Modellizzazione e predizione di segnali in reti di sensori tramite tecniche di pattern recognition

Mattia Bonaventura

25 Ottobre 2011

Ai miei nonni,

Toni, Giuliana, Angelo e Chicca,

perché senza memoria del passato

non si può immaginare il futuro.

Sommario

Il campo delle reti di sensori wireless ha conosciuto, negli ultimi anni, uno sviluppo inarrestabile; l'affinamento delle tecnologie e lo sviluppo di protocolli specifici ha contribuito ad aumentare l'affidabilità di tali sistemi, i quali ora iniziano a venire utilizzati con profitto in alcuni campi industriali. Abbiamo sviluppato un sistema di elaborazione che, sulla base dei dati raccolti da sensori di movimento disseminati nella rete, è in grado di costruire un modello riguardo le sequenze di occupazione dei vari locali. Tale modello è costruito durante una fase di apprendimento secondo i principi del machine learning, effettuando una classificazione di alcune fra le sottosequenze della serie temporale; esso viene poi utilizzato dal sistema per interpretare i dati forniti in tempo reale dai sensori, allo scopo di fare delle previsioni sull'evoluzione dell'occupazione dei locali nel futuro prossimo. Non è necessario programmare in alcun modo il sistema poiché esso stesso è in grado di costruire il modello e, successivamente, di identificare le varie sottosequenze all'interno di esso, grazie all'utilizzo di tecniche di pattern recognition. Crediamo che il nostro sistema possa trovare diverse applicazioni nel campo della domotica e del risparmio energetico in ambienti domestici e lavorativi.

Abstract

Wireless Sensor Networks (WSN) research field has grown exponentially in the last few years; improvements made on the technologies used, and the development of ad-hoc protocols, contribuited to improve reliability of these systems, which now are starting to be used in some industrial applications. We developed an elaboration system which, feeded from data collected by motion detectors placed in the network, is able to build a model regarding occupation sequences for the sites where they are placed. This model is built during a learning phase in a machine learning fashion, by means of clustering of time-series subsequences; it is then used by the system to interpret data coming in real-time from sensors, in order to make forecasts about the next-future evolution of the rooms occupancies. It is not necessary to program the system in any way, for example giving it rules for premises occupations or time references: it is the system itself which infers these rules autonomally observing the input sequences, by means of pattern recognition techniques. We believe that our system could find applications in the domotics and energy-saving field, in both domestic and office environments.

Ringraziamenti

Sento il dovere di ringraziare brevemente tutte le persone che mi hanno sostenuto ed incoraggiato permettendomi di arrivare fin qui.

Ringrazio il mio relatore, Michele Rossi, per avermi dato l'idea per questo lavoro, per aver portato pazienza con me e per il continuo supporto fornito durante la realizzazione del progetto; ringrazio inoltre tutti i ragazzi, tesisti e dottorandi, del laboratorio *SIGNET*, in particolare Matteo Danieletto, per gli utili scambi di idee, consigli e suggerimenti.

Voglio ringraziare anche tutti i miei superiori e i miei colleghi in *SAVE*, per la fiducia che hanno riposto in me e per la disponibilità che mi hanno concesso per agevolarmi nella conclusione di questo lavoro.

Grazie ai miei genitori, per non avermi fatto mai mancare niente, e per avermi insegnato cose che in nessuna Università al mondo si possono imparare, e che mi hanno permesso di diventare quello che sono; grazie a mia sorella Silvia e a tutti i parenti che in diversi modi mi sono stati vicini in questi anni. Grazie ai miei compagni di sempre, Ermanno e Nicolò, e a tutti gli amici che non mi hanno mai fatto mancare la loro compagnia. Infine, grazie ad Alessia, per tutti i motivi che io e te sappiamo e che, semplicemente, non possono starci tutti qui. Sono cosciente che, senza l'aiuto di tutti voi, non ce l'avrei fatta ad arrivare a questo punto; ed ogni giorno cerco di non dimenticarmi di quanto sono fortunato ad avervi accanto.

Indice

Capitolo 1

Introduzione

Il settore delle reti wireless di sensori (WSN, *Wireless Sensor Networks*) è, nel campo delle telecomunicazioni, fra quelli che hanno conosciuto la crescita più sostenuta nel corso degli ultimi anni. Si tratta di reti di dispositivi autonomi (detti *nodi*), ognuno dei quali è equipaggiato con uno o più sensori in grado di monitorare le caratteristiche ambientali del luogo in cui si trova (come temperatura, pressione, umidità, rumore...). I nodi devono, generalmente, trasmettere i dati relativi verso un sistema centrale; questa trasmissione, nella maggior parte delle WSN, non avviene tramite una comunicazione diretta con l'host principale, ma tramite metodi collaborativi, che coinvolgono i nodi intermedi fra quello trasmittente e quello ricevente, facendo far loro da *relay*. La trasmissione diretta, infatti, è quasi sempre impossibile per via delle prestazioni limitate dei nodi, dovute al fatto che l'obiettivo primario nel progetto di questi dispositivi è il risparmio di energia, volto a massimizzare la durata delle batterie di cui sono dotati.

Le applicazioni delle WSN sono ormai molteplici; il loro sviluppo iniziale era dettato da ragioni militari (ad esempio, per monitorare zone sensibili), ma ormai le applicazioni sono diffuse in diversi campi a livello industriale,

commerciale e di ricerca: si può pensare al controllo di processi chimici in impianti industriali, oppure al monitoraggio delle maree o dell'inquinamento ambientale. In un certo senso, basta trovare il sensore adatto, e le potenziali applicazioni sono infinite.

Nel nostro lavoro, non entreremo nei dettagli del funzionamento della nostra rete di sensori; in altre parole, non ci occuperemo dei metodi di trasmissione o di instradamento dei messaggi, dei protocolli di comunicazione, o di qualsiasi altro aspetto legato all'implementazione della rete. Il nostro sistema *utilizzerà* la rete di sensori, o meglio i servizi che essa offre (essenzialmente, uno: la trasmissione dei dati dai sensori verso l'host centrale). In definitiva, il nostro lavoro rappresenta una possibile applicazione delle reti di sensori, pensata per un'ambito domestico o di ufficio.

Vogliamo equipaggiare i nostri nodi con un sensore di presenza, in modo da registrare l'occupazione dei locali dove essi sono posizionati. Le sequenze rilevate dai sensori saranno poi trasmesse ad un sistema centrale fisso, dove saranno elaborate per costruire un modello delle serie temporali di occupazione registrate. Lo scopo del nostro lavoro è di costruire un modello di queste sequenze di occupazione che sia sufficientemente accurato da consentire di prevedere l'occupazione del locale nel futuro prossimo. In altre parole, dopo una fase iniziale di apprendimento, durante la quale il sistema accumula informazioni sulle sequenze di occupazione caratteristiche del locale, vogliamo che il sistema sia in grado di identificare la sottosequenza rilevata correntemente dal sensore per prevedere quale sarà l'occupazione futura della stanza. Non è previsto alcun intervento da parte dell'utente sul sistema: non sarà necessario programmare il sensore inserendo manualmente degli esempi di sequenza, oppure impostando alcune regole sugli orari di occupazione del locale; sarà il sistema stesso ad inferire queste regole durante la fase di

apprendimento. L'intervento da parte dell'utente si limiterà, quindi, al posizionamento del sensore nel locale: da quel momento, esso dovrà funzionare in completa autonomia.

Inoltre, il sistema dovrà essere in grado di aggiornare costantemente e gradualmente il modello costruito durante la fase di apprendimento, anche una volta dopo che essa sia terminata; le informazioni provenienti dalle nuove sottosequenze, quindi, dovranno contribuire all'evoluzione del modello, permettendogli di "imparare" tipi di sequenze che non si sono mai verificati prima e, al contrario, di "dimenticare" quelli che non si verificano per un certo periodo di tempo.

Crediamo che un'applicazione di questo genere possa essere di interesse in diversi settori; ad esempio nel campo della domotica, consentirebbe di avere sistemi di riscaldamento/condizionamento "intelligenti", in grado di attivarsi solamente nelle zone della casa cui è prevista l'occupazione nel prossimo futuro. Questo avrebbe sicuramente l'effetto di ridurre la quantità di energia utilizzata inutilmente per climatizzare abitazioni o uffici quando essi sono, in realtà, vuoti.

Durante la realizzazione del sistema, ci siamo dovuti occupare di diversi aspetti: dalla scelta dei componenti alla scrittura dell'algoritmo, dalla ricerca teorica all'implementazione hardware; ognuno di questi aspetti sarà ampiamente documentato nel corso di questo lavoro. Nel capitolo 2 si possono trovare tutti i dettagli relativi alla costruzione hardware del sensore; nel capitolo 3, vedremo quali tecniche utilizzare per la costruzione del modello di occupazione, affrontando alcune problematiche di carattere teorico; nel capitolo 4, si trovano i dettagli riguardo la realizzazione software del sistema, in pratica la traduzione in un algoritmo stabile ed efficiente dei concetti introdotti nel capitolo precedente; infine, nel capitolo 5, si potranno trovare

tutti gli esperimenti e le prove che abbiamo condotto per testare il nostro sistema, illustrando i risultati che siamo riusciti ad ottenere.

Capitolo 2

Rilevazione dei dati sperimentali

Una delle prime sfide da affrontare nell'ambito del nostro progetto è stata quella di realizzare il modulo che ci avrebbe consentito di raccogliere i dati sperimentali relativi all'occupazione del locale. Ricordiamo che il nostro obiettivo era quello di realizzare un modulo completamente *stand-alone* il quale, una volta posizionato nel locale da monitorare, grazie all'integrazione nella rete di sensori avrebbe potuto trasmettere i dati al server. Il sistema è stato progettato per funzionare on-line: ad intervalli di tempo prefissati, il modulo invia i dati al server, che li elabora istantaneamente e identifica la sequenza corrente all'interno del modello costituito fino a quel momento. Tuttavia, nelle nostre prove di funzionamento, abbiamo utilizzato il modulo prevalentemente off-line: il sensore invia sempre ad intervalli periodici i dati al server, il quale però li immagazzina costruendo una lunga sequenza che poi sarebbe stata utilizzata per l'elaborazione successiva.

2.1 La scelta dei componenti

Dovendo iniziare il progetto praticamente da zero, per prima cosa è stato necessario scegliere i componenti da utilizzare. Nelle due sezioni successive si trovano i dettagli riguardo al modulo wireless e al rilevatore di presenza che abbiamo scelto.

2.1.1 Il modulo wireless

Come base per la parte hardware del modulo abbiamo utilizzato dei sensori *TelosB*, prodotti da Crossbow; questo tipo di sensore era già in uso nel laboratorio *Signet* del Dipartimento di Ingegneria dell'Informazione, dunque per la sua disponibilità e per compatibilità con la rete di sensori esistente abbiamo deciso di utilizzarlo come base per il nostro sistema. Le caratteristiche del dispositivo sono molto interessanti: dispone di un ricetrasmettitore wireless 802.15.4 con antenna integrata ed è caratterizzato da consumi di energia particolarmente ridotti; inoltre, caratteristica fondamentale per il nostro progetto, dispone di 16 pin designati per collegare moduli aggiuntivi, che utilizzeremo per collegare il nostro sensore di movimento. Il TelosB può funzionare alimentato da due batterie AA, oppure collegato ad una presa USB.

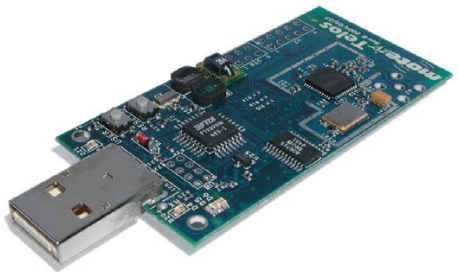

Figura 2.1: Il modulo wireless TelosB

TelosB funziona con il sistema operativo *TinyOS*, un progetto open-source nato dalla collaborazione fra Intel e la University of California. Si tratta di un sistema operativo sviluppato appositamente per dispositivi wireless a basso consumo di energia. Per ottenere una massima efficienza energetica, TinyOS non offre la possibilità di multithreading (il sistema esegue una sola applicazione per volta); inoltre, per sfruttare al meglio la memoria disponibile (TelosB è equipaggiato con una RAM da 10kB), è basato su un'architettura a componenti, che permettono di 'cucire su misura' il sistema operativo per adattarlo alle nostre esigenze, attivando solamente i servizi necessari. TinyOS è scritto in nesC (*network embedded system C*), un C-dialect progettato appositamente per adattarsi al meglio alle caratteristiche di TinyOS: sarà il linguaggio di programmazione che utilizzeremo anche noi per scrivere le nostre applicazioni.

2.1.2 Il rilevatore di presenza

Il passo successivo è stato quello di ricercare sul mercato un sensore che potesse essere adatto al nostro scopo di rilevare l'occupazione di un ambiente. Gli elementi discriminatori nella ricerca sono stati essenzialmente cinque:

- **Voltaggio** Dovevamo necessariamente trovare un sensore che potesse funzionare entro il voltaggio massimo garantito dal TelosB (3V).

- **Consumo di energia** Doveva essere basso, nell'ordine dei milliampere o, alternativamente, doveva essere possibile spegnere il sensore negli intervalli fra una lettura e l'altra, senza perderne le funzionalità alla riattivazione.

- **Dimensioni** Il sensore doveva avere avere dimensioni comparabili, o ancor meglio minori, rispetto a quelle del TelosB (che misura 3.2 cm

in lunghezza e 6.5 cm in larghezza).

- **Prezzo** Ultimo ma non meno importante, il costo del sensore ricopre un peso importante perché va ad incidere sulla concreta realizzabilità del progetto oltre i fini solamente accademici.

La nostra attenzione si è infine concentrata su due sensori basati su tecnologie differenti: un sensore ad infrarossi (PIR) prodotto dalla Parallax, e un sensore SONAR della MaxBotix: li abbiamo messi a confronto sui requisiti che avevamo individuato, oltre che su un altro parametro fondamentale quale il raggio di rilevazione.

	Parallax PIR	MaxBotix SONAR
Voltaggio	da 3 a 5V	da 2.5 a 5.5V
Energia	3 mA	da 2 a 3 mA
Dimensioni	32x24x25 mm	20x22x16 mm
Prezzo	9.99 $	29.95 $
Raggio	6.10 m	3.90 m

In generale si può vedere come entrambi i sensori rispondano ai requisiti prefissati, con prestazioni leggermente migliori del SONAR per quanto riguarda voltaggio, consumo di energia e dimensioni. Al contrario, il PIR può vantare un prezzo decisamente minore (pari ad un terzo rispetto al concorrente) e un raggio di rilevazione nettamente più esteso. Il maggior prezzo del SONAR poteva essere giustificato dal fatto che esso funge non solo da rilevatore di presenza, ma è anche in grado di misurare la distanza dal corpo

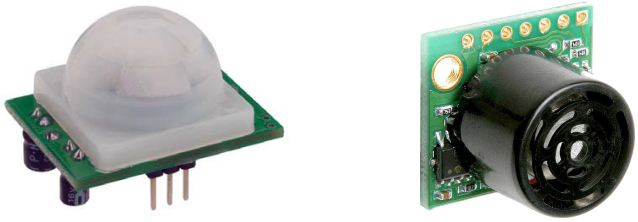

Figura 2.2: I due sensori considerati per la scelta: PIR (a sinistra) e SONAR

rilevato; inoltre, esso dispone anche di tre uscite (seriale, analogica e a larghezza d'impulso) mentre il PIR fornisce come output un semplice segnale a singolo bit high/low. Tuttavia, queste caratteristiche aggiuntive ci sembravano poco utili per i nostri scopi, dunque si è preferito scegliere il sensore ad infrarossi, in virtù della sua maggiore capacità di rilevazione ed economicità.

Vediamo nel dettaglio le caratteristiche di funzionamento di un sensore PIR (*Passive/Pyroelettric InfraRed*). Innanzitutto si deve puntualizzare che un rilevatore di questo tipo costituisce un sistema passivo: il dispositivo non emette raggi infrarossi, ma funziona misurando il livello di radiazioni provenienti dall'ambiente circostante. Queste radiazioni vengono rilevate mediante l'applicazione sul sensore di materiali piroelettrici, che posseggono cioè la capacità di generare una debole differenza di potenziale quando vengono riscaldati, o raffreddati. Solitamente un sensore è composto da due o quattro *pixel* di materiale piroelettrico, i quali poi possono essere collegati, a coppie, all'ingresso di un amplificatore operazionale: in questo modo, la temperatura media dell'ambiente non produrrà una componente continua di tensione, e un incremento della radiazione infrarossa riguardante l'intero sensore (cioè, tutti i pixel) non farà scattare il dispositivo. Dal momento che ogni oggetto emette energia sotto forma di radiazioni, un sensore di questo tipo è in grado di rilevare del movimento quando un corpo ad una determinata temperatura si muove in un ambiente con temperatura differente.

Nello specifico, il sensore che abbiamo scelto per il nostro sistema è equipaggiato anche con una lente di Fresnel, un dispositivo che permette di focalizzare la radiazione infrarossa sulla parte sensibile dell'apparecchio, aumentando la capacità di rilevazione del sistema. Il produttore specifica che per un funzionamento corretto del sensore è necessario un tempo di calibrazione compreso fra i 10 e i 60 secondi, durante il quale si dovrebbe limitare

al massimo il movimento nel campo di rilevazione del dispositivo: dovremo prestare attenzione a questa caratteristica nella progettazione del sistema, per evitare che una calibrazione errata produca risultati non consistenti.

2.2 L'assemblaggio dei componenti

Una volta definiti ed acquistati i due componenti del sistema, abbiamo provveduto a collegarli secondo le specifiche dei produttori. Come abbiamo già accennato, il TelosB possiede 16 pin dedicati appositamente per il collegamento di dispositivi esterni (sensori, display LCD, ecc.). In figura 2.3 si può vedere in maniera più evidente il modulo di espansione, suddiviso in due blocchi da 6 e 10 pin.

Per i nostri scopi sarà sufficiente l'utilizzo del blocco da 10 pin, del quale si possono vedere le caratteristiche in figura 2.4. Le specifiche per il collegamento del sensore sono molto semplici, essendo quest'ultimo dotato di soli 3 pin: due per l'alimentazione (V+ e GND) e uno per la trasmissione del segnale di uscita, come si può vedere in figura 2.5. Il segnale di uscita dal sensore ad infrarossi è un segnale analogico che sarà alto (con una tensione prossima a quella di alimentazione, secondo le specifiche del produttore) nel caso in cui venga rilevato del movimento. Dunque per trasmettere il segnale al TelosB abbiamo pensato di utilizzare, in prima istanza, il modulo ADC (*Analog to Digital Converter*) integrato nel TelosB stesso, collegando il pin di output del rilevatore al pin numero 3 del modulo di espansione. Inoltre abbiamo ovviamente collegato i pin di alimentazione del sensore ai pin numero 1 e 9 del TelosB.

Per collegare il sensore al TelosB abbiamo saldato a quest'ultimo dei connettori adatti ad accogliere i cavetti provenienti dal PIR. In figura 2.6 si può vedere un'immagine del collegamento.

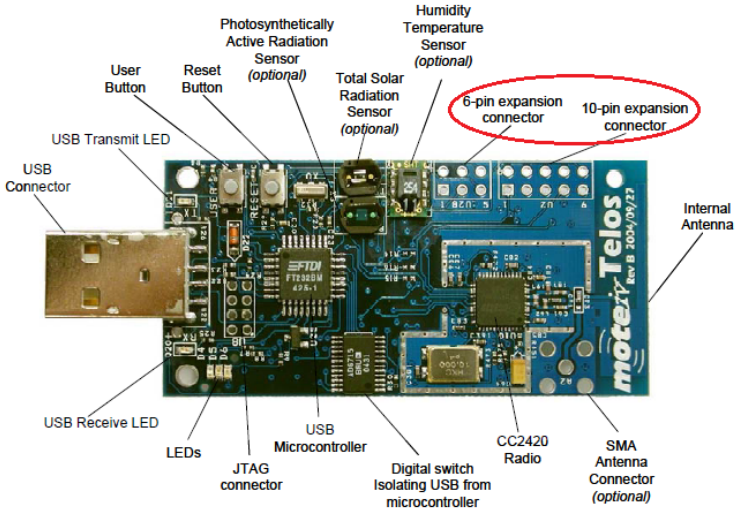

Figura 2.3: Modulo di espansione del TelosB

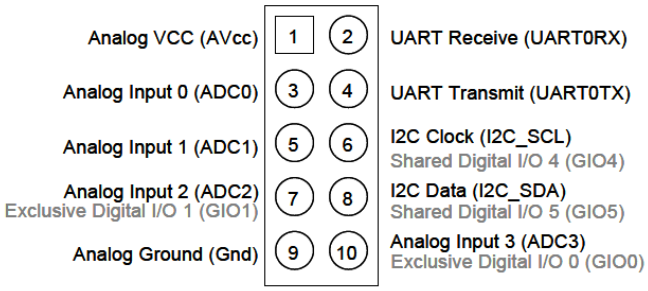

Figura 2.4: Descrizione dei pin del modulo di espansione del TelosB

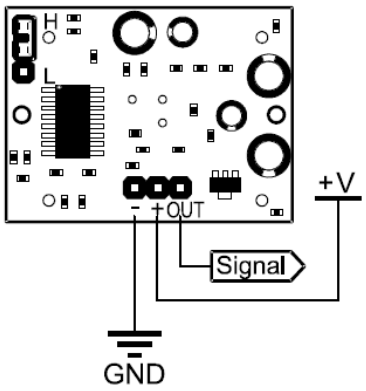

Figura 2.5: Descrizione dei pin per il collegamento del PIR

Figura 2.6: Immagine del collegamento fra il TelosB e il PIR

2.3 Il software per la rilevazione dei dati

Come abbiamo anticipato nella presentazione di TinyOS, esso supporta il linguaggio di programmazione nesC, che utilizzeremo per scrivere i programmi che serviranno per l'acquisizione dei dati dal sensore di movimento. nesC è caratterizzato da una struttura a componenti, con la particolarità che tutte le interazioni fra i componenti si realizzano mediante interfacce; i collegamenti fra i vari componenti, detti *wiring*, sono determinati al momento della compilazione.

Il sensore sarà completamente controllato via software: in un primo momento, si era pensato ad un possibile controllo hardware esterno, che permettesse di togliere l'alimentazione al PIR fra una rilevazione e l'altra, allo scopo di massimizzare il risparmio di energia. Alla fine però le caratteristiche abbastanza stringenti del sensore per quanto riguarda il tempo di calibrazione ci hanno fatto abbandonare questa ipotesi: infatti, era impossibile garantire che ogni volta, alla riaccensione, ci fossero le caratteristiche ambientali di basso

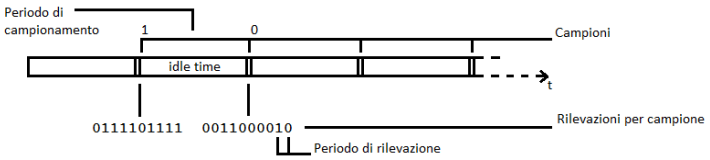

Figura 2.7: Spiegazione grafica del meccanismo di rilevazione

movimento richieste per la corretta calibrazione del sensore; questo fatto, aggiunto alla considerazione che, comunque, il consumo di energia del PIR è relativamente basso, ci ha fatto propendere per un controllo esclusivamente software.

Sono state scritte diverse versioni del programma, ma tutte sostanzialmente eseguivano le stesse operazioni di base: dopo aver configurato il modulo ADC, ne andavano a leggere, ad intervalli prefissati, l'output, il quale a sua volta rappresenta la traduzione numerica del segnale analogico proveniente in quel momento dal sensore. Questo valore numerico, variabile fra 1 e 4096 (caratteristico dell'ADC a 12 bit), veniva confrontato con una soglia fissata esattamente a metà, per stabilire se il segnale letto fosse alto o basso. Si è fatto in modo che, per ogni campione di rilevazione, si svolgessero diverse letture a distanza di qualche secondo, per evitare che una sola lettura errata potesse produrre risultati non coerenti: solo se le letture positive superavano una certa soglia percentuale sul totale delle letture effettuate, allora la lettura corrispondente a quel campione sarebbe stata considerata come una presenza. È stato fatto in modo che periodo di rilevazione e periodo di campionamento fossero due parametri modificabili tramite un file esterno di configurazione: durante le nostre prove, li abbiamo settati rispettivamente a 2 secondi e a 15 minuti. Anche il numero di rilevazioni per campione e la soglia minima per considerare positiva la lettura di un campione sono parametri modificabili.

La varianti del programma riguardano essenzialmente la modalità di trasmissione dei dati. Nella prima versione, il nodo collegato al PIR trasmetteva via radio la lettura relativa ad un campione al sensore collegato all'host fisso. Per realizzare questa versione, naturalmente, sono stati scritti due programmi differenti per i due nodi. Successivamente, dal momento che durante le nostre prove nel laboratorio Signet il TelosB rimaneva sempre collegato alla porta USB, abbiamo implementato una versione 'light' del programma la quale, invece di utilizzare la radio per la trasmissione dei dati, li stampava a video. In entrambi i casi, l'output del modulo collegato al PC veniva salvato, raccogliendo il risultato della stampa in un file di log. Infine è stata preparata una terza versione che, a differenza delle precedenti, legge i dati dalla porta GPIO (*General Purpose Input/Output*) messa a disposizione dal TelosB. In questo caso, il pin di output del rilevatore di movimento è stato collegato al pin numero 7 del TelosB (si veda nuovamente figura 2.4): questo fa sì che il segnale non passi più per l'ADC, ma che venga 'interpretato' direttamente dal sensore come un flusso di bit. Questo sarebbe possibile in virtù del fatto che il PIR che abbiamo utilizzato possiede un jumper il quale, se settato, fa sì che il segnale di output in caso di movimento risulti in una serie di impulsi alto/basso, invece di rimanere fisso a livello alto come di consueto. Purtroppo il produttore non forniva ulteriori dettagli su questa modalità (come ad esempio la frequenza e la larghezza dell'impulso), dunque non è stato possibile regolare il programma in modo da funzionare correttamente: questa versione è stata quindi accantonata, anche perché nel frattempo il funzionamento della variante con ADC si è mostrato molto affidabile.

2.4 Verifica del funzionamento del rilevatore di presenza

Un'altra fase preliminare nello sviluppo del sistema è stata quella di verifica del funzionamento del meccanismo di rilevazione dei dati. Si è preferito verificarne subito il funzionamento per poter iniziare ad utilizzare i sensori e a raccogliere i dati; era indispensabile accertarsi che i dati che avremmo raccolto, e poi utilizzato per testare il sistema, fossero coerenti.

Inizialmente abbiamo incontrato alcuni problemi poiché non riuscivamo ad ottenere dati stabili dal sistema. Sembrava che, quando il sensore rilevava per la prima volta un movimento nel suo raggio d'azione, poi l'output non tornasse più a livello basso, anche quando ormai il movimento si era esaurito. Scoprire quali erano le cause del malfunzionamento non è stato facile, essenzialmente perché non sapevamo qual era il punto esatto del sistema che provocava l'errore. Poteva trattarsi:

- di un errore nel programma scritto per acquisire i dati. Dopo aver controllato nel dettaglio l'output del programma abbiamo visto, però, che i dati arrivavano già sbagliati dall'ADC;

- di errori nella configurazione dell'ADC del TelosB. Per verificare che non fosse quello il problema, abbiamo misurato con un multimetro il segnale in uscita dal sensore, constatando che l'ADC stava funzionando a dovere: era proprio il PIR a dare quel segnale in uscita;

- di una calibrazione non adeguata del sensore di movimento. Tuttavia, dopo aver fatto diverse prove facendo particolare attenzione perché non ci fosse alcun movimento nel raggio di rilevazione del sensore, abbiamo visto che il problema persisteva;

- di un difetto di quel particolare esemplare di sensore. Poiché ne avevamo acquistati diversi, abbiamo provato con altri due PIR, osservando però lo stesso comportamento;

- di difetti nel collegamento fra il PIR e il TelosB. Analizzando con un tester i segnali di output e di alimentazione del sensore, ci siamo accorti che, se le saldature del collegamento non erano fatte in modo accurato, potevano verificarsi degli occasionali sbalzi di tensione ai quali il PIR si mostrava molto sensibile. Abbiamo dunque verificato tutti i collegamenti cercando di renderli più stabili, osservando un buon miglioramento nel comportamento del sensore;

- di problemi all'alimentazione del PIR. Mentre svolgevamo le prove di cui abbiamo parlato al punto precedente, ci siamo accorti che il TelosB fornisce attraverso il modulo di espansione una tensione leggermente inferiore ai 3V dichiarati, e richiesti dal sensore per funzionare in modo corretto. Questo, in alcune occasioni, provocava il problema descritto inizialmente. Dunque abbiamo staccato il PIR e lo abbiamo provato alimentandolo separatamente, con tre batterie AA (dunque con una tensione complessiva di 4,5V): verificando l'output con il tester abbiamo visto che in questo modo i risultati erano finalmente corretti e stabili.

Una volta individuato il problema, abbiamo voluto trovare una soluzione alternativa alle batterie per far funzionare il PIR quando il TelosB è collegato via USB. Sarebbe stato possibile in ogni caso alimentare il sensore tramite le batterie, ma esiste anche una possibilità per far funzionare il tutto utilizzando solamente il collegamento USB. Infatti, una porta USB fornisce generalmente una tensione di 5V, che dunque sarebbe sufficiente ad alimentare il sensore;

poi è il TelosB, per le sue specifiche di alimentazione, ad abbassare questa tensione ad un valore attorno ai 3V, ma inizialmente il voltaggio necessario al PIR c'è, basta sapere dove andarlo a prendere. Così, dopo aver consultato i datasheet del TelosB, abbiamo realizzato un collegamento direttamente fra l'alimentazione proveniente dalla porta USB (a 5V) e il rilevatore di presenza, come si può vedere facendo ancora riferimento a figura 2.6.

Dopo aver apportato questa modifica, abbiamo verificato che il comportamento del sistema era affidabile. A questo punto abbiamo potuto posizionare i sensori ed iniziare a rilevare i dati di occupazione dei locali interessati. I rispettivi output venivano salvati sul PC al quale il TelosB era collegato sotto forma di file di testo, pronti per essere utilizzati nelle fase successive del progetto.

Capitolo 3

Il sistema di pattern recognition

In questo capitolo, cercheremo di delineare le basi teoriche da cui siamo partiti per progettare il funzionamento del nostro sistema. Nella sezione 3.1 introdurremo i concetti e le definizioni fondamentali nell'affrontare un problema di *machine learning*. In sezione 3.2 vedremo nel dettaglio il metodo di *clustering* (classificazione) che abbiamo utilizzato per il nostro sistema. Infine, in sezione 3.3, spiegheremo come abbiamo applicato i concetti visti per costruire un sistema che fosse stabile e dalle buone prestazioni.

3.1 Concetti preliminari sul machine learning

Il sistema che vogliamo progettare si può inquadrare nel campo più generale del *machine learning*, che ora andremo brevemente ad introdurre. Il machine learning è un campo dell'intelligenza artificiale che si prefigge di sviluppare sistemi ed algoritmi che consentano ai computer di affinare la propria base di conoscenza utilizzando informazioni sperimentali, allo scopo di migliorare

la propria capacità di prendere decisioni. Si tratta di un concetto molto generale, che poi può venire declinato in varie forme a seconda del tipo di dati utilizzati, dei risultati che si vogliono ottenere e dall'approccio che si intende seguire.

Una forma di machine learning particolarmente interessante è quella relativa al *pattern recognition* (riconoscimento di sequenze); si tratta di un problema molto comune, che può essere genericamente formulato come il tentativo di individuare, all'interno di un insieme di dati grezzi, delle sequenze particolari. La difficoltà del problema sta nel fatto che i pattern da identificare sono definiti in maniera probabilistica, o talvolta non sono definiti affatto (in tal caso, l'obiettivo può essere quello di individuare sequenze ricorrenti all'interno dei dati); questo rende il pattern recognition un problema molto diverso, ad esempio, da quello della ricerca di una parola in un testo.

Per introdurre il problema può essere utile un esempio, tratto da [1]. Si immagini di voler realizzare un sistema che sia in grado di riconoscere delle cifre scritte a mano, a partire da un immagine digitalizzata (che può provenire, ad esempio, dalla lettura di un macchinario per lo smistamento della posta). Un approccio tradizionale a questo problema potrebbe essere quello di programmare il sistema inserendo delle *regole*. Si dovrebbe quindi "insegnare" al sistema come è scritta ciascuna cifra, prevedendo le possibili differenze di calligrafia; sarebbe poi sicuramente necessario definire delle eccezioni a tali regole per gestire situazioni particolari; e così via, in un crescendo di regole ed eccezioni alle regole, da programmare manualmente, con una probabilità comunque bassa di ottenere risultati apprezzabili. Infatti, sarà comunque inevitabile che alcuni degli input proposti non rientrino in alcuna delle regole definite, e risultino così ingestibili per l'algoritmo.

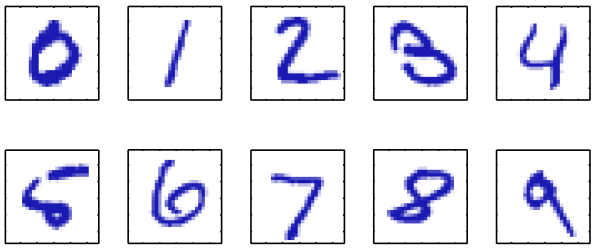

Figura 3.1: Esempio di un problema di pattern recognition: riconoscimento automatico di cifre scritte a mano (tratto da [1])

Al contrario, un approccio al problema di tipo machine learning prevede che sia la macchina stessa ad inferire le regole, "imparandole" in qualche modo a partire da dei dati forniti come esempio e, successivamente, "riconoscendo" le sequenze tramite identificazione con gli esempi proposti. Non si tratta certamente di un concetto innovativo, in linea generale: nessuno insegnerebbe ad un bambino a riconoscere il numero 9 spiegandogli come è fatto (un cerchietto in alto, dal lato destro del cerchio scende una gambetta, ...), è molto più facile farglielo vedere; rivoluzionario è il fatto di applicare questo tipo di approccio ad una macchina.

Seguendo questa intuizione, è possibile suddividere il funzionamento del sistema in due fasi. Nella prima fase, detta *learning phase* (fase di apprendimento), si fornirà al sistema un insieme di dati, detto insieme di apprendimento, formato da diverse coppie $\{(\boldsymbol{x}_1, \boldsymbol{t}_1), \ldots, (\boldsymbol{x}_N, \boldsymbol{t}_N)\}$; ciascuna coppia $(\boldsymbol{x}_i, \boldsymbol{t}_i)$ sarà formata da una possibile sequenza di input $\boldsymbol{x}_i$ e dall'output desiderato per quella sequenza $\boldsymbol{t}_i$. Nell'esempio delle cifre, l'insieme di apprendimento potrebbe essere rappresentato da diversi esempi di cifre scritte a mano, nei quali si fornisce però già al sistema l'etichetta giusta per ciascuna cifra. Questo tipo di approccio, in cui l'insieme di apprendimento è formato da coppie (input, risultato desiderato) è detto *supervised learning* (ad apprendimento supervisionato). L'insieme di apprendimento permetterà

al sistema di costruire ed adattare un modello adatto a descrivere i dati, sotto forma di una funzione $\boldsymbol{t} = \boldsymbol{y}(\boldsymbol{x})$. Il tipo di funzione è caratteristico del tipo di approccio seguito, ed è quindi già definito al momento della creazione del sistema, ma le associazioni fra le istanze $\boldsymbol{x}$ e le risposte $\boldsymbol{t}$ osservate durante la fase di learning servono ad adattare i parametri della funzione, definendone la forma esatta.

Nella seconda fase, detta *working phase* (fase di lavoro), la funzione $\boldsymbol{y}(\boldsymbol{x})$ modellata durante la prima fase verrà utilizzata per cercare di identificare sequenze di input $\boldsymbol{x}^*$ che non erano comprese nell'insieme di apprendimento iniziale: questa abilità è nota come generalizzazione. La capacità di generalizzazione di un algoritmo di pattern recognition è fondamentale, poiché solitamente l'insieme di apprendimento rappresenta solamente una piccola porzione di tutte le possibili sequenze di input.

Una forma diversa di pattern recognition è quella detta *unsupervised learning* (ad apprendimento non supervisionato). In questo caso, l'insieme di apprendimento è formato solamente dalle sequenze di input $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$; il meccanismo a due fasi (learning phase e working phase) rimane invariato, ma l'obiettivo può stavolta essere differente: per esempio, quello di individuare e raggruppare fra loro sequenze con delle caratteristiche in comune (problema del *clustering*).

Questo è esattamente il tipo di problema che dovremo affrontare con il nostro sistema: durante la fase di apprendimento, infatti, non forniremo nessuna indicazione sulle caratteristiche delle sequenze di ingresso. Non specificheremo modelli di possibili sequenze di ingresso da riconoscere, e non daremo nemmeno alcuna informazione riguardo le ore del giorno, i giorni della settimana, o in generale sul comportamento atteso per l'occupazione del locale. L'insieme di apprendimento sarà costituito solamente dalla se-

quenza rilevata dal sensore di presenza, e l'obiettivo del sistema sarà quello di raggruppare fra di loro le sequenze omogenee. Nella prossima sezione, approfondiremo l'argomento del clustering, sottolineando i motivi che ci hanno spinto ad utilizzare questo approccio per il nostro sistema.

3.2 Il metodo del clustering per fare pattern recognition

Genericamente, si può parlare di classificazione (o *clustering*) per riferirsi a tutte quelle attività che implicano il suddividere un insieme di oggetti in diversi gruppi, facendo in modo che i membri di ciascun gruppo siano *simili* tra di loro. A loro volta, i suddetti gruppi possono essere ulteriormente suddivisi in sotto-gruppi, dando luogo ad una classificazione gerarchica (*hierarchical clustering*). Fin dalla definizione si intuisce un elemento fondamentale per ogni classificazione: per poter propriamente stabilire se due oggetti sono simili tra loro, sarà necessario definire una funzione di distanza appropriata; i concetti generali relativi al clustering, tuttavia, prescindono dalla particolare funzione utilizzata.

Il processo di clustering produce tre effetti molto utili ai fini di un meccanismo di pattern recognition. Per prima cosa, la suddivisione in gruppi permette una compressione dei dati, sebbene soggetta ad una perdita di informazione. In [2] si porta un esempio di come sia possibile comprimere un'immagine utilizzando un processo di clustering: per prima cosa si crea un dizionario di k figure elementari e si suddivide la figura originale in piccole aree, identificando ciascuna di esse con l'immagine a distanza minore fra quelle all'interno del dizionario. Si sta di fatto operando una classificazione, suddividendo tutte le aree della figura originale in k gruppi i cui

"capostipiti" sono le k figure del dizionario. A questo punto è anche possibile riferirsi a ciascuna delle aree della figura originale semplicemente con l'indice del gruppo cui essa appartiene, e di cui si ha un modello nel relativo capostipite. Un meccanismo di questo tipo sarà utile nell'ambito in cui andremo ad operare, in cui avremo a che fare con sequenze molto lunghe di dati: sarà utile raggruppare sottosequenze simili tra loro poiché, in questo modo, non dovremo ricercare e memorizzare le proprietà di ogni singola sottosequenza, ma solamente di gruppi di sottosequenze.

Inoltre, ed è questo il motivo più importante per cui abbiamo scelto questo approccio, una classificazione ben fatta può essere utile nel fare previsioni. Generalmente, il clustering viene effettuato sulla base di una o più caratteristiche degli oggetti presi in considerazione; tali caratteristiche, evidentemente, devono essere necessariamente note al momento della classificazione. Tuttavia, una volta che avremo suddiviso in gruppi gli oggetti, potremo sperare che essi abbiano in comune anche altre caratteristiche "nascoste", oltre a quelle utilizzate per la classificazione: perché ciò avvenga, è necessario che esista un certo grado di correlazione fra il valore delle caratteristiche note e il valore di quelle nascoste. In particolare, nel nostro sistema, noi raggrupperemo le varie sottosequenze in base alla loro distanza, grazie ai dati raccolti durante la learning phase. Durante la working phase, identificheremo la sequenza corrente in uno dei cluster e utilizzeremo le statistiche dei membri del cluster per fare una previsione sull'evoluzione *futura* della sequenza corrente. In altre parole, faremo la classificazione in base ad una caratteristica nota (il valore corrente della sequenza), ma poi estrarremo un'informazione (relativa all'evoluzione futura della sequenza) che non è entrata in gioco nella classificazione: facendo questo, stiamo implicitamente assumendo che esista una certa correlazione fra il valore corrente della sequenza e la sua evoluzione

futura. Ci preoccuperemo più avanti di verificare questa assunzione (sebbene intuitivamente sembri sensata), così come di definire precisamente cosa intendiamo per *valore corrente* ed *evoluzione futura* di una sottosequenza.

Possiamo, infine, sottolineare una terza proprietà interessante di un processo di classificazione: una volta definiti e consolidati i cluster, i nuovi elementi che si discostino in maniera molto evidente da tutti i gruppi esistenti rappresentano probabilmente un fenomeno nuovo, sul quale è bene porre attenzione. Facendo riferimento alla nostra applicazione, si immagini di stare rilevando la sequenza di occupazione di un ufficio, con la sua naturale alternanza di presenze nelle ore diurne e non-presenze nelle ore notturne e nei giorni festivi. Dopo un certo periodo di tempo trascorso nella learning phase, i cluster saranno consolidati ed adatti alla situazione corrente. Se all'improvviso il sistema venisse spostato all'interno di un'abitazione, che generalmente presenta una sequenza di occupazione diametralmente opposta, chiaramente le sottosequenze rilevate non potrebbero corrispondere a nessuno dei cluster finora stabiliti: sarebbe dunque possibile rilevare questa anomalia e trattare appropriatamente la sequenza in oggetto (ad esempio, rifiutandosi di prevederne l'evoluzione). Deve essere tuttavia chiaro che il modello che utilizzeremo dovrà essere in grado di adattarsi ai cambiamenti dell'ambiente che sta rilevando; per tornare all'esempio, se il sensore venisse a quel punto lasciato nell'abitazione, con il passare del tempo le sequenze rilevate risulterebbero sempre meno "anomale", come conseguenza dell'adattamento del sistema alla situazione ambientale.

Esiste una vasta bibliografia per quanto riguarda i possibili algoritmi che si possono utilizzare per il clustering: a tal proposito si può vedere [3]. Per il nostro sistema, abbiamo deciso di utilizzare quello che è probabilmente l'algoritmo di classificazione più diffuso, noto come *k-means clustering*.

Vedremo i dettagli di questo algoritmo nella prossima sottosezione.

3.2.1 L'algoritmo k-means

L'algoritmo k-means consente di suddividere N punti di uno spazio I-dimensionale in k gruppi, ognuno dei quali è rappresentato da un vettore m_i che costituisce la media dei punti appartenenti a quel gruppo. La caratteristica fondamentale dell'algoritmo è quella di assegnare ciascun punto al gruppo la cui media ha distanza minore, allo scopo di ottenere una partizione $S = \{S_1, S_2, \ldots, S_k\}$ che possa minimizzare la somma dei quadrati delle distanze all'interno di ciascun cluster:

$$\underset{S}{\operatorname{argmin}} \sum_{i=1}^{k} \sum_{x_j \in S_i} ||x_j - m_i||^2 \tag{3.1}$$

L'algoritmo nella versione presentata qui risale agli anni '50, quando venne proposto da Stuart Lloyd, che tuttavia lo pubblicò solamente nel 1982, in [4]. Il nome dell'algoritmo, invece, venne usato per la prima volta da James MacQueen nel 1967 in [5]. In quasi tutte le varianti, comunque, k-means è un algoritmo iterativo, che si compone di due passi fondamentali: passo di **assegnazione** (*assignment step*) e passo di **aggiornamento** (*update step*). Nel listato 1 si può trovare l'algoritmo nelle sue linee fondamentali.

Per prima cosa, è necessario inizializzare in qualche modo le medie di ciascun cluster: nella parte relativa ai dettagli dell'implementazione, vedremo due possibili modi per farlo. A questo punto, si ripetono iterativamente i due passi a cui abbiamo appena accennato. Nel passo di assegnazione, ognuno degli N punti viene associato al cluster i la cui media m_i è quella a distanza minore dal punto considerato; quindi, nel passo di aggiornamento, le medie di ciascun gruppo vengono ricalcolate, prendendo in considerazione tutti e

Algorithm 1 Pseudo-codice per l'algoritmo k-means

1: {**Initialization**}
2: Set the means to arbitrary values $m_1^{(1)} \dots m_k^{(1)}$
3: $stable \leftarrow 0$
4: **while** $stable = 0$ **do**
5: {**Assignment step**}
6: $S_i^{(t)} \leftarrow \{x_j : ||x_j - m_i^{(t)}|| \leq ||x_j - m_{i^*}^{(t)}|| \ \forall \ i^* = 1, \dots, k\}$
7: **if** $S_i^{(t)} = S_i^{(t-1)}$ **then**
8: $stable \leftarrow 1$
9: **end if**
10: {**Update step**}
11: $m_i^{(t+1)} \leftarrow \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j$
12: **end while**

soli i punti che ora appartengono al gruppo in oggetto. A questo punto si ripetono sequenzialmente i due passi, fino al punto in cui il passo di assegnazione non modifica nessuna delle appartenenze dei punti al cluster (e, di conseguenza, le medie rimangono invariate). In questa descrizione dell'algoritmo abbiamo volutamente tralasciato alcuni dettagli che ricoprono una certa importanza, come il metodo per inizializzare l'algoritmo e la particolare funzione di distanza da utilizzare: tratteremo di questi argomenti nei capitoli successivi.

L'algoritmo, nella sua semplicità concettuale, presenta diversi punti degni di nota. Un primo fondamentale pregio è dato dalla sua semplicità computazionale: sebbene nel *worst-case* l'algoritmo possa presentare una complessità esponenziale, nella pratica questa condizione non si verifica quasi mai, e la smoothed complexity[1] dell'algoritmo è polinomiale nel numero di punti considerati. Inoltre, sebbene l'algoritmo possa considerarsi euristico (questo per via della fase di inizializzazione, la quale conterrà un elemento di casualità), si può dimostrare che esso converge sempre ad una soluzione.

[1] La smoothed complexity è un metodo alternativo a worst-case e average-case per l'analisi della complessità di un algoritmo; essa si ottiene considerando leggere perturbazioni casuali rispetto agli input del worst-case.

D'altro canto, k-means mostra anche alcuni punti deboli, con i quali dovremo necessariamente fare i conti in fase di implementazione del sistema. Per prima cosa, osserviamo che per il funzionamento dell'algoritmo è necessario definire a priori il numero di gruppi in cui si vuole partizionare l'insieme. Questo non è da considerarsi un difetto tout-court, perché generalmente questa categoria di algoritmi di clustering permette di ottenere risultati migliori, qualora il numero di gruppi sia effettivamente noto a priori; tuttavia, non essendo questo il nostro caso, il dover scegliere un valore fisso per k pone un problema che dovremo necessariamente risolvere in fase di implementazione, anche perché si può osservare sperimentalmente che la scelta di un valore di k non adatto alla distribuzione dei dati porti rapidamente a un decadimento delle prestazioni. Un secondo problema di k-means è dato dal fatto che i risultati che si ottengono dipendono pesantemente dalla fase di inizializzazione dell'algoritmo: è possibile che scelte iniziali "sfortunate" nella determinazione dei centri dei cluster non riescano più a venir corrette nel corso dell'esecuzione, trascinando l'algoritmo ad individuare classificazioni poco significative. Si può dimostrare che k-means raggiunge un ottimo locale per il problema presentato, il quale tuttavia può discostarsi significativamente dall'ottimo globale. Questo problema viene spesso affrontato confidando nella velocità di esecuzione di k-means: si esegue l'algoritmo più volte sullo stesso set di dati utilizzando diverse inizializzazioni, per poi confrontare le diverse soluzioni e tenere quella con la classificazione più promettente; è chiaro che anche *come* individuare la classificazione migliore sia uno dei problemi da affrontare in corso di implementazione.

3.3 La nostra idea

Ora che abbiamo definito tutti gli strumenti necessari, possiamo delineare nel suo complesso la strategia che intendiamo seguire per realizzare il nostro sistema di pattern recognition. Prima, però, soffermiamoci per definire formalmente la notazione che utilizzeremo.

Definizione 1. *Sequenza temporale*: definiamo la sequenza temporale $T = t_1, \ldots, t_m$ come un insieme ordinato di m variabili reali.

Definizione 2. *Sottosequenza*: data una sequenza temporale T di lunghezza m, una sottosequenza C_p di T è data dall'estrazione di $n < m$ posizioni contigue di T. In altre parole, $C_p = t_p, \ldots, t_{p+n-1}$ per $1 \leq p \leq m - n + 1$. Si noti che la sequenza T possiede $m - n + 1$ sottosequenze di questo tipo.

Durante la learning phase, dunque, considereremo delle sottosequenze dell'intera sequenza temporale T. Per ora non ci soffermiamo su *quali* sottosequenze considerare, dal momento che, come vedremo nella sottosezione 3.3.1, questo sarà un problema cruciale che è bene affrontare a parte. Suddivideremo quindi l'insieme delle sottosequenze in diversi gruppi, operando una classificazione mediante l'algoritmo k-means. Il nostro obiettivo in questa fase sarà quello di raggruppare negli stessi gruppi le sottosequenze più simili fra loro, confrontando gli m campioni di cui sono composte e la loro posizione relativa, come vedremo in dettaglio nella sottosezione 3.3.2. Per ogni sottosequenza potremo salvare anche altre informazioni, le quali tuttavia non saranno utilizzate nella classificazione; ad esempio, sarà particolarmente utile memorizzare i campioni che seguono quella determinata sequenza nella serie temporale T. In questo modo, potremo ottenere una descrizione statistica dell'evoluzione futura per quanto riguarda le sottosequenze appartenenti a quel determinato cluster.

Durante la working phase, considereremo di volta in volta una sottosequenza $\boldsymbol{c}$ formata dagli ultimi n campioni rilevati dal sensore. La nostra idea è quella di identificare la sottosequenza corrente in uno dei cluster che abbiamo costruito durante la learning phase, fornendo una prima modellizzazione della sottosequenza. Una volta che avremo identificato $\boldsymbol{c}$ all'interno di un gruppo, potremo utilizzare le informazioni aggregate relative al gruppo per fare delle previsioni su caratteristiche della sottosequenza che, al momento, non sono note: nel nostro caso, utilizzeremo le informazioni relative all'evoluzione futura delle varie sottosequenze del gruppo, per fare delle previsioni sull'evoluzione della sottosequenza corrente. Infine, si aggiorneranno tutte le strutture dati con le informazioni portate dalla nuova sottosequenza: in particolare, si dovranno aggiornare le medie del k-means (con il conseguente riposizionamento dei cluster che ne potrebbe conseguire) e tutte le informazioni che siano state eventualmente memorizzate relativamente al gruppo. Infatti, a questo punto, la sottosequenza $\boldsymbol{c}$ sarà parte a tutti gli effetti del gruppo, e quindi contribuirà come gli altri membri a caratterizzarlo; nel nostro caso, l'evoluzione futura di $\boldsymbol{c}$ andrà a mediare con tutte le altre già note per stabilire l'evoluzione futura attesa per quel determinato cluster.

L'idea alla base del sistema che vogliamo progettare è tutto sommato semplice, sebbene restino da definire molti particolari fondamentali, dei quali ci occuperemo nei prossimi capitoli. Nelle sottosezioni successive, ci occuperemo di definire quali sottosequenze utilizzare per il clustering, e di quale funzione di distanza sia più appropriato utilizzare.

3.3.1 Come utilizzare k-means con sottosequenze temporali?

Sebbene l'idea alla base del sistema fosse chiara, è subito sorto un problema interessante riguardante l'implementazione dell'algoritmo di classificazione.

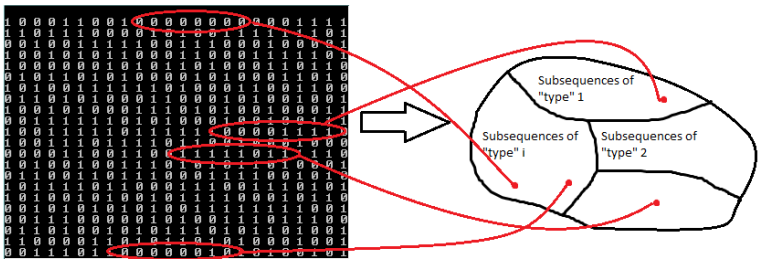

Figura 3.2: Learning phase

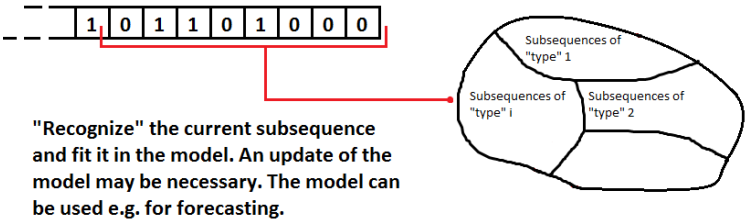

Figura 3.3: Working phase

Ricordiamo che, per quanto riguarda la learning phase, il solo input del sistema è costituito dagli m campioni dell'unica sequenza temporale T, che rappresentano le letture effettuate dal sensore di movimento durante l'intera fase. Per poter operare la classificazione, è necessario prima individuare delle sottosequenze della sequenza T che poi potremo utilizzare come input dell'algoritmo k-means, e ci siamo chiesti quale fosse il metodo migliore per selezionare queste sottosequenze.

Inizialmente, la nostra idea per procedere era abbastanza intuitiva: intendevamo estrarre tutte le $m-n+1$ sottosequenze di T mediante l'utilizzo di una sliding window. Questo metodo è noto in letteratura come *Subsequence Time Series (STS) Clustering* e, almeno fino al 2004, è stato estensivamente usato come base per decine di pubblicazioni: fra queste, possiamo segnalare un importante lavoro di Das [6], nel quale si ricercano delle sequenze ricorrenti all'interno della serie dei prezzi del NASDAQ. In figura 3.4 possiamo vedere un'illustrazione di questo procedimento, tratta da [7], al quale si può

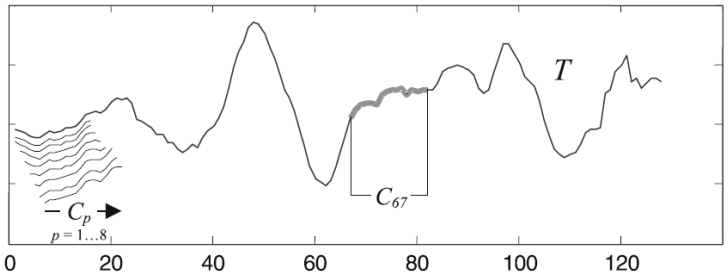

Figura 3.4: Estrazione delle sottosequenze mediante sliding window (tratto da [7])

anche fare riferimento per un elenco dettagliato delle pubblicazioni che fanno uso di STS Clustering. Incoraggiati dal lavoro di Das, che sembrava produrre risultati concreti, e dalla notevole bibliografia al riguardo, la nostra idea di utilizzare tutte le sottosequenze ne usciva rafforzata. Sfortunatamente, si sarebbe rivelata essere completamente sbagliata.

Nel 2004, infatti, Keogh e Lin pubblicarono un lavoro dal titolo eloquente: *Clustering of time-series subsequences is meaningless: implications for previous and future research* [7]. In esso viene dimostrato inequivocabilmente, mediante prove sperimentali e dimostrazioni formali, che operare una classificazione di tutte le sottosequenze prelevate da una sequenza temporale, porta invariabilmente a risultati privi di significato; dove con "privo di significato" si intende che l'output prodotto da tali algoritmi è indipendente dalla sequenza di input considerata. In una delle prove, gli autori considerarono i cluster prodotti eseguendo diverse volte l'algoritmo sullo stesso dataset (i prezzi di chiusura dell'indice S&P500), e quelli ottenuti utilizzando un dataset completamente diverso (una serie *random walk*); i risultati mostrarono che, incredibilmente, la distanza media fra i cluster ottenuti da diverse esecuzioni con lo stesso dataset era dello stesso ordine di grandezza rispetto alla distanza fra i cluster ottenuti dai due dataset diversi.

È utile soffermarsi brevemente sulle cause che portano a questi risultati, allo scopo di considerare un procedimento che possa essere esente da questi difetti. Per prima cosa, si immagini di voler ricercare la media di tutte le sottosequenze della sequenza temporale T: ciò equivale, di fatto, ad effettuare una classificazione con $k = 1$ e a ricercare il centro di questo cluster. È chiaro che ogni campione v_i, $1 \leq i \leq n$ del vettore media sarà ottenuto considerando essenzialmente *tutti* i campioni della serie temporale (precisamente, tutti i campioni da t_i a t_{m-n+i}). Per $m \gg n$, dunque, i campioni all'inizio e alla fine di T diventato asintoticamente irrilevanti, e si ha che per il calcolo di tutti gli elementi del vettore media vengono considerati essenzialmente gli stessi campioni: ciò fa sì che il vettore media non possa essere altro che una sequenza di elementi uguali. Questa è una condizione che dovrà essere sempre verificata ed ha un impatto catastrofico sul nostro algoritmo di classificazione; infatti, dobbiamo osservare che la media di tutte le sottosequenze corrisponde alla media pesata (per il numero di elementi appartenenti a ciascun cluster) dei centri dei cluster che verranno individuati. Dunque stiamo vincolando i centri dei cluster ad avere questa particolare proprietà: se i pattern che dobbiamo trovare *non* hanno questa caratteristica (cioè la loro media pesata non è una sequenza costante) allora, semplicemente, essi non verranno individuati.

Non è tutto: l'efficacia del STS Clustering è minata anche dall'importanza del fenomeno dei *trivial matches*. Diamo alcune ulteriori definizioni:

Definizione 3. *Match*: dato un raggio R e due sottosequenze C_p e C_q, dove p e q sono le posizioni del primo elemento della sottosequenza, diremo che C_q è un *match* per C_p se $dist(C_p, C_q) \leq R$.

Definizione 4. *Trivial matches*: dato un raggio R e due sottosequenze C_p e C_q, dove p e q sono le posizioni del primo elemento della sottosequenza,

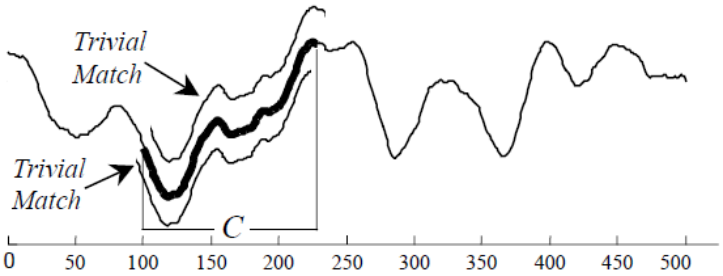

Figura 3.5: Esempio di trivial match (tratto da [8])

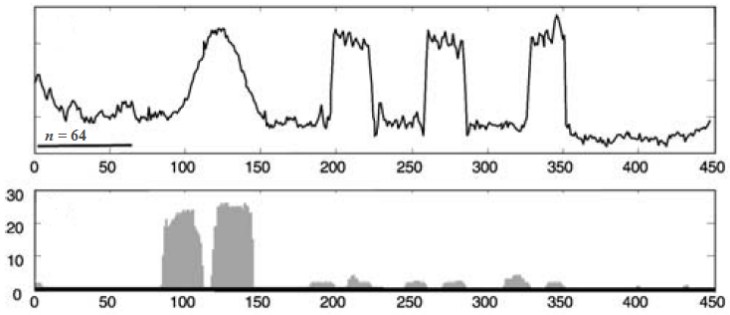

Figura 3.6: Importanza dei trivial match nel clustering (tratto da [7])

se C_q è match per C_p, diremo che è un *trivial match* se $p = q$, oppure se non esiste nessuna sottosequenza C_r, con $p < r < q$ o $q < r < p$, tale che $dist(C_p, C_r) > R$.

In figura 3.5 si può vedere un esempio di trivial match, tratto da [8]. Sostanzialmente, la definizione esprime il concetto che è molto probabile che i migliori match di una certa sottosequenza C_p si trovino nelle immediate vicinanze di essa, sotto forma di una versione leggermente shiftata della stessa. Come possono incidere i trivial match nell'algoritmo di clustering? L'esempio di figura 3.6, tratto da [7], è illuminante: nella figura superiore si può vedere una sequenza temporale caratterizzata da una forma gaussiana e da tre impulsi rettangolari, i quali sembrerebbe che possano formare un cluster. Nel grafico sottostante, ogni elemento i rappresenta il numero di trivial mat-

ches della sottosequenza C_i (considerando $n = 64$, $R = 1$): si può osservare come la presenza di sottosequenze che variano lentamente (come la parte ascendente e discendente della gaussiana) provochi un'esplosione del numero di trivial matches, mentre sottosequenze anche ben definite come i tre impulsi rettangolari ne contino un numero decisamente minore. La conseguenza è che, se si provasse ad eseguire un qualsiasi algoritmo di classificazione su questo insieme di sottosequenze, i centri dei vari cluster sarebbero irrimediabilmente attratti dalla zona densa di matches, finendo per assomigliare alle due semplici linee (ascendente e discendente) che compongono la gaussiana. Più in generale si può concludere che tutte le sottosequenze, anche ricorrenti, caratterizzate da transizioni nette, avrebbero una forte probabilità di essere ignorate dall'algoritmo di classificazione, a favore di altre sottosequenze dagli andamenti più regolari.

Quest'ultima osservazione mette una pietra tombale sulla reale applicabilità del STS clustering con tutte le sottosequenze: non solo i pattern da individuare dovrebbero avere una media pesata pari ad una costante, ma dovrebbero anche avere, fissato un raggio R, lo stesso numero di trivial matches, per essere trattati equamente dall'algoritmo. È altamente inverosimile che entrambe queste condizioni possano verificarsi, ragion per cui in [7] viene proposto un metodo alternativo per estrarre solo alcune sottosequenze dalla serie temporale T, evitando il verificarsi di questi problemi. A questo scopo, viene introdotto il concetto di *motif*, di cui ora daremo la definizione.

Definizione 5. *1-motif*: data una sequenza temporale T e un raggio R, allora il motif più ricorrente in T, detto *1-motif*, è la sottosequenza C_1 di lunghezza n che possiede il più alto numero di non-trivial matches.

Definizione 6. *K-motif*: data una sequenza temporale T e un raggio R, allora il K-esimo motif più ricorrente in T, detto *K-motif*, è la sottosequenza

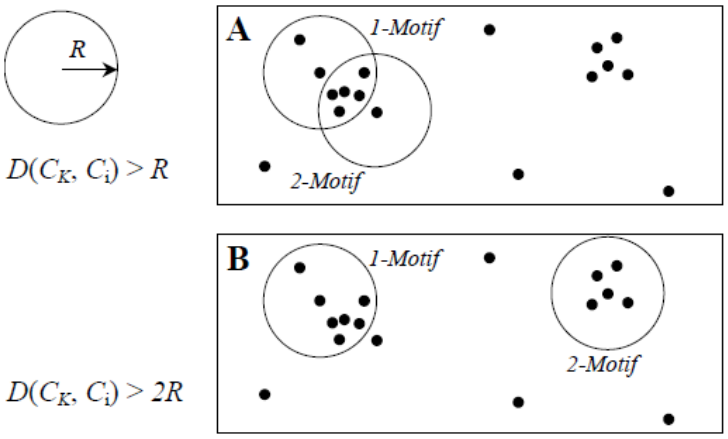

Figura 3.7: Spiegazione del motivo per cui un motif C_i deve avere distanza $> 2R$ (e non solo $> R$) dai precedenti (tratto da [8]).

C_K di lunghezza n che possiede il più alto numero di non-trivial matches e soddisfa $dist(C_K, C_j) > 2R \; \forall \; 1 \leq j < K$.

Si osservi che la definizione implica che l'appartenenza di una sottosequenza ad un motif sia mutuamente esclusiva, impedendo che due motif possano condividere gran parte dei loro elementi: questo concetto è ben illustrato in figura 3.7.

Il concetto di motif è simile a quello di cluster, ma si differenzia da esso per due motivi fondamentali. Per prima cosa, i motif finiscono per comprendere una piccola porzione dell'intero insieme delle sottosequenze; inoltre, nella definizione di motif sono esplicitamente esclusi i trivial matches. Si noti che questi sono proprio i due fattori che, per i motivi esposti sopra, rendono priva di significato una classificazione basata sull'intero insieme delle sottosequenze.

A questo punto, la strategia per effettuare il clustering delle sottosequenze potrebbe articolarsi come segue: per prima cosa, si trovano nell'intero insieme delle sottosequenze i K-motif più significativi, con K scelto in modo

che $K >> k$. A questo punto si può procedere normalmente con l'algoritmo di classificazione (k-means, nel nostro caso), per suddividere i K motif in k cluster omogenei.

3.3.2 La funzione di distanza

Un altro punto importante che finora non è stato ancora affrontato riguarda la funzione di distanza che utilizzeremo per confrontare fra di loro le sottosequenze temporali estratte dalla serie T.

Bisogna innanzitutto puntualizzare che l'algoritmo k-means richiede, per funzionare correttamente, che la funzione di distanza sia di tipo euclideo. Questa di per sé non è una limitazione, infatti la distanza classica di tipo euclideo è quella largamente più utilizzata nel campo del pattern recognition. Generalmente, per confrontare due sequenze $\boldsymbol{c} = \{c_1, \ldots, c_N\}$ e $\boldsymbol{d} = \{d_1, \ldots, d_N\}$ composte da N campioni si utilizza una misura euclidea N-dimensionale:

$$dist(\boldsymbol{c}, \boldsymbol{d}) = \sqrt{(c_1 - d_1)^2 + \ldots + (c_N - d_N)^2} \tag{3.2}$$

Tuttavia, questo tipo di funzione potrebbe non rivelarsi adatto per i nostri scopi: dobbiamo prima fermarci a riflettere con attenzione su quali potrebbero essere, nello specifico della nostra applicazione, due sequenze *simili*. Dobbiamo osservare per prima cosa che, nel nostro caso, due sequenze perfettamente identiche potrebbero avere significati molto diversi, a seconda della loro posizione; possiamo visualizzare la situazione mediante l'esempio di figura 3.8: in essa sono raffigurate due sequenze $\boldsymbol{c}$ e $\boldsymbol{d}$ composte esattamente dalla stessa sequenza di campioni, ma che iniziano in due istanti diversi (t_c e t_d) della serie temporale T. È chiaro che in qualche modo la differenza

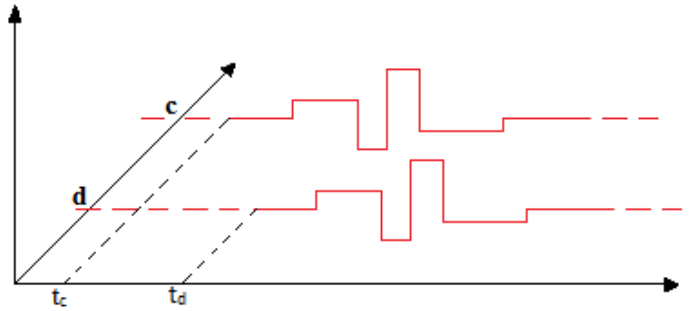

Figura 3.8: Scelta di una funzione di distanza appropriata: sequenze uguali ma che iniziano in momenti diversi devono avere distanza > 0

di posizione fra le due sequenze considerate dovrà entrare nel calcolo della distanza.

Un'altra differenza sostanziale rispetto alla maggior parte dei problemi di pattern recognition sta nel fatto che, nel nostro problema, la base temporale ha una struttura periodica: riteniamo che due sequenze che si verifichino alla stessa ora in giorni diversi (quindi in posizioni diverse della sequenza temporale) debbano essere necessariamente associate come simili. Di conseguenza, dovremo fare in modo che la differenza di posizione non sia valutata in termini assoluti ma con riferimento alla durata di un periodo, come mostrato in figura 3.9.

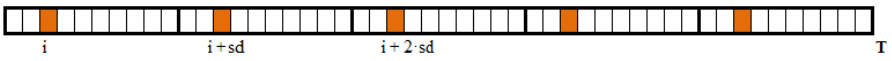

Figura 3.9: Periodicità della sequenza temporale

Non abbiamo trovato in letteratura nessun esempio di funzione di distanza che potesse adattarsi a questa particolare situazione, così abbiamo pensato di generalizzare la normale misura euclidea, considerandola però in

$N + 1$ dimensioni:

$$dist(\boldsymbol{c}, \boldsymbol{d}) = \sqrt{(c_1 - d_1)^2 + \ldots + (c_N - d_N)^2 + (o_c - o_d)^2} \qquad (3.3)$$

dove $o_c = t_c \bmod sd$, $o_d = t_d \bmod sd$ scrivendo con sd il periodo considerato (quindi, nel nostro caso, il numero di campioni in un giorno). In questo modo due sottosequenze identiche, che si verificano esattamente alla stessa ora in giorni diversi, avranno distanza pari a 0.

Capitolo 4

Implementazione software del sistema

In questo capitolo ci occuperemo di dare una descrizione dettagliata delle scelte operate per l'implementazione vera e propria dell'algoritmo, per dare seguito alle idee presentate nel capitolo 3. Per la scrittura del codice è stato utilizzato *MATLAB*, per la grande potenza espressiva e per la semplicità che offre nel maneggiare strutture dati complesse. Il codice nel suo complesso si può trovare in Appendice, mentre in questo capitolo esporremo estratti delle parti più significative, facendo uso di pseudo-codice dove possibile per non rendere la trattazione vincolata ad uno specifico linguaggio di programmazione.

4.1 Le strutture dati

L'algoritmo che abbiamo delineato nel capitolo precedente non specifica quali valori assegnare ad alcuni parametri fondamentali, che andranno ad incidere pesantemente sul funzionamento del sistema. Fra i più importanti, vi sono

la lunghezza delle sottosequenze, il numero di motif e di cluster, il raggio, la lunghezza del forecast; non era facile prevedere quali sarebbero potuti essere dei valori appropriati per questi parametri quando si è iniziato a progettare il sistema. Si è quindi fatto in modo che tutto il codice non funzionasse solamente con una particolare configurazione di parametri, ma che fosse poi possibile modificare questi valori per cercare il settaggio che garantiva le migliori performance.

Pensiamo che sia utile, a questo punto, per permettere di seguire agevolmente la trattazione, esporre le principali variabili e strutture dati che abbiamo utilizzato nel programma, che si possono trovare rispettivamente in tabella 4.1 e 4.2. Assieme alla descrizione dei parametri fondamentali, per fissare le idee abbiamo inserito anche i valori che abbiamo utilizzato per testare il sistema; qualora in corrispondenza di una variabile si trovino più valori, significa che abbiamo eseguito diverse prove per valutare quale fosse quello più adatto da assegnare al parametro.

4.2 Importazione dei dati

Il primo passo che dovrà compiere il sistema di elaborazione dei dati sarà quello di leggere le sequenze di input per memorizzarle nelle matrici $\boldsymbol{T}$ e $\boldsymbol{T1}$, che saranno utilizzate per la learning phase e per la working phase rispettivamente. Utilizzeremo due diversi tipi di sequenze di input: uno corrispondente ai dati sperimentali, ottenuti mediante le letture del rilevatore di presenza collocato in un'abitazione; e uno ottenuto tramite un simulatore che abbiamo scritto appositamente.

Nel primo caso, ricordiamo che abbiamo programmato il TelosB per stampare a video il risultato delle letture; questo output veniva poi catturato dal sistema operativo dell'host cui il sensore era collegato e memorizzato in un

Nomi delle variabili e significato		Valori
n	Lunghezza delle sottosequenze	12,16,24,36
fl	Lunghezza del forecast	4
k	Numero di cluster del k-means	4,6,8,12
K	Numero di motif	70
R	Raggio per determinare un match	1,2,4,6
sd	Numero di campioni per giorno	96

Tabella 4.1: Parametri fondamentali e rispettivi valori

Nome	Dimensioni	Contenuto
T	2 × m	Nella prima riga contiene gli m campioni della sequenza di input della learning phase; nella seconda, la posizione relativa di quel campione all'interno del periodo
T1	2 × m1	Come per la matrice $\boldsymbol{T}$, ma relativamente alla working phase.
mot_mtx	K × n	Matrice contenente, nella i-esima riga, gli n campioni del motif i
off	K × 1	Vettore contenente, nella i-esima riga, la posizione relativa del motif i
nxt_mtx	K × fl	Matrice contenente, nella i-esima riga, gli fl campioni successivi al motif i nella serie temporale
loc	1 × K	Vettore contenente la posizione dei motif all'interno di $\boldsymbol{T}$
cnt	1 × K	Vettore contenente il numero di occorrenze dei motif
lst	1 × K	Vettore contenente il numero di campioni trascorso dall'ultima occorrenza dei motif
wgh	1 × K	Vettore contenente il peso dei motif
kmns_mtx	k × K	Definisce l'appartenenza dei motif a ciascun cluster: $kmns\_mtx(i,j) = 1$ se il motif j appartiene al cluster i, 0 altrimenti.
mns	k × n	Matrice contenente, nella i-esima riga, gli n campioni della media pesata dei motif appartenenti al cluster i
mns_off	k × 1	Vettore contenente, nella i-esima riga, la media pesata degli offset dei motif appartenenti al cluster i
fm_mtx	k × fl	Matrice contenente, nella i-esima riga, gli fl campioni della media pesata delle sequenze successive ai motif appartenenti al cluster i

Tabella 4.2: Strutture dati utilizzate nel codice

file di log, che sarà quello che utilizzeremo per importare i dati. I dati così letti riguardano 30 giorni di rilevazioni, che utilizzeremo sia per la learning phase (matrice **T**) che per la working phase (matrice **T1**). Questo punto non presenta particolarità degne di nota, si tratta semplicemente della lettura di un file di testo utilizzando gli strumenti messi a disposizione da MATLAB. L'unico aspetto che merita una puntualizzazione riguarda le tempistiche della working phase. Questa fase, infatti, durante il funzionamento vero e proprio del sistema dovrebbe avvenire online, in questa sequenza:

1. il sensore legge i dati e, quando disponibili, li scrive su una struttura adeguata;

2. l'elaboratore rimane in attesa di nuovi dati. non appena un nuovo campione viene scritto dal sensore, questo viene immediatamente elaborato per identificare la sottosequenza corrente ed eventualmente fornire una previsione.

Nelle nostre simulazioni di funzionamento, questo meccanismo a due fasi viene modificato e la working phase avviene offline: prima è stata letta e memorizzata l'intera sequenza di input, mentre successivamente si è proceduto all'elaborazione della stessa. Si è scelto questo approccio soprattutto per motivi pratici, in ogni caso dal punto di vista concettuale non c'è nessuna differenza fra i due metodi di funzionamento, e il sistema è adattabile con poche modifiche per poter funzionare online.

Nel secondo caso, invece, i dati in ingresso erano prodotti a tempo di esecuzione da un algoritmo che abbiamo progettato appositamente, chiamato *Office Simulator*. Come suggerisce il nome, il programma simula la sequenza di occupazione di un ufficio nell'arco delle 24 ore. Il programma è in grado di simulare diversi orari di arrivo e di partenza dall'ufficio (all'interno di

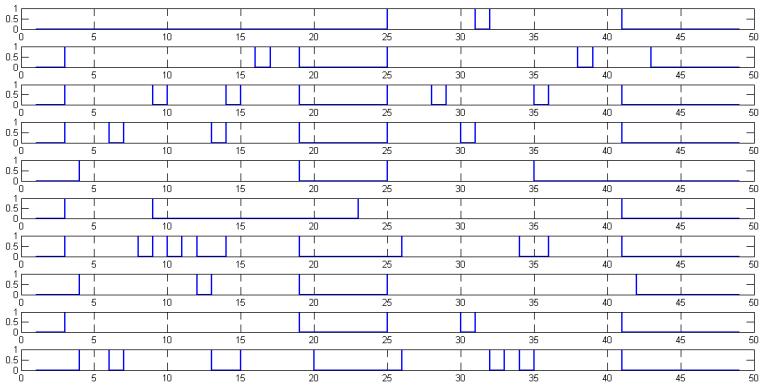

Figura 4.1: Esempio delle sequenze prodotte da Office Simulator

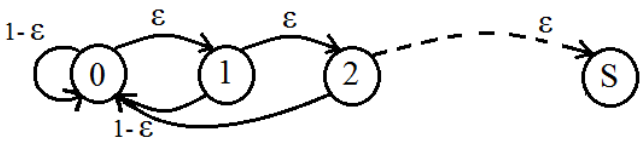

Figura 4.2: Catena di Markov utilizzata per simulare la presenza nell'ufficio

un range prestabilito), arrivi in ritardo (fuori dal range) e pause pranzo di diversa durata; inoltre simula assenze occasionali, per brevi periodi oppure per l'intera mattinata, l'intero pomeriggio o tutto il giorno. In figura 4.1 si può vedere un esempio dell'output prodotto dal simulatore.

Vogliamo soffermarci brevemente sul sistema che abbiamo utilizzato per simulare la permanenza nell'ufficio, all'interno degli orari stabiliti in maniera probabilistica. Abbiamo modellato questa situazione come una catena di Markov con S stati, come si può vedere in figura 4.2; nel nostro sistema abbiamo impostato $S = 3$. Finchè la catena si trova nello stato 0, l'ufficio risulta "occupato", e si permane in questo stato con probabilità $1 - \epsilon$; ϵ rappresenta la probabilità di uscire dallo stato 0 per portarsi nel primo degli stati che rappresenta l'assenza dall'ufficio. Una volta raggiunto lo stato i,

$0 < i < S$, si torna allo stato 0 con probabilità $1 - \epsilon$, mentre con probabilità ϵ si raggiunge lo stato $i + 1$. Lo stato S è uno stato assorbente per la catena di Markov, che sta a significare che l'occupante dell'ufficio se ne è andato definitivamente; quando si raggiunge questo stato, l'algoritmo fa sì che la sequenza mostri un'assenza da quel momento fino alla fine della mattinata (o della giornata intera).

L'utilizzo di un simulatore si è reso necessario in quanto i soli dati ricavati dal sensore non erano sufficienti a testare l'algoritmo in maniera estensiva: avevamo bisogno di una grande mole di dati per provare la solidità dell'algoritmo in diverse situazioni particolari, e per avere informazioni attendibili sulle performance dell'algoritmo, escludendo la possibilità che queste potessero essere provocate dalla particolare sequenza di input. Inoltre l'utilizzo del simulatore rende possibile l'utilizzo di sequenze diverse per la fase di learning e quella di funzionamento, permettendo di testare il sistema in condizioni simili a quelle reali.

4.3 La learning phase

4.3.1 La ricerca dei motif

Un primo punto fondamentale nell'implementazione dell'algoritmo è stato quello riguardante la ricerca dei motif nella serie temporale $\boldsymbol{T}$. Infatti, un problema che finora abbiamo tralasciato è quello riguardante la complessità computazionale di tale ricerca: nel listato 2 si può vedere un esempio di algoritmo *brute-force* per ricercare l'1-motif in una serie temporale. Esso tratta ogni possibile sottosequenza della sequenza temporale $\boldsymbol{T}$ come un possibile candidato per essere 1-motif; per ogni candidato, è necessario scorrere tutta la sequenza temporale per verificare quanti non-trivial matches può contare.

Algorithm 2 Pseudo-codice per l'algoritmo brute-force *Find1Motif*

```
1: best_count ← 0;
2: for i = 1 to length(T) − n + 1 do
3:    count ← 0;
4:    pointers ← null;
5:    for j = 1 to length(T) − n + 1 do
6:       if non_trivial_match(C_[i:i+n−1], C_[j:j+n−1], R) then
7:          count ← count + 1;
8:          pointers ←append(pointers, j);
9:       end if
10:   end for
11:   if count > best_count then
12:      best_count ← count;
13:      best_loc ← i;
14:      motif_matches ← pointers;
15:   end if
16: end for
```

In generale, si effettuano $O(m^2)$ chiamate alla funzione di distanza, quindi l'algoritmo ha complessità computazionale che cresce con il quadrato della lunghezza della serie temporale T. Per ricercare i motif successivi al primo, si ha una complessità via via decrescente per via del fatto che non tutte le sottosequenze saranno candidabili per essere motif: questo a causa della condizione che vuole che due motif debbano avere una distanza pari almeno a $2R$.

Un primo suggerimento per tagliare la complessità computazionale potrebbe essere quello di memorizzare le distanze già calcolate, e di sfruttare la proprietà di simmetria della distanza per evitare di calcolare $dist(B, A)$ quando sia già nota $dist(A, B)$. Questo consentirebbe di dimezzare il tempo di esecuzione, ma richiederebbe uno spazio di memorizzazione per $m(m-1)/2$ valori, che rende questa possibilità difficilmente attuabile.

In [8], gli autori propongono un algoritmo di complessità sub-quadratica per la risoluzione del problema. La soluzione proposta si basa sul concetto, molto semplice, di disuguaglianza triangolare. Supponiamo di trovarci nel

ciclo più interno dell'algoritmo brute-force del listato 2, mentre scorriamo tutte le sottosequenze cercando i possibili matches per la sottosequenza Q. Una volta calcolata la distanza $dist(Q, C_a) = d_{qa}$, se dai cicli precedenti era già nota la distanza $dist(C_a, C_b) = d_{ab}$, allora è possibile che queste due informazioni congiunte ci permettano di evitare il calcolo della distanza fra Q e C_b. Infatti, per la disuguaglianza triangolare:

$$D(Q, C_a) \leq D(Q, C_b) + D(C_a, C_b) \tag{4.1}$$

e quindi

$$D(Q, C_b) \geq D(Q, C_a) - D(C_a, C_b) \tag{4.2}$$

A questo punto, se il secondo membro di 4.2 è $> R$, possiamo avere la certezza che C_b non può essere un match per la sottosequenza Q. Il contributo rilevante di [8] è dato dal fatto che gli autori propongono un metodo per evitare di memorizzare tutti gli $O(m^2)$ valori di distanza, considerando solamente delle matrici di dimensione significativamente minore, ma allo stesso modo garantendo che all'interno di esse si trovino tutte le sottosequenze che sono entro una distanza R dal candidato motif. Non ci soffermeremo sui dettagli dell'algoritmo perché non sono importanti ai nostri scopi: ciò che ci interessava era introdurre l'idea alla base del suo funzionamento, perché da essa ci siamo ispirati per modellare un algoritmo di ricerca dei motif con complessità sub-quadratica.

L'idea è quella di "tagliare" tutte le chiamate alla funzione di distanza per quelle sottosequenze che sappiamo già che non potranno essere un match per il candidato motif. Questo è facilmente fattibile, in virtù della particolare funzione di distanza che abbiamo deciso di utilizzare per il nostro sistema. Ricordiamo infatti che abbiamo definito la distanza fra due sottosequenze $\boldsymbol{c}$

e $\boldsymbol{d}$, con inizio agli istanti t_c e t_d, come:

$$dist(\boldsymbol{c}, \boldsymbol{d}) = \sqrt{(c_1 - d_1)^2 + \ldots + (c_N - d_N)^2 + (o_c - o_d)^2} \tag{4.3}$$

con $o_c = t_c \bmod sd$, $o_d = t_d \bmod sd$. Di conseguenza, possiamo ottenere un bound inferiore per la distanza fra due sottosequenze, semplicemente considerando la loro posizione relativa all'interno di un periodo. In particolare,

$$dist(\boldsymbol{c}, \boldsymbol{d}) > \sqrt{(o_c - o_d)^2} = |o_c - o_d| \tag{4.4}$$

Se il secondo membro di 4.4 risulta essere $> R$, allora non sarà necessario valutare gli n campioni delle due sottosequenze per misurarne la distanza. In altre parole, non sarà necessario confrontare la sottosequenza corrente con tutte le altre sottosequenze, ma solamente con le sottosequenze che, in tutti i giorni considerati, si sono verificate attorno alla stessa ora (e quindi occupano una posizione simile all'interno del periodo); tutte le altre sottosequenze, per le quali la differenza di posizione $|o_c - o_d| > R$, possono essere scartate a priori. Vediamo nel dettaglio questo cosa comporta, supponendo di voler cercare i matches per una sottosequenza c con indice t_c e posizione relativa $o_c = t_c \mod sd$:

- vanno analizzate tutte le sottosequenze con indice $p - R \leq i \leq p + R$, $p = o_c + k \cdot sd, k = 0, 1, \ldots, N\_DAYS$;
- si può omettere di calcolare la distanza per tutte le altre sottosequenze

In figura 4.3 è illustrato il concetto: per ogni candidato motif (nella figura, una sottosequenza lunga 3 campioni) dovremo considerare tutte e sole le sottosequenze corrispondenti a quella posizione e a quelle immediatamente limitrofe, per tutti i giorni considerati.

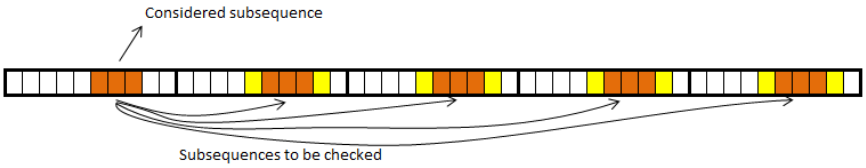

Figura 4.3: Sottosequenze da considerare per ricercare i matches del candidato motif

Infine, osserviamo che l'algoritmo proposto non ha complessità lineare: per ognuno degli $m-n+1$ candidati motif si confrontano $(m-n+1) \cdot R/sd$ sottosequenze rispetto alle $(m-n+1)$ iniziali. La complessità computazionale rimane $O(m^2)$, ma il taglio di un fattore sd/R al numero di chiamate alla funzione di distanza produce un miglioramento significativo delle prestazioni, quantificabile fra uno e due ordini di grandezza nella nostra applicazione.

Nel listato 3 si può trovare lo pseudo-codice che illustra la versione ottimizzata dell'algoritmo per la ricerca dei K motif. La variabile mi, inizializzata in riga 2, rappresenta quale motif stiamo ricercando (da 1 a K); mentre la variabile i indica il candidato motif correntemente elaborato. Tutte le sottosequenze da 1 a $m-n-fl+1$ saranno candidati motif; si noti che è necessario escludere le sottosequenze da $m-n-fl+2$ a $m-n+1$ poiché non vogliamo trovare motif che siano alla fine della serie temporale, dei quali non possediamo dati relativi all'evoluzione futura. A riga 4 possiamo vedere che l'algoritmo si ferma quando abbiamo trovato tutti e K i motif richiesti oppure se non è più possibile trovare alcun motif: questo è possibile, ad esempio, quando tutte le sottosequenze rimaste non sono adatte ad essere motif, in quanto hanno una distanza $< 2R$ da qualcuno dei motif già trovati fino a quel momento. Infatti, come possiamo vedere a riga 8, per tutti i motif successivi all'1-motif si verifica preliminarmente questa condizione, procedendo ai passi successivi solamente quando questa è soddisfatta. A questo punto,

Algorithm 3 Algoritmo *FindKMotif* nella versione ottimizzata

```
1: (loc, cnt, lst) = FindKMotif(T, K, n, R, sd, fl)
2: mi ← 1
3: nomoremotif ← 0
4: while (mi ≤ K) and (nomoremotif = 0) do
5:   nomoremotif ← 1
6:   for i = 1 to m − n − fl + 1 do
7:     suitable ← 1;
8:     if any previously found motif from l = loc(1) to loc(mi − 1) has
       dist(i, l) < 2 · R then
9:       suitable ← 0
10:    end if
11:    if suitable = 1 then
12:      count ← 0
13:      for every day d in time series T do
14:        start_index ← (i mod sd) + (sd · d)
15:        for subsequences j from start_index − R to start_index + R
           do
16:          if dist(i, j) ≤ R then
17:            count ← count + 1
18:            temp_lst ← j
19:          end if
20:        end for
21:      end for
22:      if count > cnt(mi) then
23:        cnt(mi) ← count
24:        loc(mi) ← i
25:        lst(mi) ← temp_lst
26:        nomoremotif ← 0
27:      end if
28:    end if
29:  end for
30:  mi ← mi + 1
31: end while
```

come si vede a riga 13, si ricercano i matches per il candidato motif i, comparando solamente alcune sottosequenze, come spiegato in precedenza. Infine, quando si trova che un candidato i ha un numero di match più alto rispetto ai candidati precedenti, si aggiornano i vettori ***cnt***, ***lst*** e ***loc*** relativamente alla posizione mi.

4.3.2 La funzione *weight*

Veniamo ora ad altri dettagli riguardanti l'implementazione dell'algoritmo. È interessante segnalare che abbiamo implementato una versione pesata di k-means, nella quale cioè ogni motif non ha la stessa importanza nella determinazione del centro del cluster. In questo modo, ogni motif i sarà caratterizzato da un peso (contenuto in $wgh(i)$) e il centro del motif sarà ottenuto dalla media pesata dei motif appartenenti al cluster; similmente, la previsione dell'evoluzione futura per quel motif si otterrà con la media pesata delle evoluzioni future dei vari motif. Un punto cruciale è stato quello di definire precisamente il metodo per il calcolo del peso; volevamo che esso fosse funzione di due parametri:

1. il numero di occorrenze del motif in un periodo che parte da ED (*Expire Days*) giorni fa, fino ad ora. Questa informazione viene salvata, per ogni motif i, in $cnt(i)$ al momento della ricerca dei motif (essenzialmente, se la lunghezza della learning phase corrisponde con ED, che è la condizione che noi abbiamo simulato, $cnt(i)$ corrisponde al numero di match trovati durante l'algoritmo FindKMotif). Nelle elaborazioni successive della working phase, mantenere aggiornata precisamente questa informazione sarebbe stato dispendioso; infatti sarebbe stato necessario mantenere un'indicazione temporale per ogni match, per non conteggiarlo più dopo ED giorni da quando si fosse presentato. Abbia-

mo preferito quindi utilizzare un metodo più "empirico" ma ugualmente efficace: all'inizio di ogni giorno, viene sottratta al vettore ***cnt*** (quindi a tutti i k motif) una quantità pari a $1/ED$ del valore iniziale di ***cnt***. Ovviamente, quando durante la working phase un determinato motif si ripresenta, allora la quantità corrispondente viene incrementata di 1; inoltre, il valore iniziale viene settato con questo nuovo valore. Si può obiettare che questo procedimento non sia equo nel caso in cui molti dei match per qualche particolare motif siano concentrati in un breve periodo temporale, e che quindi non sia garantito che il contenuto di $cnt(i)$ rappresenti effettivamente, in ogni momento, il numero di match del motif i negli ultimi ED giorni. Rappresenta però una buona stima di quel valore ed inoltre, cosa che per noi era fondamentale, garantisce che dopo ED giorni senza che il motif si ripresenti, si abbia $cnt(i) = 0$;

2. la lontananza nel tempo dell'ultimo match per il motif i. Vogliamo che il peso di ogni motif vada decrescendo via via che si avanza nella sequenza e non si presentano altri match di quel motif; sostanzialmente, a parità di numero di match, un motif che si è presentato recentemente avrà un peso maggiore rispetto ad un altro che si è presentato per l'ultima volta più tempo fa. Questa informazione è salvata, per ogni motif i, in $lst(i)$ e viene aggiornata ogni volta che si verifica un match per il motif.

Ognuna di queste caratteristiche va a determinare uno dei due fattori che comporrà il peso complessivo. I due fattori, che chiameremo wgh_cnt e wgh_lst, si possono ottenere con diverse funzioni di ***cnt*** e ***lst***. Le funzioni dovevano però rispondere a due esigenze:

1. avere valori compresi fra 0 (minimo) e 1 (massimo);

2. - per la funzione $wgh\_cnt(i)$, essere monotona crescente con $cnt(i)$;

   - per la funzione $wgh\_lst(i)$, essere monotona decrescente con $lst(i)$.

Abbiamo quindi messo a punto due funzioni che rispondessero alle caratteristiche; una con un andamento lineare, l'altra con andamento esponenziale. Qui sotto si può trovare la versione di wgh_cnt e wgh_lst con andamento lineare:

$$lst\_wgh(i) = \max(0, 1 - (lst(i)/EP)) \tag{4.5}$$

$$cnt\_wgh(i) = \min(1, cnt(i)/CM); \tag{4.6}$$

dove $EP = ED \cdot sd$ esprime lo stesso concetto di ED, solamente espresso in campioni anziché in giorni; mentre CM rappresenta il numero di match che è necessario avere per ottenere il punteggio massimo: nelle nostre simulazioni abbiamo impostato questo parametro uguale ad ED, che significa che è necessario mediamente un match al giorno (negli ultimi ED giorni) per avere il punteggio massimo.

Queste sono le funzioni wgh_cnt e wgh_lst nella versione esponenziale:

$$lst\_wgh(i) = e^{-d \cdot lst(i)} \tag{4.7}$$

$$cnt\_wgh(i) = e^{-c \cdot (ED - cnt(i))} \tag{4.8}$$

dove c e d sono due coefficienti calcolati appositamente per rispettare i bound previsti al punto 2) delle condizioni per la funzione weight. In particolare, per la funzione wgh_lst abbiamo imposto che per un motif con ultima occorrenza ED giorni fa (e quindi EP sottosequenze fa), il peso fosse pari ad

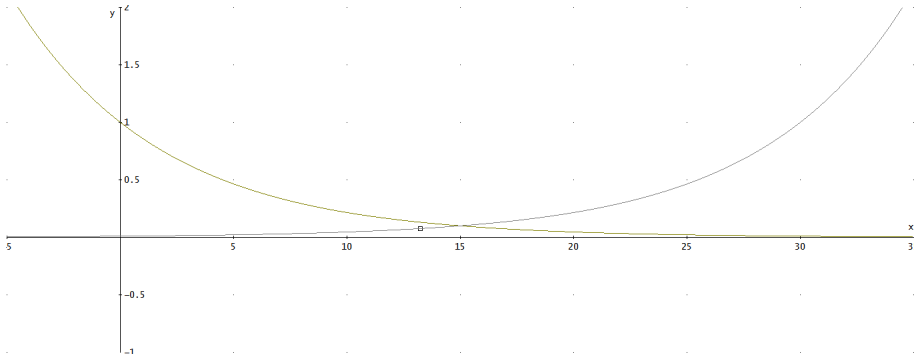

Figura 4.4: Andamento delle due funzioni wgh_cnt e wgh_lst con andamento esponenziale ($ED = 30, CM = 30, \epsilon = 0.01$)

una quantità prefissata ϵ:

$$e^{(-d \cdot EP)} = \epsilon \Rightarrow d = \frac{\ln(\epsilon)}{EP} \tag{4.9}$$

e analogamente per wgh_cnt, imponendo invece che un motif con conteggio pari a 0 avesse un peso sempre pari a ϵ:

$$e^{-c \cdot (ED - 0)} = \epsilon \Rightarrow c = \frac{\ln(\epsilon)}{ED} \tag{4.10}$$

Un altro aspetto da stabilire è stato il modo in cui le due funzioni avrebbero dovuto comporsi per ottenere la funzione wgh finale. Al riguardo, c'erano essenzialmente due possibilità: sommare i due fattori, oppure moltiplicarli tra loro. Alla fine abbiamo deciso di operare una moltiplicazione fra le due funzioni, la quale corrisponde ad un'operazione logica di AND fra le due condizioni che analizziamo; il nostro pensiero è stato che un motif è veramente interessante, e deve quindi avere un peso significativo, solo se è abbastanza frequente **e** se si è verificato di recente. Abbiamo quindi posto, per un motif i,

$$wgh(i) = wgh\_cnt(i) \cdot wgh\_lst(i) \tag{4.11}$$

4.3.3 L'algoritmo k-means

In questa sezione vedremo come è stato implementato l'algoritmo k-means, basandoci sul procedimento delineato nel capitolo 3.

Ricordiamo che, per prima cosa, è necessario inizializzare l'algoritmo assegnando in qualche modo dei valori iniziali alle medie dei cluster. In letteratura si possono trovare due diversi metodi per operare questa inizializzazione:

1. Si inizializzano le medie con k fra gli N valori dell'intero insieme, scelti casualmente. Questo metodo, noto come *Forgy*, ha l'effetto di spargere i centri dei cluster in tutto lo spazio degli elementi.

2. Si assegna casualmente ogni elemento dell'insieme ad uno dei k cluster, e poi si calcolano le medie dei gruppi così ottenuti. Questo metodo è detto *Random Partition* e, come si può intuire, produce un effetto contrario rispetto al precedente, posizionando i centri dei cluster in prossimità del centro dell'insieme.

In [9], si dimostra come, generalmente, il metodo Random Partition faccia osservare risultati migliori. Tuttavia, quale dei due metodi sia da preferire è fortemente dipendente dalla struttura dei dati, dalla loro distribuzione nello spazio e dalla loro variabilità. Noi abbiamo provato ad implementare entrambi i metodi nel nostro algoritmo di k-means; abbiamo quindi osservato che i risultati finali (quelli del forecast sull'evoluzione futura delle sottosequenze) erano leggermente migliori utilizzando il Random Partition. Questo si può probabilmente spiegare con il fatto che, con il metodo Forgy, talvolta vengono scelte delle sottosequenze ai margini dell'insieme come centri del cluster; il cluster che viene così a formarsi sarà scarsamente popolato, ed

Algorithm 4 Pseudo-codice per l'algoritmo k-means

1: $(kmns\_mtx, mns, mns\_off) = \text{kMeans}(mot\_mtx, off, wgh, k)$
2: **for** $i = 1$ to K **do**
3: $r \leftarrow \text{random}(1, k)$
4: $kmns\_mtx(i, r) \leftarrow 1$
5: **end for**
6: $(mns, mns\_off) \leftarrow \text{updatemeans}(kmns\_mtx, wgh, mot\_mtx, off)$
7: $stable \leftarrow 0$;
8: **while** $stable = 0$ **do**
9: $(stable, kmns\_mtx) \leftarrow \text{assignment}(mot\_mtx, mns, kmns\_mtx)$
10: $(mns, mns\_off) \leftarrow \text{updatemeans}(kmns\_mtx, wgh, mot\_mtx, off)$
11: **end while**

abbiamo osservato che talvolta si componeva solamente del centro, quindi era di fatto inutile ai fini della classificazione.

Veniamo ora all'implementazione vera e propria. Di per sé il codice è molto semplice e ricalca in maniera fedele l'idea per l'algoritmo abbozzata nel capitolo 3. Vi si trova innanzitutto una parte di inizializzazione, secondo il metodo Random Partition appena visto; quindi inizia un ciclo *while* all'interno del quale si svolgono ripetutamente le due fasi di assegnazione e di aggiornamento delle medie. Il listato 4 mostra il codice completo.

Passiamo ora, quindi, ad analizzare le due sottofunzioni che svolgono le operazioni fondamentali di k-means: *updatemeans* e *assignment*. Si può trovare il codice di entrambe le funzioni a pagina 58[1].

La prima è una funzione molto semplice, soprattutto grazie alla forma che abbiamo scelto per memorizzare le diverse strutture dati. Si procede aggiornando la media di un cluster alla volta, in questo modo: per prima cosa, si "screma" il vettore dei pesi $\boldsymbol{wgh}$ azzerando le posizioni relative ai motif che non appartengono al cluster considerato, salvando il risultato nel vettore $\boldsymbol{zw}$ (riga 3 dell'algoritmo). A questo punto, l'operazione fondamentale: si

[1] Si noti che nello pseudo-codice riportato, si scrive con $\cdot$ l'operazione di moltiplicazione elemento per elemento fra matrici della stessa dimensione, mentre con $\times$ si intende la moltiplicazione fra matrici vera e propria

Algorithm 5 Pseudo-codice per la subroutine updatemeans

1: $(mns, mns\_off) =$ updatemeans$(kmns_mtx, wgh, mot_mtx, off)$
2: **for** $i = 1$ to k **do**
3: $zw \leftarrow kmns\_mtx(i) \cdot wgh$
4: $cl \leftarrow zw \times mot\_mtx$
5: $ol \leftarrow zw \times off_vec$
6: $tw \leftarrow sum(zw)$
7: $mns(i) \leftarrow cl/tw$
8: $mns\_off(i) \leftarrow ol/tw$
9: **end for**

Algorithm 6 Pseudo-codice per la subroutine assignment

1: $(stable, kmns\_mtx) =$ assignment$(mot_mtx, mns, kmns_mtx)$
2: $stable \leftarrow 1$
3: **for** $i = 1$ to K **do**
4: $nrst\_mean \leftarrow 1$
5: $dist\_nrst\_mean \leftarrow dist(i, 1)$
6: **for** $j = 2$ to k **do**
7: $d \leftarrow$ kmDist(i, j)
8: **if** $d < dist\_nrst\_mean$ **then**
9: $dist\_nrst\_mean \leftarrow d$
10: $nrst\_mean \leftarrow j$
11: **end if**
12: **end for**
13: **if** $kmns\_mtx(nrst\_mean, i) = 0$ **then**
14: $stable \leftarrow 0$
15: put to 0 the i-th column of $kmns\_mtx$
16: $kmns\_mtx(nrst\_mean, i) \leftarrow 1$
17: **end if**
18: **end for**

esegue una moltiplicazione fra la matrice contenente i motif ***mot_mtx*** e il vettore ***zw***, ottenendo in questo modo una combinazione lineare di tutti i motif, ognuno con il proprio peso come coefficiente (si noti che, per via dell'operazione svolta prima, i motif che non appartengono a questo cluster partecipano alla combinazione lineare con peso nullo). Nell'algoritmo questa operazione è svolta in riga 4 per i motif, in riga 5 per gli offset. Infine, si normalizza il vettore così ottenuto dividendo per la somma dei pesi dei motif considerati (righe 7 e 8).

Per quanto riguarda la funzione di assegnamento, si procede semplicemente scorrendo tutti i motif, cercando per ognuno quale, fra i vari centri dei cluster, è quello a distanza minore. A questo punto si controlla (riga 13) se il motif che stiamo analizzando apparteneva già a quel cluster; in caso negativo, si aggiorna la variabile *stable* che poi verrà ritornata all'algoritmo k-means, si azzera l'assegnamento attuale del motif e si procede ad assegnarlo al nuovo gruppo.

4.4 La working phase

Durante la working phase devono essere eseguite tre operazioni fondamentali:

1. identificare la sequenza corrente $\boldsymbol{c}$ con uno dei K motif che sono stati definiti finora (inizialmente durante la learning phase e successivamente modificati). Durante questa fase, sarà necessario prevedere come comportarsi nel caso in cui la sottosequenza non sia match per nessuno dei motif;

2. basandosi sui risultati del punto 1, utilizzare le informazioni del cluster cui appartiene il motif ***cm*** che abbiamo associato alla sottosequenza, per ottenere informazioni sull'evoluzione futura di $\boldsymbol{c}$;

3. infine, aggiornare le strutture dati relative al motif ***cm*** e, conseguentemente, il suo peso; dopo questa operazione dovranno essere ricalcolate le medie dei cluster ed, eventualmente, modificati gli assegnamenti dei motifs ai cluster.

L'approccio scelto può sembrare inutilmente contorto: perché andare a ricercare un match per la sottosequenza fra tutti i motifs, e poi utilizzare il cluster relativo al motif per ottenere le informazioni? Non sarebbe stato possibile assegnare direttamente la sottosequenza $\boldsymbol{c}$ al cluster con centro a distanza minore, ed utilizzare quello per il forecast? La risposta è no, perché in quel caso saremmo incappati nello stesso errore che abbiamo voluto evitare durante la learning phase, e cioè che essenzialmente saremmo finiti con il fare k-means fra tutte le sottosequenze, con i risultati indesiderabili che abbiamo già esposto nella sezione 3.3. Ecco il motivo per cui la sottosequenza corrente ha solo un effetto indiretto nell'aggiornamento del cluster: essa comporterà una variazione del peso del motif cui è associata e questo, in seconda battuta, provocherà l'aggiornamento delle medie dei gruppi. Questo implica anche che, durante tutta la working phase, il numero di elementi coinvolti nell'operazione di k-means non verrà mai modificato: saranno sempre i K motif iniziali, nel senso che il loro numero rimarrà invariato, sebbene alcuni motif potranno essere sostituiti con altri nel corso dell'esecuzione, come vedremo più avanti.

Un'altra cosa che è bene sottolineare è che, mentre i cluster vengono continuamente modificati nelle medie e nella composizione, per via dell'evolversi dei pesi dei motifs, per i motifs stessi questo non avviene. In altre parole, un motif che viene individuato durante la learning phase, rimarrà invariato fino al termine dell'algoritmo, oppure fino alla sua sostituzione con un altro motif: le sottosequenze che verranno via via associate ad esso durante la

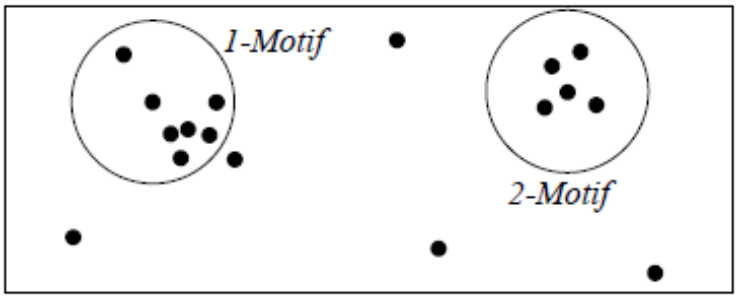

Figura 4.5: Rappresentazione grafica dell'individuazione di un motif

working phase avranno il solo effetto di modificare il suo peso, ma la sequenza di bit del motif rimarrà immutata. Per capire meglio i motivi di questa decisione, si pensi al modo in cui il motif viene individuato; si può fare riferimento a figura 3.7, che qui riportiamo per comodità. Quando il motif viene scelto, esso incarna il centro fra tutte le sottosequenze che, in qualche modo, rappresenta; modificare la sequenza di bit del motif, ad esempio mediandola con le sottosequenze che via via vengono ad esso associate, significa spostare quel centro e, di conseguenza, la posizione della sfera di raggio R che da quel centro ha origine. Questo potrebbe provocare che alcune delle sottosequenze associate inizialmente al motif non vi siano più incluse, mentre altre, diverse, si troverebbero entro i bordi della sfera; inoltre, fatto ben più preoccupante, il centro del motif potrebbe venirsi a trovare ad una distanza $< 2R$ dal centro di un altro motif, portando le due rispettive sfere ad intersecarsi. Per questi motivi, considereremo ogni motif come "una cosa sola" con tutte le sottosequenze che ad esso corrispondono; in altre parole, ogni volta che si verifica una sottosequenza che ha un match con un determinato motif, nei fatti ci comporteremo esattamente come se si fosse verificato il motif stesso: chiaramente, tanto più è ridotto il raggio R, tanto più questo approccio trova giustificazione.

4.4.1 Identificazione delle sottosequenze

Ora che abbiamo chiarito il metodo con cui procedere, possiamo delineare il codice per implementare le varie fasi che abbiamo elencato sopra. La prima parte, ossia quella relativa all'identificazione delle sottosequenze con un motif, è piuttosto semplice; nel listato 7 si può trovare il relativo pseudo-codice. Si scorrono tutti i K motifs alla ricerca di uno a distanza $< R$ dalla sottosequenza corrente, interrompendo la ricerca dopo averlo trovato: quello sarà il motif ***cm*** che verrà utilizzato per le elaborazioni successive. L'unica cosa degna di nota è che, approfittando del fatto che durante l'esecuzione vengono calcolate tutte le distanze fra la sottosequenza $\boldsymbol{c}$ e i vari motifs, si cerca anche se la sottosequenza ha motif a distanza $< 2R$: chiameremo questi motif *collisioni* per la sottosequenza $\boldsymbol{c}$, in quanto impedirebbero alla sottosequenza di diventare essa stessa un motif. Questo sarà utile nel caso in cui non si trovi nessun motif a distanza $< R$ (e dunque, nessun match

Algorithm 7 Pseudo-codice per l'algoritmo CheckMotif

```
(cm, ncoll, coll) = CheckMotif(c, mot_mtx, R)
found ← 0
ncoll ← 0
i ← 0
while (i ≤ K)and(found = 0) do
  d = dist(c, mot_mtx(i, :))
  if d ≤ R then
    found ← 1
    cm ← i
  else
    if d ≤ 2R then
      ncoll ← ncoll + 1
      coll(ncoll) ← i
    end if
  end if
  i ← i + 1
end while
```

per la sottosequenza $\boldsymbol{c}$); in questo caso, come vedremo più avanti, una delle possibili opzioni sarà quella di eleggere la sottosequenza stessa a motif: sarà quindi utile sapere in anticipo se questo è possibile oppure no. Il numero di motif che collidono con $\boldsymbol{c}$ è memorizzato in *ncoll*, le relative posizioni nel vettore ***coll***.

4.4.2 Previsione dell'evoluzione futura

Siamo giunti al punto cruciale di tutto il nostro sistema, quello dove dovremo utilizzare le informazioni raccolte e memorizzate per tradurle in previsioni sull'evoluzione futura della sottosequenza corrente $\boldsymbol{c}$. Giunti a questo punto, non ci sono particolari difficoltà concettuali da superare, ma il numero di possibilità presenti rende il codice piuttosto complicato; cercheremo quindi di procedere con ordine, analizzando una alla volta le possibilità che si possono verificare:

- la sottosequenza corrente $\boldsymbol{c}$ corrisponde ad un motif $\boldsymbol{cm}$;
- la sottosequenza corrente $\boldsymbol{c}$ non corrisponde a nessun motif, ma può essere essa stessa un motif;
- la sottosequenza corrente $\boldsymbol{c}$ non corrisponde a nessun motif e collide con un motif già esistente che può essere sostituito;
- la sottosequenza corrente $\boldsymbol{c}$ non corrisponde a nessun motif e non può essere motif in alcun modo.

La sottosequenza corrente c corrisponde ad un motif cm

Questo è il caso concettualmente più semplice poiché non deve essere svolta nessun'altra azione rispetto a quelle già descritte all'inizio di questa sezione.

Per prima cosa, vanno aggiornate le statistiche relative al motif ***cm***, incrementando $cnt(cm)$ ed aggiornando $lst(cm)$ con la posizione corrente; queste modifiche provocheranno un aggiornamento del peso del motif $wgh(cm)$. A sua volta, l'aggiornamento del peso farà sì che venga modificata la media del cluster cui appartiene il motif ***cm***, operazione che può innescare dei cambiamenti negli assegnamenti dei motif ai cluster. Per svolgere le operazioni di aggiornamento delle medie e delle appartenenze, abbiamo riutilizzato il codice scritto per l'algoritmo k-means, implementando un meccanismo molto simile. Tutte queste operazioni di aggiornamento delle strutture dati non vengono effettuate nel caso in cui la sottosequenza ***c*** sia un trivial match per ***cm***: cioè, se la sottosequenza precedente era stata anch'essa associata allo stesso motif.

In questo caso, la previsione dell'evoluzione futura si può basare semplicemente sulle informazioni relative al cluster cui il motif ***cm*** appartiene: ricordiamo che si può trovare la media pesata delle evoluzioni future degli appartenenti ad ogni cluster nella matrice ***fm_mtx***. Confrontando i bit dell'evoluzione prevista con quella effettiva, possiamo a questo punto costruire diverse statistiche circa l'affidabilità del sistema.

La sottosequenza corrente *c* non corrisponde a nessun motif, ma può essere essa stessa un motif

Veniamo ora ad alcuni casi più complicati. Il primo caso che andremo ad analizzare è quello per cui la sottosequenza non ha match con nessuno dei motif esistenti e, allo stesso tempo, nessuno di essi si trova ad una distanza minore di $2R$. Tuttavia, questo non è sufficiente a fare della sottosequenza ***c*** un motif. Ricordiamo, infatti, che abbiamo deciso che il numero iniziale K di motifs dovrà rimanere invariato durante l'esecuzione dell'algoritmo; nel

caso in cui si scelga un valore di R non troppo elevato, il rischio è altrimenti quello di finire per aumentare in modo arbitrario il numero di motif: questo rende inutile la scelta iniziale del valore di K, e ci avvicina pericolosamente al caso che stiamo cercando di evitare a tutti i costi, in cui una larga parte delle sottosequenze finisce per essere considerata nell'algoritmo k-means.

Per questi motivi, se intendiamo fare in modo che la sottosequenza $\boldsymbol{c}$ diventi essa stessa un motif, necessariamente essa dovrà prendere il posto di un motif già esistente. La scelta naturalmente ricadrà sul motif che, in quel momento, ha peso minore (detto $\boldsymbol{mwm}$, *minimum weight motif*) ma riteniamo che non sia giusto procedere in ogni caso a questa sostituzione. Abbiamo deciso di porre due condizioni perché questa sostituzione possa avvenire:

- la sottosequenza $\boldsymbol{c}$ deve avere un peso più alto rispetto al motif da sostituire $\boldsymbol{mwm}$. Questa è una condizione difficile da verificare poiché $\boldsymbol{c}$, non essendo motif, al momento non possiede un peso: è necessario andare a ritroso negli ultimi ED giorni della serie temporale per trovare il numero di volte in cui la sottosequenza si è verificata, per poter valutare se si tratta di un'occorrenza occasionale oppure di qualcosa di più ricorrente che merita di essere considerato come motif;

- il motif da sostituire $\boldsymbol{mwm}$ deve avere un peso sufficientemente basso. Questa può sembrare una condizione superflua, poiché $\boldsymbol{mwm}$ è già, fra tutti i motif, quello con il peso *più basso*. In realtà questa condizione è stata imposta pensando a quei casi in cui tutti i motif hanno pesi simili e, considerando la somma dei pesi, ciascun motif vi contribuisce per una parte attorno ad $1/K$. Noi abbiamo imposto che, perché il motif sia sostituibile, $wgh(mwm)$ deve contribuire alla somma totale dei pesi per una quantità non superiore a $2/K$. Il motivo di questa decisione

è che riteniamo che sia sbagliato sostituire un motif che, anche se è quello a peso minore, ha una certa importanza nell'insieme dei pesi; infatti, se *mwm* ha un peso comunque significativo, è perché si tratta di un motif che si è verificato recentemente, e che con buona probabilità si verificherà di nuovo. Se lo sostituissimo ora, al suo verificarsi ci troveremmo di fronte allo stesso problema, innescando una girandola di sostituzioni fra motif dal peso simile che non gioverebbe sicuramente alla stabilità e all'efficienza del sistema.

Solamente se queste due condizioni sono verificate, allora la sottosequenza $\boldsymbol{c}$ diventerà motif al posto di $\boldsymbol{mwm}$. In questo caso, si dovranno aggiornare tutte le strutture dati relative al nuovo motif; inoltre, si osservi che il nuovo motif potrà essere assegnato ad un cluster diverso da quello cui era assegnato $\boldsymbol{mwm}$: si dovrà procedere per prima cosa all'assegnazione al cluster con centro più vicino, per poi ricalcolare le medie dei cluster, proseguendo finché non si raggiunge una configurazione stabile, come nell'algoritmo k-means.

A questo punto, sarà possibile fare il forecast come se ci trovassimo nel punto precedente: ora la sottosequenza $\boldsymbol{c}$ è un motif a tutti gli effetti ed appartiene ad un cluster, che possiamo utilizzare per ricavare le informazioni sull'evoluzione futura.

La sottosequenza corrente $\boldsymbol{c}$ non corrisponde a nessun motif e collide con un motif già esistente che può essere sostituito

Anche questo caso è piuttosto frequente e riguarda tutte quelle volte in cui la sottosequenza $\boldsymbol{c}$ non è un match per nessun motif (poiché si trovano tutti a distanza $> R$) ma, allo stesso tempo, non può essere essa stessa un motif perché uno o più di quelli già esistenti si trovano ad una distanza $< 2R$. Nel caso in cui il motif che provoca la collisione sia solamente uno (lo chiameremo

coll), si potrebbe pensare di sostituirlo con la sottosequenza ***c***, facendola diventare essa stessa un motif. Perché la sostituzione sia consentita, abbiamo deciso di applicare le stesse regole che abbiamo visto al punto precedente riguardo il peso del motif da sostituire rispetto al peso del nuovo motif e rispetto al totale dei pesi di tutti i motif. Se entrambe le condizioni sono verificate, allora si procede esattamente come nel caso del punto precedente, facendo diventare motif la sottosequenza ***c*** al posto di ***coll***, provvedendo ad aggiornare le relative strutture dati e a calcolarne il peso; quindi, si assegnerà il nuovo motif al cluster con il centro più vicino, procedendo poi ad aggiornare le medie dei cluster. A questo punto, come al punto precedente, possiamo sfruttare le informazioni del cluster per avere informazioni sull'evoluzione futura di ***c***.

Nel caso in cui, invece, i motifs che collidono con ***c*** siano più di uno, non procederemo ad alcuna sostituzione. Non ci sembrava una buona idea quella di rimuovere tutti i motif interessati per inserire quello nuovo, perché secondo noi facendo in questo modo era alto il rischio di perdere informazioni, ritrovandosi alla fine con un set di motifs meno significativo. La soluzione ideale sarebbe stata quella di "riorganizzare" completamente i motifs interessati, spostandone i centri e le relative sfere per massimizzare il numero di sottosequenze incluse, compresa ***c*** se possibile; però ci sembrava una strada difficilmente percorribile e molto dispendiosa dal punto di vista computazionale. Dunque in questo caso, come per tutti gli altri casi in cui non fosse possibile inquadrare ***c*** all'interno di un motif, seguiremo un'altra strada, che spiegheremo nel dettaglio nel prossimo punto.

La sottosequenza corrente *c* non corrisponde a nessun motif e non può essere motif in alcun modo

Come accennato, in questo punto spiegheremo come abbiamo deciso di procedere per tutte quelle situazioni nelle quali la sottosequenza $\boldsymbol{c}$ non è match per nessun motif e, per vari motivi, essa stessa non può essere un motif; i motivi di questa situazione possono essere molteplici e sono stati introdotti nei punti precedenti, comunque li riassumiamo qui per completezza:

1. $\boldsymbol{c}$ non collide con nessun motif ma il motif a peso minimo $\boldsymbol{mwm}$, di cui dovrebbe prendere il posto, ha peso maggiore;
2. $\boldsymbol{c}$ non collide con nessun motif ma il motif a peso minimo $\boldsymbol{mwm}$, di cui dovrebbe prendere il posto, ha peso con rapporto superiore a $1/2K$ rispetto al totale dei pesi;
3. $\boldsymbol{c}$ collide con un motif $\boldsymbol{coll}$, ma non può prenderne il posto poiché esso ha peso maggiore;
4. $\boldsymbol{c}$ collide con un motif $\boldsymbol{coll}$, ma non può prenderne il posto poiché esso ha peso con rapporto superiore a $1/2k$ rispetto al totale dei pesi;
5. $\boldsymbol{c}$ collide con due o più motif già esistenti.

In tutti questi casi, il metodo "classico" che abbiamo utilizzato finora per fare i forecast sulle evoluzioni future non è praticabile; non riuscendo ad associare la sottosequenza $\boldsymbol{c}$ a nessun motif, e nell'impossibilità di aggiungerla direttamente in uno dei cluster per il k-means, per i motivi che abbiamo spiegato più volte, non ci resta che gestire questo caso nel miglior modo possibile, evitando di fare previsioni che non siamo in condizioni di fare.

Nei casi sopra citati, quindi, abbiamo deciso di riutilizzare il forecast che è stato calcolato dal sistema per la sottosequenza immediatamente pre-

cedente a $\boldsymbol{c}$; ovviamente il primo degli fl bit della sequenza dovrà essere eliminato (poiché riguardava la rilevazione che è stata appena effettuata) e i rimanenti bit shiftati, lasciandoci con una sequenza di forecast lunga $fl - 1$ bit. Se, sfortunatamente, durante la sottosequenza successiva dovesse ripresentarsi questa situazione, la procedura può essere ripetuta fino a fl volte, diminuendo via via i bit della sequenza di forecast; alla prima sottosequenza che ricada in uno degli altri casi, tuttavia, la sequenza di previsione verrebbe ricostruita integralmente con i suoi fl bit.

L'approccio per gestire le previsioni in questo caso particolare può sembrare in qualche modo "rinunciatario", ma noi crediamo che fosse il solo modo di procedere, coerentemente con le caratteristiche del sistema e con il modo in cui è stato progettato. Inoltre, se andiamo ad analizzare nel dettaglio i vari motivi che ci possono portare a questo punto, troviamo che questo modo di procedere si rivela adatto per gran parte di essi:

- nel caso 1, il fatto di avere una sottosequenza lontana da tutti i motif già esistenti, ma con un peso così basso, è chiaro indice del fatto che ci troviamo di fronte a una sottosequenza anomala, che si è verificata raramente in passato: in questo caso, quella di rifiutarsi di fare una previsione può essere la scelta più corretta. Si osservi, tuttavia, che se la stessa sottosequenza dovesse verificarsi nuovamente nei giorni successivi, via via il suo peso andrebbe ad aumentare (ricordiamo che per calcolare il peso di un nuovo candidato a motif, prima verifichiamo le sue occorrenze negli ED giorni precedenti) e quindi essa diventerebbe sempre meno anomala, finendo per avere peso sufficiente ad essere inserita come motif;

- nei casi 3, 4 e 5, che riguardano le collisioni con altri motif, riteniamo che l'approccio seguito non possa provocare grossi danni. Infatti, se

la sottosequenza corrente $\boldsymbol{c}$ si trova ad una distanza $R < d < 2R$ da qualcuno dei motif già esistenti, pensiamo che sia altamente probabile che una fra le sottosequenze immediatamente precedenti o successive si trovi a distanza $< R$ da qualcuno di quegli stessi motif, costituendo quindi un match. Quindi ci troviamo in uno dei due casi: o la sequenza di forecast che andiamo a riutilizzare è già stata ottenuta utilizzando uno di quei motif che collidono (nel caso di match con una delle sottosequenze che precedono $\boldsymbol{c}$); oppure, nel futuro immediato, potremo ottenere una nuova sequenza di forecast utilizzando sempre uno di quei motif (nel caso di match con una delle sottosequenza che seguono $\boldsymbol{c}$). In ogni caso, la scelta operata sembra essere indolore.

Rimane fuori il caso numero 2 (nessuna collisione, ma motif a peso minimo con peso troppo alto per poter essere sostituito). In questo caso, obiettivamente, la soluzione migliore sembrerebbe quella di poter aggiungere un

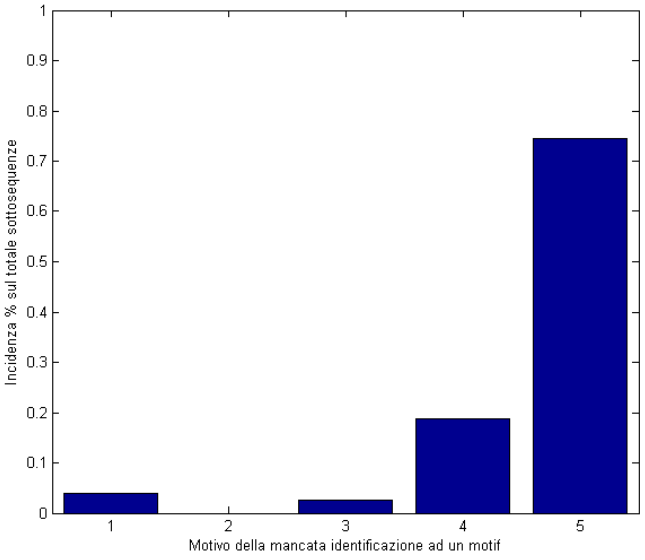

Figura 4.6: Incidenza delle diverse cause sul totale di sottosequenze non associabili ad un motif

motif ulteriore, per far spazio alla nuova sottosequenza $\boldsymbol{c}$ senza rinunciare ad un motif comunque rilevante. Dobbiamo però osservare che, sebbene questa possibilità sia giustamente stata inclusa per completezza, essa si è rivelata essere piuttosto rara: su circa 30000 sottosequenze analizzate, non si è mai verificata. In figura 4.6 possiamo trovare l'incidenza di ciascuna delle 5 cause elencate, sul totale delle sottosequenze trattate in questo modo.

Capitolo 5

Test del sistema e risultati

In questo capitolo, spiegheremo come si è proceduto per testare il nostro sistema, illustrando i risultati ottenuti.

5.1 Test della parte hardware

Per quanto riguarda la parte hardware, abbiamo posizionato i quattro sensori da noi assemblati in due diversi siti: due all'interno di un'abitazione, e due nel laboratorio SIGNET del Dipartimento di Ingegneria dell'Informazione. I due sensori che si trovavano all'interno del laboratorio erano indipendenti fra loro, essendo collegati tramite cavo USB al server su cui venivano poi memorizzati i dati; entrambi possedevano un sensore di movimento e, in questo modo, potevano monitorare due zone diverse della stanza che ospita il laboratorio. Al contrario, per quelli all'interno dell'abitazione, uno dei due si trovava collegato all'host fisso, mentre il secondo era alimentato a batterie ed era posizionato su una parete in uno degli ambienti della casa; in questo caso, i due sensori comunicavano tra loro via radio.

Veniamo ora a una valutazione delle prestazioni della parte hardware del sistema. Riguardo il TelosB non c'è molto da dire: essendo in uso da molto

tempo nel nostro laboratorio, non avevamo dubbi sulla sua affidabilità, e infatti non ha dato alcun tipo di problema durante gli esperimenti.

Per quanto riguarda il rilevatore di presenza, bisogna dire che una volta superate le difficoltà iniziali (di cui abbiamo riferito in sezione 2.4), si è dimostrato tutto sommato un buon prodotto in relazione al suo costo. Bisogna osservare poi, anche ricordando il principio di funzionamento del sensore a cui abbiamo accennato in sezione 2.1.2, che esso dovrebbe essere chiamato più correttamente rilevatore *di movimento* e non *di presenza*; ci siamo accorti di questa differenza con le prime prove, quando abbiamo osservato che se una persona sta completamente ferma, nel raggio di rilevazione del sensore, dopo un certo tempo essa non viene più rilevata. Questo è perché il sensore si adatta piuttosto velocemente ai cambiamenti dei livelli di emissione di raggi infrarossi nell'ambiente: quando la persona entra nel raggio di rilevazione, provoca inevitabilmente una variazione di tale livello per alcune aree del sensore; a questo punto, se la persona rimane completamente ferma, i livelli per tutte le aree del sensore rimangono invariate e, poco a poco, il sensore si adatta a tali livelli finendo per considerarli parte del "rumore di fondo" e non segnalando più la presenza della persona. Si noti che invece, se la persona si muove in maniera sufficiente a far diminuire i livelli di emissione che incidono su un'area del sensore, per farli aumentare su un'altra, la sua presenza continuerà ad essere segnalata. Definire precisamente quel "in maniera sufficiente" non è semplice: sia perché si tratta di una condizione difficile da misurare, sia perché essa varia notevolmente in funzione della distanza dal sensore, delle condizioni ambientali, ecc. Dalle nostre prove empiriche, possiamo dire che a volte è sufficiente un movimento del braccio, altre volte è necessario spostarsi con tutto il corpo di qualche decina di centimetri prima di tornare ad essere rilevati.

Abbiamo posto rimedio a questa caratteristica indesiderata del rilevatore di presenza agendo sul software che regola le letture dal sensore; come abbiamo spiegato in sezione 2.3, abbiamo fatto in modo che ogni singolo campione fosse deciso sulla base di più letture consecutive, a qualche secondo di distanza; inoltre, osservando che, per i motivi visti sopra, le mancate rilevazioni erano sistematicamente più numerose dei "falsi allarmi" (che non si sono praticamente mai verificati), abbiamo abbassato la soglia di letture positive necessarie ad impostare quel campione come presenza, dal 50% al 20%.

Con questi semplici accorgimenti, siamo riusciti ad ottenere dei risultati coerenti dal sistema, che ha mostrato una buona affidabilità durante tutto il periodo di test. Confrontando la sequenza memorizzata dal sensore posto nell'abitazione con quelle che sapevamo essere la reale occupazione degli ambienti, abbiamo visto che c'è stata ugualmente qualche mancata rilevazione, ma entro un livello che riteniamo fisiologico per questo tipo di sistemi. In definitiva riteniamo che la parte hardware del sistema abbia raggiunto interamente gli obiettivi che ci eravamo prefissi, e per la quale era stata progettata.

5.2 Simulazioni

Un punto fondamentale della fase di testing è stato quello di simulazione del funzionamento del sistema di elaborazione delle sequenze. Come avevamo accennato in sezione 4.2, abbiamo verificato il funzionamento del sistema utilizzando due diversi tipi di sequenze di ingresso:

- la sequenza reale rilevata dal sensore che era stato collocato nell'abitazione per verificarne il funzionamento. Questa sequenza riguarda

30 giorni di rilevazioni, che è il periodo di tempo che solitamente, nel corso delle simulazioni, abbiamo impostato come durata per la sola learning phase; dunque riteniamo che la sua durata non sia sufficiente per simulare sia la learning phase che la working phase. In generale, quindi, utilizzeremo la sequenza nella sua interezza per la learning phase, riutilizzandola a questo punto per la working phase. Pensiamo che questo approccio non infici l'attendibilità dei nostri risultati, in quanto in tutte le nostre simulazioni abbiamo posto il parametro ED (che abbiamo definito in sezione 4.3.2) pari alla lunghezza della working phase; questo significa che l'influenza delle primissime sottosequenze della working phase sarà praticamente nullo quando questa giunge al termine e si inizia la learning phase. In altre parole, le sottosequenze della working phase, dal punto di vista dell'algoritmo, saranno viste come "nuove", anche se nuove non sono. Si noti che, ovviamente, questo discorso non sarebbe stato valido se avessimo deciso di utilizzare una sequenza molto più breve di ED, ripetendola: ad esempio, utilizzare come serie temporale per learning e working phase la ripetizione ciclica di una sequenza lunga 5 giorni non avrebbe avuto alcun senso al fine di valutare la capacità del sistema di fare previsioni;

- la sequenza simulata prodotta dall'algoritmo Office Simulator. In questo caso, non abbiamo alcun limite alla lunghezza delle sequenze simulabili; la possibilità di generare un numero potenzialmente infinito di sequenze diverse dà la possibilità di fare numerose prove sull'algoritmo, testando la sua robustezza con centinaia di esecuzioni e valutando le performance al variare dei parametri fondamentali: abbiamo infatti testato il sistema con diversi valori di n, k, R e fl. Nel caso di sequenze simulate, la learning phase rimarrà di 30 giorni, mentre la working

phase potrà essere lunga a piacere (e la sottosequenza utilizzata sarà, ovviamente, diversa da quella utilizzata per la fase di apprendimento).

Nella prossima sezione si possono trovare i risultati di queste simulazioni.

5.3 Risultati

Ci sono diversi aspetti del nostro algoritmo che abbiamo voluto monitorare per poterne valutare il funzionamento, e che esporremo in questa sezione; ovviamente il più importante sarà quello relativo alla capacità di forecasting dell'algoritmo, che noi crediamo essere straordinariamente interessante, ma non ci fermeremo a quello. Per prima cosa, analizzeremo la capacità dell'algoritmo di modellizzare in modo corretto le sottosequenze che si presentano in ingresso; successivamente, verificheremo quali sono le prestazioni del sistema per quanto riguarda la capacità di prevedere l'evoluzione futura di una sottosequenza: ci soffermeremo dettagliatamente su questo punto, cercando di individuare le configurazioni che garantiscono le migliori prestazioni e cercando di spiegare i motivi per i quali alcuni valori per i parametri sono migliori di altri; infine, verificheremo la capacità del sistema di adattarsi al variare delle condizioni ambientali.

5.3.1 Capacità del sistema di modellizzare le sequenze

Come anticipato, per prima cosa ci soffermeremo sulla capacità del sistema di riconoscere le sottosequenze che si presentano in ingresso, inserendole all'interno di uno dei cluster formati durante la learning phase. Come indice per questa valutazione abbiamo considerato la distanza media fra la sottosequenza di ingresso e il centro del cluster cui essa viene associata. Ricordiamo che, inizialmente, ogni sottosequenza viene associata ad un motif a distanza

$< R$; ogni motif è associato ad un gruppo ben definito, e noi considereremo quel gruppo come il modello per la sottosequenza in ingresso. In figura 5.1 possiamo vedere i grafici delle distanze medie, al variare dei parametri fondamentali del sistema k, n e R.

Dall'immagine possiamo vedere chiaramente come la distanza media dai centri dei cluster vari considerevolmente con il numero di cluster k; questo non è sicuramente un risultato sorprendente, poiché appare naturale che, con l'aumentare dei gruppi, la distanza media delle nuove sottosequenze dal centro di ciascuno sia destinata a diminuire. Dobbiamo però osservare che non è nemmeno possibile aumentare arbitrariamente il numero dei cluster, pena il ritrovarsi con un insieme di cluster poco popolati e, conseguentemente, poco utili a formare un modello per le sottosequenze entranti. Anche il parametro R, sebbene in misura minore, influisce sui risultati della distanza media; possiamo spiegarci questo fatto pensando che un raggio maggiore fa sì che aumenti la distanza media fra una sottosequenza e il motif cui essa è associata.

Possiamo concludere dicendo che una valutazione sulla distanza media fra le sottosequenze e i centri dei cluster che vengono loro associate, e sul numero di elementi per cluster, può essere utile a determinare un valore di k significativo per il caso preso in esame. Si può pensare, ad esempio, di procedere aumentando progressivamente k (possibilmente, eseguendo diverse volte k-means per ciascun valore considerato), finché non si osserva che l'aumentare ulteriormente il numero di cluster produrrebbe gruppi con un numero di partecipanti troppo basso. Per quanto riguarda, invece, i valori di R e n, essi influenzano particolarmente altre caratteristiche dell'algoritmo, di cui daremo conto nella sottosezione successiva.

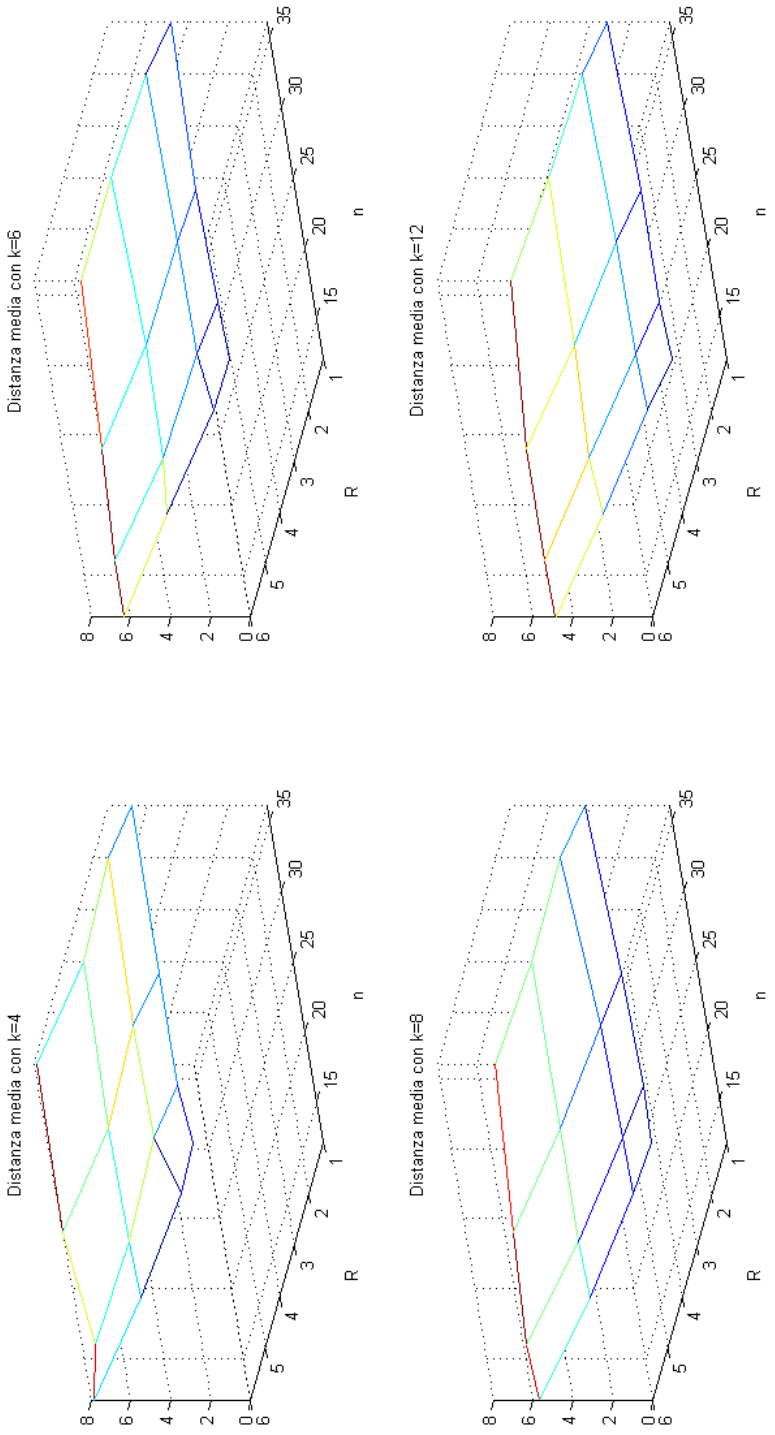

Figura 5.1: Distanza media delle sottosequenze dal centro del cluster cui sono associate, al variare di k, n, R.

5.3.2 Capacità del sistema di prevedere l'evoluzione futura

Ora veniamo alla parte più caratteristica dell'intero sistema, ovvero la capacità di prevedere l'evoluzione futura di quella che sarà l'occupazione del locale. In figura 5.2 si può osservare la percentuale di campioni corretti sul totale di campioni previsti dall'algoritmo al variare di k, n e R. Osserviamo per prima cosa che, con le configurazioni migliori dei parametri considerati, il sistema si posiziona stabilmente oltre il 90% di bit correttamente indovinati! Vediamo più nel dettaglio quali sono i valori che consentono di ottenere i parametri migliori.

I quattro grafici rappresentano gli esperimenti compiuti con quattro diversi valori di k. Raffrontando i diversi grafici, possiamo vedere che il valore di k influenza le performance dell'algoritmo, nel senso che un numero più elevato di cluster consente di ottenere risultati migliori. Questo non è un risultato inaspettato: si può facilmente spiegare con il fatto che, con più gruppi, è possibile giungere ad una classificazione più precisa, discriminando alcuni motif che altrimenti, con un valore minore di k, sarebbero stati associati allo stesso cluster. Era lecito aspettarsi questo risultato anche alla luce delle considerazioni fatte nella sottosezione precedente: è chiaro che, tanto migliore sarà la modellizzazione, tanto più accurate risulteranno poi le rispettive previsioni.

Ora consideriamo un singolo grafico, ad esempio quello relativo a $k = 12$, ed analizziamo la dipendenza dei risultati dagli altri due parametri R ed n. Un fatto che sicuramente salta all'occhio, ad una prima analisi dei risultati, è la forte dipendenza dal parametro R (il raggio utilizzato per determinare i motifs). Questo non ci sorprende più di tanto, poiché da subito era chiaro che quello sarebbe stato un parametro fondamentale nel determinare il funzionamento del sistema; quello che invece non ci saremmo

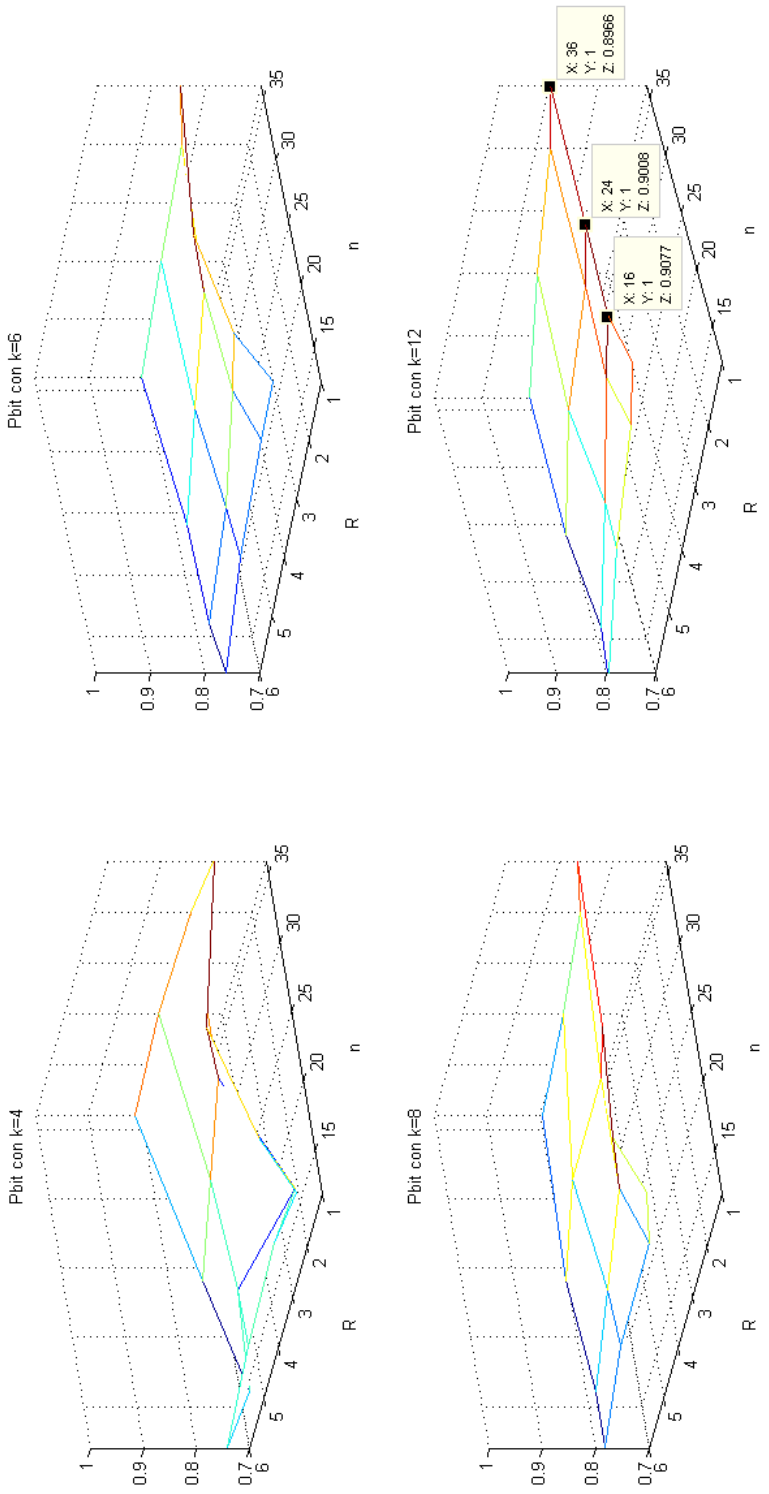

Figura 5.2: Percentuale di bit corretti sul totale di bit previsti al variare di k, n, R.

aspettati è il fatto che il miglior comportamento del sistema si registri con una scelta "estrema" per R e non con un valore intermedio, come avremmo invece pensato. Tutti i migliori risultati sono stati ottenuti con un valore di $R = 1$; si osservi che con questa configurazione, sono considerate match per un motif $\boldsymbol{c}$ solamente sottosequenze identiche a $\boldsymbol{c}$ stessa e che si trovano shiftate al massimo di un campione, oppure sottosequenze con al massimo un bit di differenza e che si trovano nella stessa posizione relativa.

Cerchiamo di capire meglio quali potrebbero essere le cause che portano ad un risultato così sorprendente. Crediamo che, per interpretare correttamente questi risultati, per prima cosa sia necessario mettersi d'accordo su quale parametro bisognerebbe valutare per misurare l'affidabilità del sistema. Durante questa prova, il valore fl che indica la lunghezza, in campioni, del periodo di forecast è rimasto fisso a 4. Considerando che il numero totale di sottosequenze elaborate durante la working phase è pari a $m1 - n - fl + 1$, allora il numero *massimo* di bit previsti sarà pari a $(m1 - n - fl + 1) \cdot fl$, ma il numero *reale* sarà inferiore, poiché i bit previsti per ogni sottosequenza potrebbero essere meno di fl per l'impossibilità di avere un forecast aggiornato, fino al caso estremo in cui non viene fatta nessuna previsione, seguendo il meccanismo esposto in sezione 4.4. La percentuale riportata nel grafico rappresenta il numero di bit corretti sul totale di bit previsti per le varie sottosequenze; dobbiamo riflettere sul fatto se questo sia effettivamente un buon parametro per valutare il sistema.

Infatti, si può obiettare che un sistema con ottime prestazioni sulla percentuale di bit corretti sul totale di quelli indovinati non sia necessariamente un buon sistema: non lo sarebbe se, ad esempio, esso fosse impossibilitato a fornire previsioni per una buona parte delle sottosequenze che si presentano. Cerchiamo di inquadrare meglio il problema: l'algoritmo con $R = 1$

effettivamente mostra di indovinare 9 su 10 dei bit che prevede; ma le sue caratteristiche lo portano a tentare di indovinare un numero di bit significativamente minore rispetto alle altre versioni? In altre parole, le buone performance della versione con $R = 1$ sono dovute al fatto che esso sia essenzialmente poco "coraggioso", evitando di fornire previsioni in situazioni dove, magari, gli altri algoritmi sbagliano? Come vedremo fra poco, la risposta a questa domanda è sostanzialmente affermativa. Vogliamo sottolineare che questo, a nostro avviso, non è necessariamente un punto debole! Come abbiamo avuto modo di sostenere in altri punti del nostro lavoro, una qualità di un sistema di pattern recognition deve essere quella di saper riconoscere sottosequenze anomale, trattandole in maniera appropriata; può essere saggio evitare di fare previsioni quando si incontra una sottosequenza dalle caratteristiche nuove, della quale non si hanno informazioni nella base di conoscenza accumulata finora dal sistema. Sebbene siamo convinti della bontà di questo approccio, riteniamo anche che un buon sistema di pattern recognition e prediction debba raggiungere un compromesso ragionevole fra la quota di bit indovinati e la frazione di sottosequenze della quale è in grado di fare una previsione; obiettivamente, un sistema che indovini anche il 100% dei bit ma che si sbilanci a prevedere una sottosequenza su dieci è di scarsa utilità.

Il ragionamento precedente può aiutarci a capire una delle possibili cause del successo dell'algoritmo con $R = 1$, e ci suggerisce un modo alternativo per valutare la bontà del nostro sistema, considerando il numero di bit corretti in rapporto al numero totale di bit indovinabili. In altre parole,

$$P'_{bitok} = \frac{\#\ bit\ corretti}{(m1 - n - fl + 1) \cdot fl} \tag{5.1}$$

Vogliamo ribadire che, a nostro avviso, questo deve considerarsi un metodo *complementare* al precedente per la valutazione complessiva del sistema, la quale deve giungere da un compromesso fra le due esigenze opposte; in altre parole, non bisogna considerare questa misura come un indice assoluto della bontà del sistema poiché, specialmente di fronte a sequenze imprevedibili e assolutamente nuove, la miglior risposta sarebbe proprio quella di astenersi dal fare una previsione; questo avrebbe l'effetto di diminuire il denominatore nella 5.1, avvicinandoci progressivamente ai valori espressi nel primo grafico. In figura 5.3 si possono trovare i grafici relativi alla percentuale di bit corretti sul totale di bit teoricamente prevedibili, al variare di k, n ed R.

Possiamo vedere come l'aspetto del grafico vari radicalmente, suggerendoci un punto di vista differente nella valutazione del sistema. Vediamo alcuni risultati che emergono da questi nuovi dati:

- configurazioni che sembravano molto promettenti si rivelano essere scadenti se si considera il numero di sottosequenze sulle quali si astengono (ad esempio la configurazione $R = 1, n = 36, k = 12$ che passa dall'89,7% al 58%); configurazioni di questo tipo sono caratterizzate da un atteggiamento molto prudente, dettato dalla difficoltà di associare le sottosequenze ai motif individuati: questo fa sì che il sistema in molti casi non possa fare delle previsioni, riducendo sostanzialmente la sua efficacia. Questo genere di configurazioni era proprio quello che intendevamo "smascherare" con questo nuovo indice di valutazione, mostrando che i buoni risultati da essi ottenuti non erano indice di un *vero* buon funzionamento dell'algoritmo;

- altre, fra quelle che erano risultate le migliori, si assestano su un livello accettabile (ad esempio le configurazioni $R = 1$, $n = 12$, $k = 12$ e $R = 1, n = 16, k = 12$ che passano dal 90% all'80% e al 77% rispetti-

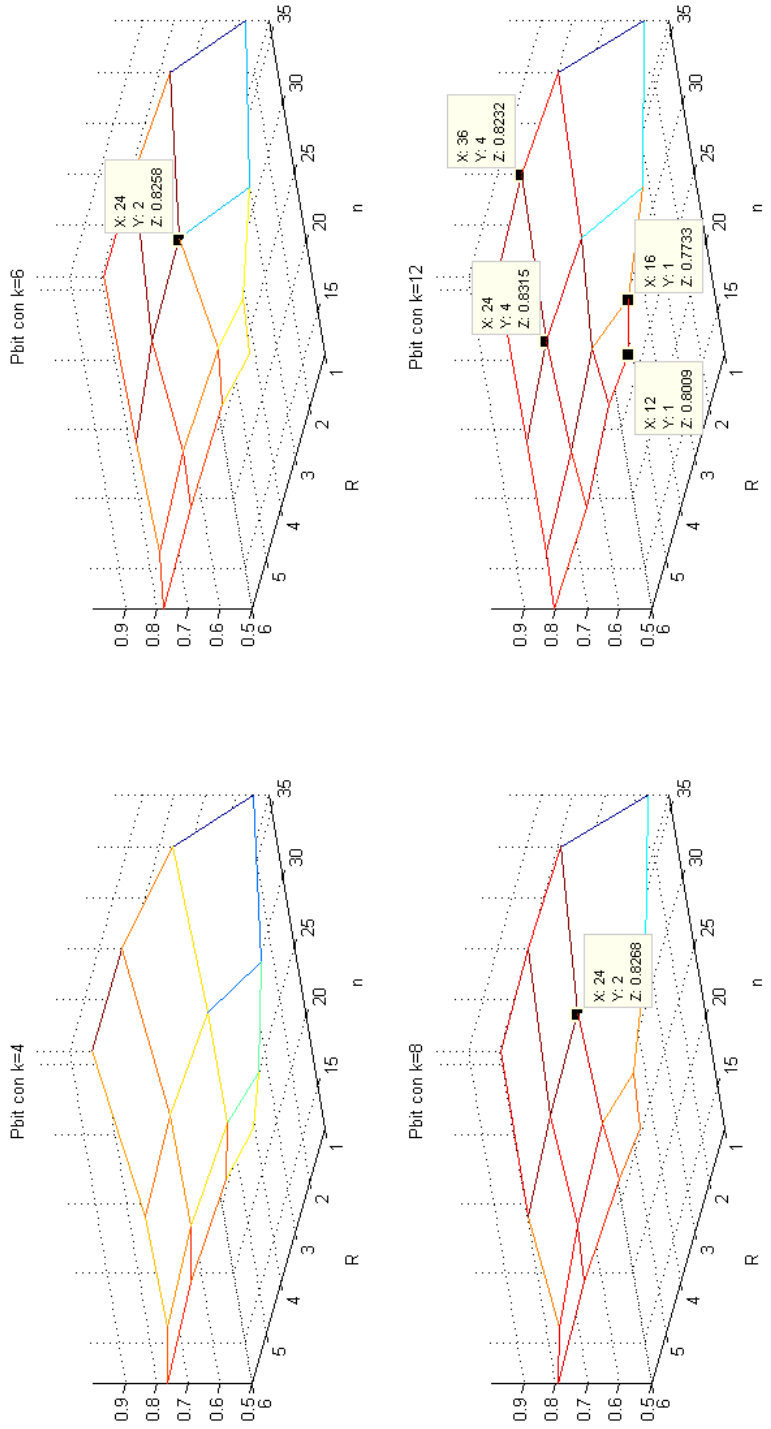

Figura 5.3: Percentuale di bit corretti sul totale di bit teoricamente prevedibili al variare di k, n, R.

vamente); ricordando che è necessario valutare congiuntamente i due indici, riteniamo che questo tipo di configurazione sia fra i migliori fra quelli proposti. Riteniamo che sia un buon risultato quello di un sistema che, quando da una previsione, la da corretta nel 90% dei casi, e che comunque riesce a dare una previsione corretta nell'80% dei casi possibili; la capacità del sistema di sapersi "arrendere" di fronte a situazioni impreviste, infatti, può essere sfruttata con profitto a più alto livello: ad esempio prendendo la decisione che provoca meno disagio per l'utente (nel caso di un sistema di riscaldamento, si può decidere ad esempio di accendere comunque l'impianto in caso di dubbio), oppure prendendo una decisione di default che può essere impostata dall'utente stesso;

- infine, alcune configurazioni che mostravano già buoni risultati rimangono sostanzialmente invariate, finendo per risultare le migliori con questo indice di valutazione (ad esempio le configurazioni $R = 2, n = 24$, $R = 4, n = 24$ oppure $R = 4, n = 36$ con $k = 8, 12$ che mostrano risultati attorno all'83%); assieme a quelle del punto precedente, riteniamo essere queste configurazioni le migliori possibili per il nostro sistema. Rispetto alle precedenti, queste mostrano un comportamento più equilibrato, dovuto essenzialmente ad una scelta meno restrittiva per il valore di R; questo si traduce in una capacità del sistema di reagire a quasi tutte le sottosequenze che si presentano in ingresso, ma d'altro canto diminuisce la sua capacità di individuare una sottosequenza d'ingresso anomala, finendo per provare a prevedere quasi tutte le sottosequenze, in tal modo abbassando la sua percentuale di bit corretti sul totale di bit previsti.

Finora abbiamo analizzato i dati che abbiamo ottenuto, scoprendo alcu-

ne configurazioni più promettenti di altre per assicurare buone prestazioni al nostro algoritmo. Un grosso limite di questo approccio è che i dati sono stati ottenuti in modo empirico, e non sappiamo se le configurazioni migliori per le particolari sequenze analizzate possano rivelarsi tali anche per altri casi. Ora, dunque, cercheremo di analizzare i motivi per cui alcune configurazioni si comportano meglio di altre, provando a formulare delle regole generali riguardo i valori assegnati ai diversi parametri per ottenere dei buoni risultati.

Per prima cosa, vogliamo soffermarci ad analizzare il numero totale di sottosequenze non riconosciute per ciascuna configurazione: a tale proposito, si faccia riferimento a figura 5.4, dove si vede tale quantità al variare dei parametri R ed n; abbiamo rimosso la dipendenza da k poiché questo parametro non entra in gioco nel modo in cui la sottosequenza viene associata al motif. Per ogni grafico si vede il numero totale di sottosequenze non riconosciute e il dettaglio delle cause che hanno provocato il mancato riconoscimento, che analizzeremo più tardi in questa sezione.

Si può osservare come il numero di sottosequenze non riconosciute aumenti costantemente con l'aumentare della lunghezza n delle sottosequenze, mentre diminuisca con l'aumentare del raggio R. Questo risultato non è affatto sorprendente e si può facilmente spiegare pensando ai motif come a delle sfere che occupano lo spazio $(n+1)$-dimensionale: maggiore è il raggio R delle sfere, maggiore sarà la porzione di spazio occupata dai K motif; al contrario, aumentando n possiamo immaginare di allargare lo spazio in cui si collocano le sfere. Possiamo ipotizzare che la probabilità per una sottosequenza di essere match per uno qualsiasi dei motif individuati sia pari alla frazione di spazio occupato dalle K sfere, in rapporto allo spazio complessivo. Si può quindi facilmente spiegare il pessimo risultato della configurazione con

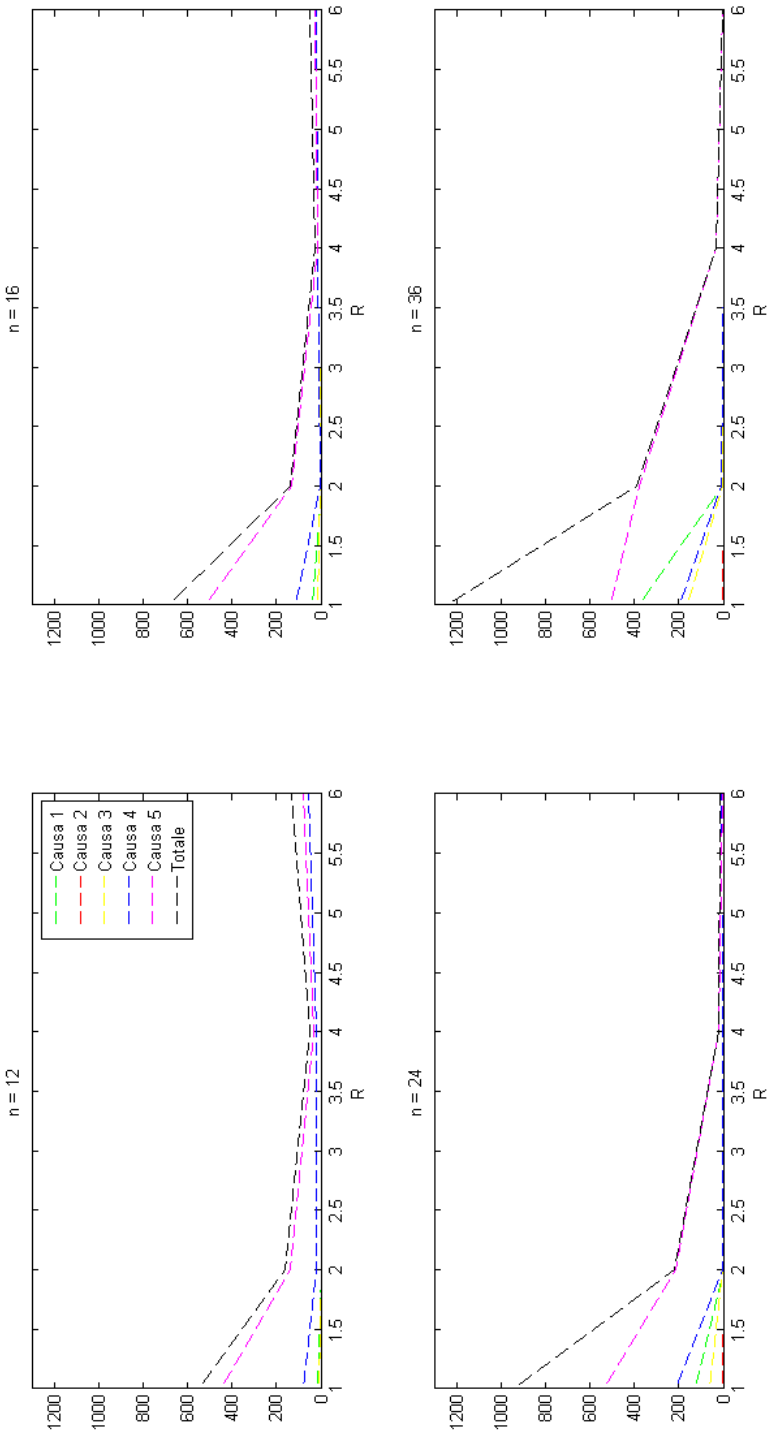

Figura 5.4: Numero di sottosequenze non riconosciute e relative cause, al variare di n ed R.

$R = 1, n = 36$: con questa configurazione le K sfere di raggio così ridotto non riescono ad occupare che una minima parte dello spazio $(n+1)$-dimensionale, diminuendo drasticamente la probabilità di un match.

Possiamo dunque trarre una relazione diretta fra il "riempimento" dello spazio $n + 1$-dimensionale e la bontà delle prestazioni del nostro sistema? Come è possibile stimare questo fattore in maniera sensata per la nostra applicazione? Abbiamo pensato di considerare, per ogni esecuzione dell'algoritmo, il numero di sottosequenze adatte a diventare motif durante la ricerca dell'ultimo motif (il K-motif). Ricordiamo che, perché una sottosequenza possa essere motif, essa deve avere una distanza $> 2R$ da tutti gli altri motif (in modo che ci sia spazio per collocare una nuova sfera in corrispondenza di essa); dunque il numero di sottosequenze candidabili durante la ricerca dell'ultimo motif ci dà una buona stima del tasso di riempimento dello spazio $(n + 1)$-dimensionale nella zona occupata dalle sottosequenze.[1] In figura 5.5 possiamo trovare il tasso di riempimento per diversi valori di R ed n.

Il grafico mostra una caratteristica comune molto importante fra le configurazioni caratterizzate dalle prestazioni migliori. Tre fra le configurazioni più promettenti, $(R = 1, n = 12)$, $(R = 1, n = 16)$ e $(R = 2, n = 24)$ mostrano un tasso di riempimento molto simile, con un numero di sottosequenze ancora eleggibile a motif variabile fra il 4% e l'8% del numero totale di sottosequenze. Le altre configurazioni con prestazioni accettabili oscillano con valori poco superiori od inferiori a questo range. Dunque si può pensare ad un metodo per affinare il nostro algoritmo: invece di tenere fisso il numero di motif K, è possibile iterare la ricerca dei motif finché il tasso di

[1] Il tasso di riempimento dell'intero spazio $(n + 1)$-dimensionale è ridicolmente più basso, avendo esso una cardinalità di $2^n \cdot 96$ punti; il fatto è che esso è per gran parte vuoto, mentre la maggior parte delle sottosequenze (e, conseguentemente, delle sfere dei motif) si concentra in alcune aree ristrette dello spazio. Per questo parliamo di "tasso di riempimento dello spazio $(n + 1)$-dimensionale *nella zona occupata dalle sottosequenze*", che è ciò che realmente ci interessa stimare.

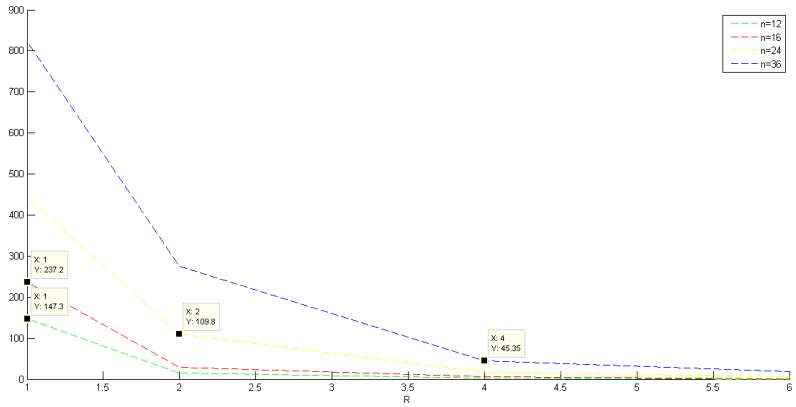

Figura 5.5: Stima del tasso di riempimento dello spazio $(n+1)$-dimensionale, nella zona occupata dalle sottosequenze

riempimento non si avvicina a questa soglia. Questo metodo consentirebbe inoltre di rendere il sistema adattabile a sequenze di ingresso di ogni tipo, senza dover condurre preliminarmente un'analisi per verificare quali siano i parametri più adatti.

Soffermiamoci ora sull'incidenza delle diverse cause che provocano la mancata identificazione, al variare delle configurazioni; ricordiamo che i possibili motivi che portano a questo punto possono essere 5, secondo quanto descritto in dettaglio nella sezione 4.4 e che riportiamo qui per completezza:

1. $\boldsymbol{c}$ non collide con nessun motif ma il motif a peso minimo $\boldsymbol{mwm}$, di cui dovrebbe prendere il posto, ha peso maggiore;

2. $\boldsymbol{c}$ non collide con nessun motif ma il motif a peso minimo $\boldsymbol{mwm}$, di cui dovrebbe prendere il posto, ha peso con rapporto superiore a $1/2K$ rispetto al totale dei pesi;

3. $\boldsymbol{c}$ collide con un motif $\boldsymbol{coll}$, ma non può prenderne il posto poiché esso ha peso maggiore;

4. $\boldsymbol{c}$ collide con un motif ***coll***, ma non può prenderne il posto poiché esso ha peso con rapporto superiore a $1/2k$ rispetto al totale dei pesi;

5. $\boldsymbol{c}$ collide con due o più motif già esistenti.

Tornando a riferirci alla figura 5.4, possiamo fare alcune ulteriori considerazioni alla luce di quanto abbiamo scoperto riguardo il tasso di riempimento dello spazio:

- la causa numero 1, come ci aspettavamo, è generalmente poco significativa e mostra una qualche incidenza solamente in configurazioni con un tasso di riempimento molto basso;

- la causa numero 2, come avevamo già anticipato in sezione 4.4, è praticamente ininfluente sul funzionamento del sistema a causa della sua estrema rarità;

- le cause numero 3 e 4 numericamente incidono in maniera simile; esse costituiscono circa il 50% sul totale di sequenze non identificate per configurazioni con tasso di riempimento molto alto (si vedano ad esempio le configurazioni $R = 6, n = 12$ oppure $R = 6, n = 16$). Osserviamo anche che la causa numero 4, a differenza della 2, ha un'incidenza non trascurabile;

- la causa numero 5 è, per ogni configurazione, quella che causa il maggior numero di mancate identificazioni; la sua incidenza va dal quasi 100% nelle configurazioni ad alto tasso di riempimento, fino a sotto il 50% in quelle a basso riempimento (si veda ad esempio $R = 1, n = 36$). La sua grande rilevanza suggerisce che la gestione dei casi di doppia collisione è, obiettivamente, uno dei punti deboli del sistema, che dovrebbe essere oggetto di qualche miglioramento.

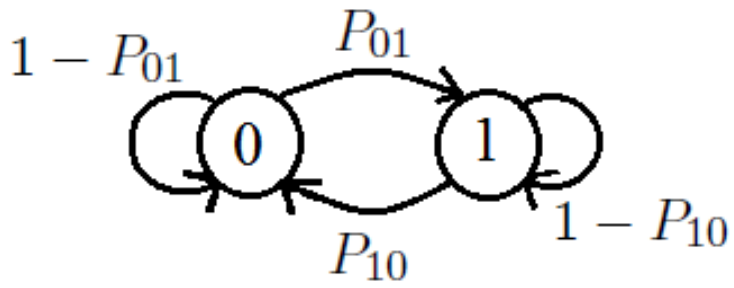

Figura 5.6: Catena di Markov generica per modellare una sequenza di bit errati/corretti

Veniamo ora ad analizzare un altro aspetto delle prestazioni del sistema. Volevamo capire se la distribuzione degli errori nelle diverse previsioni potesse essere inquadrata come una variabile casuale con distribuzione IID (*Independent and Identically Distribuited*). Generalmente, la successione di bit errati e bit corretti si può inquadrare all'interno di una catena di Markov del tipo mostrato in figura 5.6, dove la permanenza nello stato 0 significa "bit corretto" mentre lo stato 1 significa "bit errato".

Generalmente, una catena di Markov siffatta non dà luogo ad una sequenza dove i bit a 1 (errati) sono disposti secondo una distribuzione IID; questo si verifica se e solo se le due probabilità di transizione fra i due stati sono complementari fra loro:

$$P_{01} = 1 - P_{10} = \epsilon \tag{5.2}$$

e, di conseguenza

$$P_{10} = 1 - P_{01} = 1 - \epsilon \tag{5.3}$$

In questo caso effettivamente si verifica che i bit errati sono disposti nella sequenza risultante secondo una distribuzione IID con probabilità uguale a ϵ; se calcoliamo la durata media della permanenza nello stato 1, secondo le

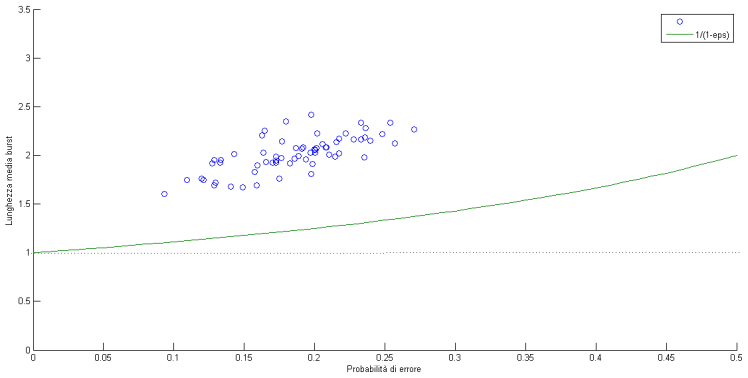

Figura 5.7: Lunghezza media dei burst di errore in relazione alla probabilità media di errore

proprietà delle catene di Markov, otteniamo che

$$L = \frac{1}{P_{10}} \tag{5.4}$$

Dunque, procedendo a ritroso, abbiamo fatto questo esperimento: abbiamo analizzato tutte le sequenze di forecast fornite dall'algoritmo, cercando la lunghezza di ogni *burst* di errori (un burst è una sequenza di uno o più errori consecutivi); a questo punto abbiamo calcolato la lunghezza media di un burst di errori, confrontandola con la probabilità media di errore per quella particolare configurazione; questo allo scopo di verificare se gli errori si verificano nelle nostre previsioni secondo una distribuzione indipendente, oppure se sussiste una certa correlazione fra le probabilità $P[bit\ i\ errato]$ e $P[bit\ i+1\ errato]$. In quest'ultimo caso, sicuramente la lunghezza media di un burst di errori sarà maggiore di quella prevista dall'equazione 5.4. In figura 5.7 troviamo i risultati di questa prova per le diverse configurazioni.

Come si può vedere, la lunghezza media del burst è superiore a quella nel caso di errori indipendenti; il risultato corrisponde alle nostre aspettative, poiché è la tipologia stessa del sistema, e delle previsioni che andiamo ad

effettuare, a rendere probabili sequenze di bit errati. Si pensi ad un caso banale come può essere il rientro a casa con un largo anticipo rispetto all'orario previsto; mentre la previsione per quella sottosequenza era con tutti i bit a 0 (assenza per il prossimo campione e per i successivi $fl - 1$), il rientro e la permanenza a casa fanno sì che tutti i bit di quella previsione risultino errati. Questo, evidentemente, incide in maniera pesante sulla lunghezza media del burst, e ci conferma che i bit errati di una sequenza di forecast non sono distribuiti in maniera IID.

5.3.3 Adattamento all'evolversi delle condizioni

Abbiamo voluto infine testare la capacità del sistema di reagire ad una variazione delle condizioni ambientali. L'algoritmo è stato pensato, tramite il meccanismo dei motif pesati e della loro sostituzione, per adattarsi lentamente e ininterrottamente alle variazioni esterne, che si manifestano nel nostro sistema sotto forma di sottosequenze "anomale". In questo modo, il modello che viene costruito all'atto della learning phase non è fisso, ma evolve continuamente grazie alle informazioni che arrivano dalle nuove sottosequenze della working phase. Chiaramente, si vuole che il sistema abbia una certa reattività, e che sia in grado di adattare velocemente il modello alle nuove informazioni; allo stesso tempo, però, questo processo di adattamento non può essere troppo veloce, altrimenti si corre il rischio di rovinare informazioni consolidate sulla base di avvenimenti estemporanei. Nel nostro sistema, la velocità di adattamento è dettata fondamentalmente dal parametro ED, che gioca un ruolo determinante nel calcolo dei pesi dei motif.

Abbiamo dunque provato a fare la seguente simulazione. Per prima cosa, abbiamo condotto la learning phase utilizzando la sequenza letta dal rilevatore nell'abitazione, con la tecnica consueta; successivamente, abbiamo

simulato un improvviso spostamento del sensore in un ambiente caratterizzato da sequenze di occupazione completamente diverse, come un ufficio: per fare ciò, abbiamo costruito la sequenza della working phase utilizzando Office Simulator. In questo modo, facendo la learning phase e la working phase con due sequenze temporali dalle caratteristiche diametralmente opposte, vogliamo mettere alla prova il nostro sistema vedendo dopo quanti giorni della working phase le prestazioni ritornano ad un livello accettabile. Questo è chiaramente un esperimento estremo e qualcosa che, generalmente, non ha alcun senso fare: i cambiamenti che si verificano nell'occupazione di uno stesso locale, con l'evolversi delle abitudini, o delle stagioni, sono generalmente molto meno drastici. Se proprio fosse necessario spostare il sensore da un locale ad un altro con caratteristiche così differenti, probabilmente avrebbe più senso resettarlo e farlo partire da zero, in quanto le informazioni accumulate durante la learning phase sono così sbagliate che si avrebbero risultati migliori tirando a caso. In questo senso, il nostro esperimento è una sorta di "worst case" per la capacità di adattamento del sistema: durante le situazioni reali, ci aspettiamo che l'algoritmo reagisca più velocemente e in maniera migliore.

In figura 5.8 si può vedere il grafico della media mobile a 5 giorni della percentuale di bit corretti: in altre parole, il valore indicato per il giorno d indica i bit indovinati in quel giorno e nei 4 precedenti; questo meccanismo serve a rendere i dati più smoothed, neutralizzando le forti oscillazioni che si verificano nella percentuale di bit corretti di giorno in giorno ed aiutando ad identificare un trend più generale. Possiamo vedere chiaramente come, dopo un avvio decisamente pessimo, l'algoritmo poco per volta riesca ad aggiustare il tiro, adattando il modello alle nuove informazioni provenienti dalla sequenza di working phase. Durante i primi dieci giorni, come ci aspettava-

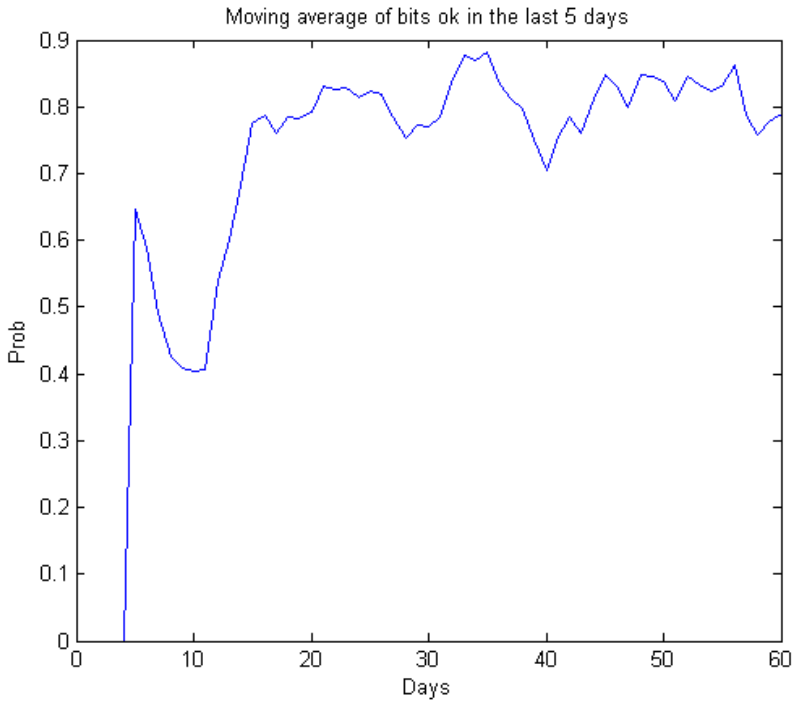

Figura 5.8: Media mobile a 5 giorni della percentuale di bit corretti, a partire dall'inizio della working phase

mo, il modello costruito durante la learning phase non è di alcun aiuto nel forecast, producendo risultati comparabili a quelli che si otterrebbero senza nessuna informazione preliminare. Però, dopo circa venti giorni di working phase con la nuova sequenza, si può già vedere come i risultati inizino a farsi accettabili: segno che i meccanismi per gli aggiornamenti dei motif e dei cluster stanno funzionando a dovere. Alla fine del periodo di prova, dopo sessanta giorni di simulazione con la nuova sequenza (un periodo equivalente a due volte ED), le percentuali di bit corretti sono del tutto comparabili a quelle che si sono ottenute con gli altri esperimenti, dunque possiamo dirci molto soddisfatti di questo particolare risultato.

Capitolo 6

Conclusioni

Possiamo sostenere di aver raggiunto buona parte degli obiettivi che ci eravamo proposti, con risultati che sono di gran lunga migliori di quelli che ci saremmo aspettati.

Siamo riusciti a progettare ed implementare un sistema completamente autonomo che, senza alcun intervento esterno o programmazione preventiva, riesce a costruire un modello della serie temporale di occupazione di un locale, mediante il clustering di alcune sottosequenze della serie stessa; dal momento che non ha senso fare il clustering di tutte le sottosequenze di una serie temporale, abbiamo introdotto il concetto di K-motif, cioè delle K sottosequenze più ricorrenti, che abbiamo poi utilizzato per fare la classificazione.

Successivamente, il modello costruito viene utilizzato con successo per prevedere l'occupazione del locale nel futuro prossimo, con risultati che arrivano fino al 90% di campioni previsti correttamente sul totale dei campioni previsti. Il sistema è stato programmato per tenere aggiornato il modello sulla base delle informazioni ricavate dalle nuove sottosequenze, in modo che esso possa evolvere anche una volta terminata la fase di apprendimento. Messo alla prova su questo particolare aspetto, tramite un cambiamento ra-

dicale delle sequenze in ingresso, il sistema ha mostrato di comportarsi bene, ricostruendo progressivamente la base di conoscenza attraverso le nuove sottosequenze, e raggiungendo in tempi accettabili prestazioni vicine a quelle ottimali.

Infine, abbiamo osservato che alcuni dei parametri che caratterizzano il sistema hanno un impatto importante sulle prestazioni del sistema stesso; abbiamo quindi individuato le motivazioni che provocano queste variazioni nelle performance, suggerendo alcuni metodi per il settaggio corretto di questi parametri.

Durante l'analisi delle prestazioni del sistema, sono emersi alcuni aspetti che permetterebbero di migliorare ulteriormente il funzionamento dell'algoritmo; ad esempio sarebbe auspicabile un meccanismo che consenta l'*autotuning* dei parametri fondamentali, ed una migliore gestione del caso in cui una sottosequenza collide con due motif già esistenti. Alcuni metodi per sviluppare questi accorgimenti sono stati accennati nel corso della trattazione, e potranno essere oggetto di lavoro nel futuro.

Bibliografia

[1] C. M. Bishop, *Pattern Recognition and Machine Learning.* Springer, 2006.

[2] D. J. C. MacKay, *Information Theory, Inference, and Learning Algorithms.* Cambridge, UK: Cambridge University Press, 2003.

[3] T. W. Liao, "Clustering of time series data - a survey," in *Pattern Recognition*, vol. 38, 2005, pp. 1857–1874.

[4] S. Lloyd, "Least square quantization in pcl," in *IEEE Transactions on Information Theory*, vol. 28, Mar. 1982, pp. 129–137.

[5] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, Berkeley, CA, USA, 1967, pp. 281–297.

[6] G. Das, K. Lin, H. Mannila, G. Renganathan, and P. Smyth, "Rule discovery from time series," in *Proceedings of the 4th Int'l Conference on Knowledge Discovery and Data Mining*, New York, NY, USA, Aug. 1998, pp. 16–22.

[7] J. Lin, E. J. Keogh, and W. Truppel, "Clustering of streaming time series is meaningless," in *Proceedings of the 8th ACM SIGMOD workshop on*

Research issues in data mining and knowledge discovery, San Diego, CA, USA, 2003, pp. 56–65.

[8] J. Lin, E. J. Keogh, S. Lonardi, and P. Patel, "Finding motifs in time series," in *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, July 2002.

[9] G. Hamerly and C. Elkan, "Alternatives to the k-means algorithm that find better clusterings," in *Proceedings of the eleventh international conference on Information and knowledge management*, 2002, pp. 600–607.