-	-	-	-

- **K = 10, Suddivisione B, Vero Positivo = sequenza prodotta dalla pompa in raffreddamento, Vero Negativo = sequenza prodotta dalla pompa in riscaldamento, 21 elementi del test set per fold.**

Fold	Accuratezza	Errore	Precisione	Recall	TP	FP	TN	FN
1	71.43%	28.57%	100.0%	60.0%	9	0	6	6
2	90.48%	9.52%	77.78%	100.0%	7	2	12	0
3	80.95%	19.05%	100.0%	69.23%	9	0	8	4
4	80.95%	19.05%	100.0%	69.23%	9	0	8	4
5	71.43%	28.57%	100.0%	60.0%	9	0	6	6
6	71.43%	28.57%	100.0%	60.0%	9	0	6	6
7	90.48%	9.52%	100.0%	81.82%	9	0	10	2
8	85.71%	14.29%	88.89%	80.0%	8	1	10	2
9	80.95%	19.05%	100.0%	69.23%	9	0	8	4
10	85.71%	14.29%	100.0%	75.0%	9	0	9	3
Media	**80.95%**	**19.05%**	**96.67%**	**72.45%**	-	-	-	-

I risultati presentati sono confermati dall'utilizzo in un caso generico del classificatore in seguito all'allenamento dei modelli mediante la funzione **Train Model** dell'applicazione.

Previa rimozione e annotazione in un file separato di alcune sequenze da un dataset, ad esempio quello etichettato come **suddivisione A** (con la creazione dei file di indice **winterA_test.dsi** e **summerA_test.dsi**), si allenano i singoli modelli, per poi procedere alla sottomissione del classificatore delle sequenze di osservazioni rimosse in precedenza mediante la funzione **Test sequence**.

I risultati ottenuti sono corretti rispetto al dataset dal quale le rispettive sequenze sono state estratte:

Nome file sequenza	log[P(sequenza | summer Model)	log[P(sequenza | winterModel)]	Atteso	Effettivo
data.2017_01_12.hst	6477.813131970134	4202.776386441207	winterModel	winterModel
data.2017_01_13.hst	6432.749538320009	4329.066982531899	winterModel	winterModel
data.2017_01_14.hst	6088.179118069385	4685.972331490983	winterModel	winterModel
data.2017_01_15.hst	5810.167856903872	4067.4919364511857	winterModel	winterModel
data.2017_09_16.hst	4441.603393676366	4339.919786205181	winterModel	winterModel

data.2017_09_17.hst	4437.22591626912	4338.87411969486	winterModel	winterModel
data.2017_09_18.hst	4512.888592487438	4406.736552950849	winterModel	winterModel
data.2017_09_19.hst	4493.772525342585	4389.142717138516	winterModel	winterModel
data.2017_11_19.hst	5258.255509142559	3954.523920792024	winterModel	winterModel
data.2017_11_20.hst	5692.159606938386	4890.366670296 5365	winterModel	winterModel
data.2017_11_21.hst	5659.55125192972	4897.426991441054	winterModel	winterModel
data.2017_11_22.hst	5904.581994777958	4549.095661536647	winterModel	winterModel
data.2017_11_23.hst	5125.256332482168	4863.978417506566	winterModel	winterModel
data.2017_12_06.hst	6172.993099253877	4606.932462228659	winterModel	winterModel
data.2017_12_07.hst	6507.9493941586215	5192.080377417 96	winterModel	winterModel
data.2017_07_14.hst	6106.946694706773	7630.043466967045	summerModel	summerModel
data.2017_07_15.hst	6108.7108659255455	7641.960505578122	summerModel	summerModel
data.2017_08_15.hst	5728.060534443196	7029.863535785458	summerModel	summerModel
data.2017_08_16.hst	6013.861323296502	7634.059921794559	summerModel	summerModel
data.2017_08_17.hst	6001.898270370947	7641.498224786428	summerModel	summerModel
data.2017_08_18.hst	3617.678086480028	4611.997731271 9675	summerModel	summerModel
data.2017_08_19.hst	5948.904630139642	7623.564146184135	summerModel	summerModel
data.2017_09_05.hst	4463.189363655585	4385.851632030736	summerModel	**winterModel**
data.2017_09_06.hst	4881.692992316295	5179.722868356471	summerModel	summerModel

Fra tutti i risultati ottenuti, solo una sequenza di test è risultata essere classificata in modo non corretto. Il motivo per questa classificazione errata può essere dovuto al fatto che la **termopompa** avesse funzionato in riscaldamento nel giorno considerato, in quanto non è raro imbattersi in giornate con temperature inferiori alla media nel mese di settembre.

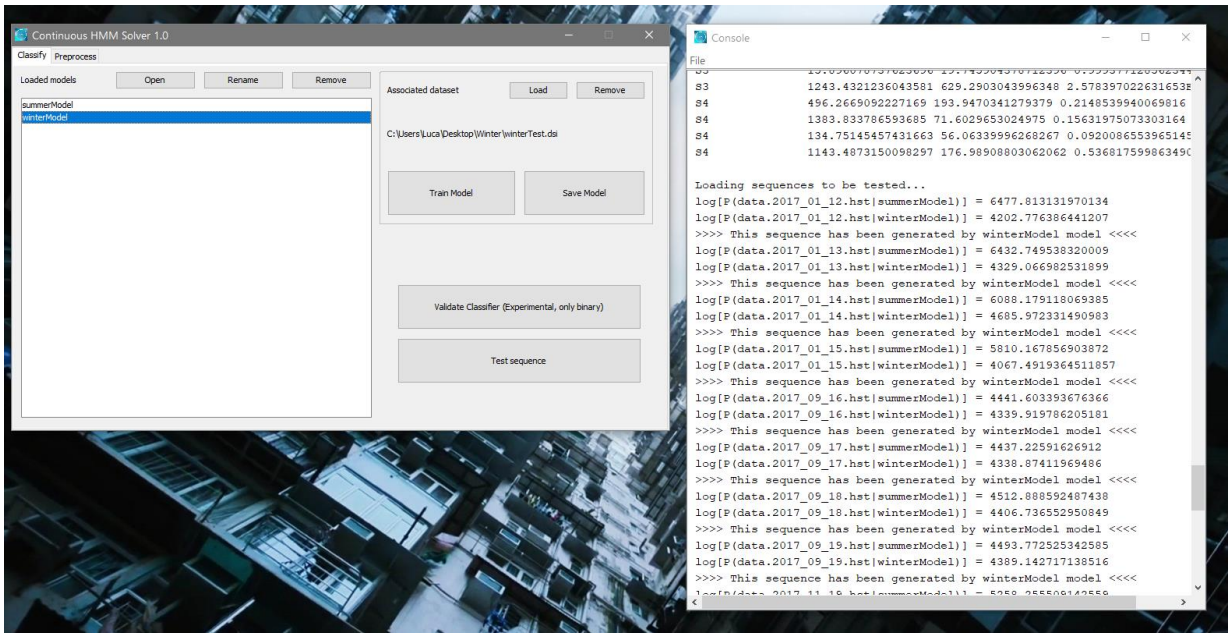

Figura 17 - Screenshot dei risultati presentati in tabella

Un ultimo commento viene destinato alle sequenze di stati ottenute mediante l'algoritmo di **Viterbi**, non riportate nel documento per questioni di spazio e risultanti essere etichettate correttamente rispetto a quanto riportato dal grafico corrispondente alla sequenza. Essendoci presenti errori di classificazione, **Continuous HMM Model** prevede il salvataggio delle sequenze di stati per entrambi i modelli. L'utente sarà in seguito incaricato di determinare se effettivamente la sequenza sia prodotta dal modello per il raffreddamento o riscaldamento e potrà consultare il rispettivo file prodotto dall'applicazione.

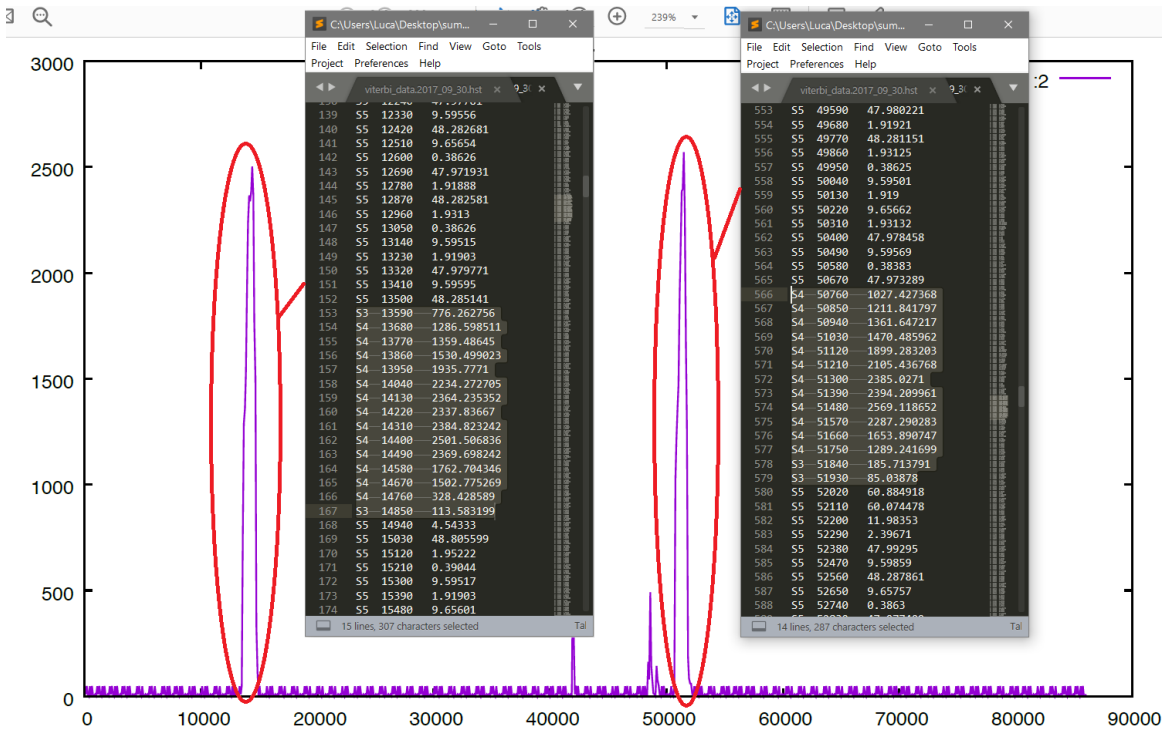

Figura 18 - Risultati sequenza di stati con Viterbi

5 – Conclusioni e sviluppi futuri

Il progetto ha avuto modo di potere approfondire ed applicare le conoscenze acquisite durante il corso di Estrazione e Rappresentazione della Conoscenza ad un caso d'uso reale. Sempre più spesso, il machine learning entra in contatto con la vita quotidiana di una persona, che essa ne sia a conoscenza o meno. Ne è un esempio la massiccia diffusione di dispositivi IoT che implementano anche conoscenze presentate durante il corso per le applicazioni pratiche più disparate.

Numerose ore sono state spese al fine di creare un'applicazione che potesse essere ben progettata e collaudata, ma nonostante questo numerose idee non sono state implementate per mancanza di tempo:

1. Implementare un modulo di Validazione più sofisticato, che possa tenere in considerazione ulteriori indicatori e supporti la matrice di confusione per N modelli differenti, anziché il limite di due.
2. Inserire messaggi di progresso per le fasi di training e validazione incrociata, in modo che l'utente possa visualizzare lo stato dell'apprendimento.
3. Aggiungere una funzione per la generazione di sequenze a partire da un modello caricato dall'applicazione.
4. Integrare direttamente nell'applicazione gli strumenti di "pulizia" del dataset da sequenze nulle e polarizzazione delle distribuzioni normali, senza doversi appoggiare a script esterno.
5. Supporto alle sequenze di osservazioni discrete.
6. Riscrittura del progetto in C++ o C#, al fine di aumentare la velocità complessiva dell'applicazione.

Altro miglioramento è la creazione di un dataset di dimensioni maggiori e le cui sequenze sono etichettate correttamente, in modo da ottenere risultati più precisi e attendibili.

Bibliografia

[1] Eugine Kang, *Hidden Markov Model*, https://medium.com/@kangeugine/hidden-markov-model-7681c22f5b9, consultato il 3 settembre 2018.

[2] A.A.V.V., *Wikipedia - Hidden Markov model*, https://en.wikipedia.org/wiki/Hidden_Markov_model, consultato il 3 settembre 2018.

[3] L. R. Rabiner, *A Tutorial of Hidden Markov Models and Selected Applications in Speech Recognition*, Proceedings of the IEEE, Vol. 77, No. 2, pagg. 257-273, febbraio 1989.

[4] A. Rahimi, *An Erratum for "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition"*, http://alumni.media.mit.edu/~rahimi/rabiner/rabiner-errata/, consultato il 2 febbraio 2018.

[5] A.A.V.V., *Trying to compare hashes and delete files with same hash in powershell,* https://stackoverflow.com/questions/44358602/trying-to-compare-hashes-and-delete-files-with-same-hash-in-powershell, consultato il 17 novembre 2018.

[6] A.A.V.V., *SciKIT Learn Documentation – sklearn.cluster.KMeans,* https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html, consultato il 14 dicembre 2018.

[7] A.A.V.V., *Machine Learning with Python: Expectation Maximization and Gaussian Mixture Models in Python*, https://www.python-course.eu/expectation_maximization_and_gaussian_mixture_models.php, consultato il 14 dicembre 2018.

[8] Jason Brownlee, *A Gentle Introduction to k-fold Cross-Validation*, https://machinelearningmastery.com/k-fold-cross-validation/, consultato il 2 gennaio 2019.