Università degli Studi di Padova
Dipartimento di Scienze Statistiche
Corso di Laurea Triennale in

Statistica e Gestione delle Imprese

RELAZIONE FINALE

DATA MINING PER SERIE STORICHE

Relatore Prof. Luisa Bisaglia
Dipartimento di Scienze Statistiche

Laureanda: Chiara Gennaro
Matricola: 1006241

Anno Accademico 2012/2013

Indice

Introduzione

L'uso di strumenti di raccolta automatica unito alla maturità della tecnologia dei database ha portato ad avere collezioni di informazione di grandi dimensioni. Avere molti dati è un vantaggio, ma ciò rende anche molto complicata la loro gestione e per utilizzarli occorrono strumenti sempre più sofisticati.

Quindi, a causa di questa vasta disponibilità di enormi quantità di dati e della necessità di trasformare questi dati in informazioni e conoscenze utili per ampie applicazioni, il *data mining* ha attirato una grande attenzione nel settore dell'informazione negli ultimi anni.

Il *data mining* (letteralmente: "estrazione di dati") è il processo di scoperta di conoscenze interessanti, quali i *pattern*, le associazioni, i cambiamenti, le anomalie e le strutture significative, da una grande quantità di dati memorizzati nei database. In altre parole, con il termine *data mining* si intende l'applicazione di una o più tecniche che consentono l'esplorazione di grandi quantità di dati, con l'obiettivo di individuare le informazioni più significative e di renderle disponibili e utilizzabili. Si noti che i concetti di informazione e di significato sono legati strettamente al dominio applicativo in cui si esegue il *data mining*; in altri termini, un dato può essere interessante o trascurabile a seconda del tipo di applicazione in cui si vuole operare.

I dati di serie storiche rappresentato una percentuale sempre più grande della disponibilità totale di dati. Data l'ubiquità delle serie temporali e data l'esponenziale crescita delle dimensioni dei database, c'è stata di recente un'esplosione di interesse per il *data mining* delle serie storiche.

Nel contesto del *data mining* lo studio delle serie temporali permette di trovare la rappresentazione più efficiente relativa ai dati in oggetto, effettuare misure di similarità tra serie storiche e raggruppare e classificare i dati.

Tuttavia, il più evidente problema del *data mining* per le serie storiche nasce proprio dall'elevata dimensionalità dei dati. Con la rapida crescita delle fonti di informazioni digitali, gli algoritmi di *data mining* devono confrontare dataset sempre più massicci.

Si noti che il vero significato dei termini "simile a" o " formazione di gruppi", normalmente intuitivi, diventano poco chiari in uno spazio ad elevata dimensionalità. Il

motivo è che, quando aumenta la dimensionalità, tutti gli oggetti diventano essenzialmente equidistanti tra loro, e quindi la classificazione e il *clustering* (due temi molto importanti del *data mining*) perdono il loro significato. Questo problema è noto come "*curse of dimensionality"*. L'idea chiave che permette un significativo *data mining* per le serie storiche è che, anche se la dimensionalità originale dei dati potrebbe essere alta, la dimensionalità "*intrinseca"* è tipicamente minore. Per questo motivo, praticamente tutti gli algoritmi del *data mining* per le serie storiche evitano di operare sui dati "grezzi" originali.
Al fine di superare questo ostacolo, sono state introdotte tre componenti di implementazione che rappresentano gli aspetti fondamentali del *data mining* per le serie storiche:

- **Metodi di rappresentazione**: Una tecnica di rappresentazione riduce la dimensionalità dei dati mantenendo però le caratteristiche essenziali della serie storica.
- **Misure di similarità**: Una misura di similarità permette di riconoscere oggetti simili anche se essi non sono matematicamente identici.
- **Tecniche di indicizzazione**: La tecnica di indicizzazione permette il minimo consumo di spazio e di complessità computazionale.

Quindi, a causa dell'elevata dimensionalità delle serie storiche, è essenziale ideare rappresentazioni a più bassa dimensionalità che mantengano le caratteristiche fondamentali di una serie. Dato uno schema di rappresentazione, la distanza tra serie storiche necessita di essere attentamente definita al fine di mostrare aspetti rilevanti della similarità sottostante. Infine, lo schema di indicizzazione deve consentire una gestione e un'interrogazione efficiente dei sempre più grandi dataset.
In questo elaborato andremo quindi, prima di tutto, a descrivere questi aspetti fondamentali del *data mining* per le serie storiche, cioè i metodi di rappresentazione (Capitolo 1), le misure di similarità (Capitolo 2) e le tecniche di indicizzazione (Capitolo 3).
Successivamente (Capito 4) andremo ad illustrare alcuni degli obiettivi del *data mining*, quali *query by content, classification, clustering, anomaly detection* e *motif discovery*, trattando alcuni loro algoritmi tra i più comuni.

Capitolo 1- Metodi di rappresentazione

Le serie storiche sono essenzialmente dati ad alta dimensionalità. Definire algoritmi che lavorino direttamente sulla serie storica grezza potrebbe essere perciò computazionalmente troppo costoso.

La motivazione principale delle rappresentazioni è quindi quella di enfatizzare le caratteristiche essenziali dei dati in modo conciso. I benefici aggiunti ottenuti sono una memoria efficiente, un'accelerazione del processo e la rimozione del rumore (*noise*) implicito.

Queste proprietà basilari conducono ai seguenti requisiti per ogni rappresentazione (Esling e Agon, 2012):

- Significativa riduzione della dimensionalità dei dati;
- Enfasi sulle caratteristiche di forma fondamentali;
- Basso costo computazionale per elaborare la rappresentazione;
- Insensibilità al rumore o gestione implicita del rumore.

Esistono molti metodi di rappresentazione delle serie storiche, ognuno dei quali è proposto con l'obiettivo di sostenere la ricerca di similarità e i *tasks* del *data mining*.

È possibile effettuare una classificazione delle tecniche maggiori (Wang et al., 2012; Ratanamahatana et al., 2010) organizzata in maniera gerarchica, come si può vedere in Figura 1. Individuiamo così due categorie base:

- *DATA ADAPTIVE representations*: in questa categoria, una rappresentazione comune sarà scelta per tutti gli elementi del database che minimizzano l'errore di ricostruzione globale.
- *NON-DATA ADAPTIVE representations*: al contrario, questi metodi valutano le proprietà locali dei dati e di conseguenza costruiscono una rappresentazione approssimata.

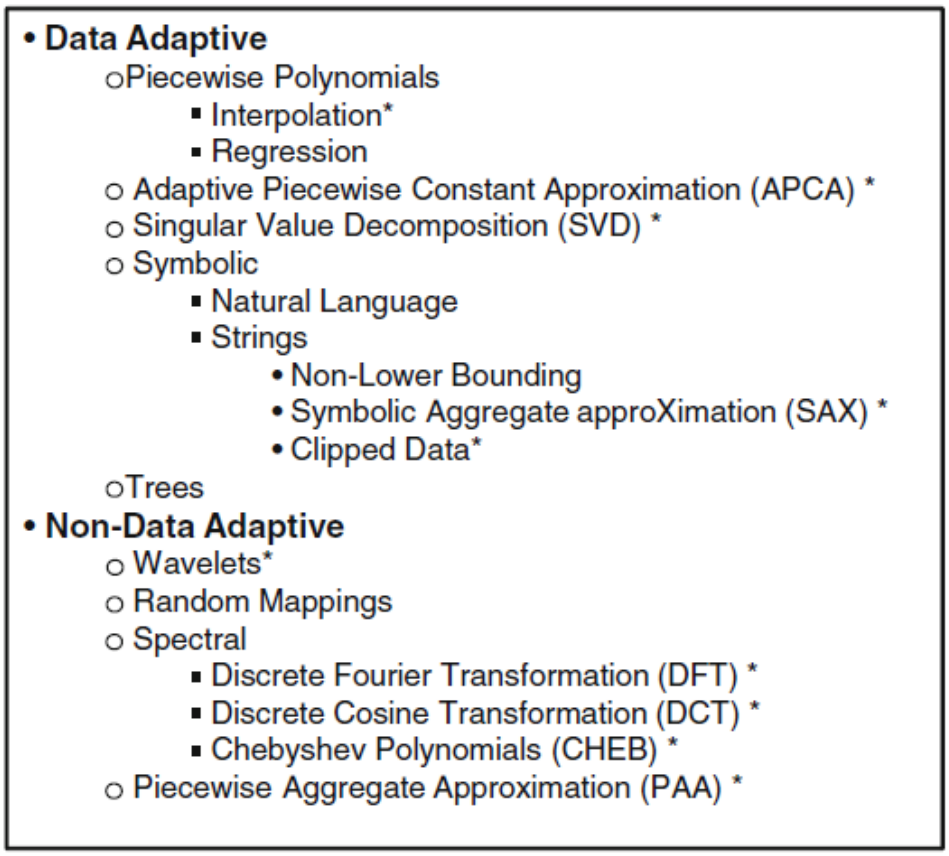


Figura 1: Classificazione dei metodi di rappresentazione più comuni raggruppati in due categorie: *Data Adaptive* e *Non-Data Adaptive*.

Per esempio, *la Adaptive Piecewise constant Approximation* (APCA, una tecnica *Adaptive*) trasforma ogni serie storica in un insieme di segmenti di valori costanti di lunghezza diversa tali che i loro errori di ricostruzione siano minimi.

Dall'altra parte la *Piecewise Aggregate Approximation* (PAA, una tecnica *Non-Adaptive*), approssima una serie storica dividendola in segmenti di ugual misura e registrando il valore medio dei punti che cadono all'interno del segmento.

Le rappresentazioni che presentano l'asterisco (*) nella Figura 1 hanno la proprietà di permettere il *lower bounding*. Quest'ultima, essenzialmente, permette di definire una misura di distanza che può essere applicata alle rappresentazioni in scala (cioè compresse) della serie storica corrispondente, la quale è garantita per essere inferiore o uguale alla "vera" distanza che si misura sui dati grezzi. Supponiamo quindi di avere due serie storiche Q e C di dimensione *m* nello spazio originale. La distanza euclidea *D(Q,C)* è la "vera" funzione della misura di distanza. Le serie storiche Q' e C' sono le nuove versioni

delle serie originarie e *LB_D(Q' ,C')* è la nuova funzione di distanza nel *low space*. Le 2 funzioni, illustrate in Figura 2, dovrebbero essere: $D(Q,C) \geq LB\_D(Q',C')$.

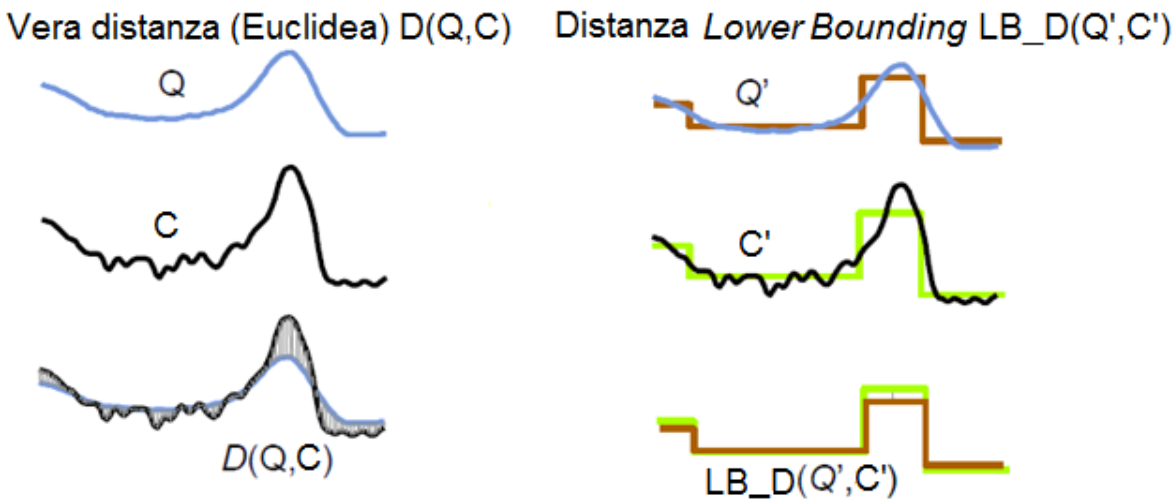


Figura 2: Illustrazione della differenza fra la "vera" distanza D(Q,C) e la distanza *Lower Bounding* LB_D(Q',C').

In questo elaborato andremo ad approfondire solo alcune delle rappresentazioni elencate nella Figura 1, precisamente la *Discrete Fourier Trasform (DFT), la Discrete Wavelet Trasform (DWT), la Piecewise Aggregate Approximation (PAA), l'Adaptive Piecewise constant Approximation (APCA) e la Symbolic Aggregate approXimation (SAX).*

1.1 Discrete Fourier Trasform

Tradizionalmente, ci sono due strade per analizzare i dati delle serie storiche: l'analisi nel dominio del tempo e l'analisi nel dominio della frequenza.

L'analisi nel dominio temporale esamina come una serie storica si evolve nel tempo, con strumenti come la funzione di autocorrelazione.

L'analisi nel dominio di frequenza, anche conosciuta come analisi spettrale, studia come componenti periodiche a diverse frequenze descrivono l'evoluzione di una serie storica. In molte applicazioni è conveniente trasformare una funzione con una combinazione lineare di funzioni di base. Per esempio, qualsiasi funzione continua su un insieme compatto può essere approssimata con un insieme di funzioni polinomiali. Se queste funzioni di base sono funzioni trigonometriche, tale trasformata viene chiamata trasformata di Fourier (Agrawal et al., 1993; Wu et al., 2000; Sasha e Zhu, 2004; Cassisi et al., 2012). L'idea

base della decomposizione spettrale è che ogni segnale, non importa quanto complesso sia, può essere rappresentato dalla sovrapposizione di un numero finito di curve di seno e coseno, in cui ogni curva è rappresentata da un singolo numero complesso conosciuto come coefficiente di Fourier. La trasformata di Fourier infatti è il principale strumento dell'analisi spettrale nelle serie storiche. Nel *time series data mining*, essa viene usata come uno strumento per la riduzione dei dati.

Anche se la serie di Fourier contiene infiniti termini, molti dei coefficienti di Fourier hanno un'ampiezza molto bassa e quindi contribuiscono poco alla ricostruzione del segnale. Questi coefficienti a bassa ampiezza possono essere scartati senza troppa perdita di informazioni. La serie storica ricostruita con questi pochi coefficienti DFT (*Discrete Fourier Trasform*) è l'approssimazione DFT della serie originaria. Usando sempre più coefficienti l'approssimazione migliora, ma il trend della serie originale è comunque catturato anche se utilizziamo pochi coefficienti DFT.

Nella Figura 3 viene mostrato come la DFT decompone una serie storica in frequenze differenti. Ogni frequenza è rappresentata da un coefficiente DFT. I segnali sul lato destro dell'equazione rappresentano porzioni del segnale originale a frequenze specifiche.

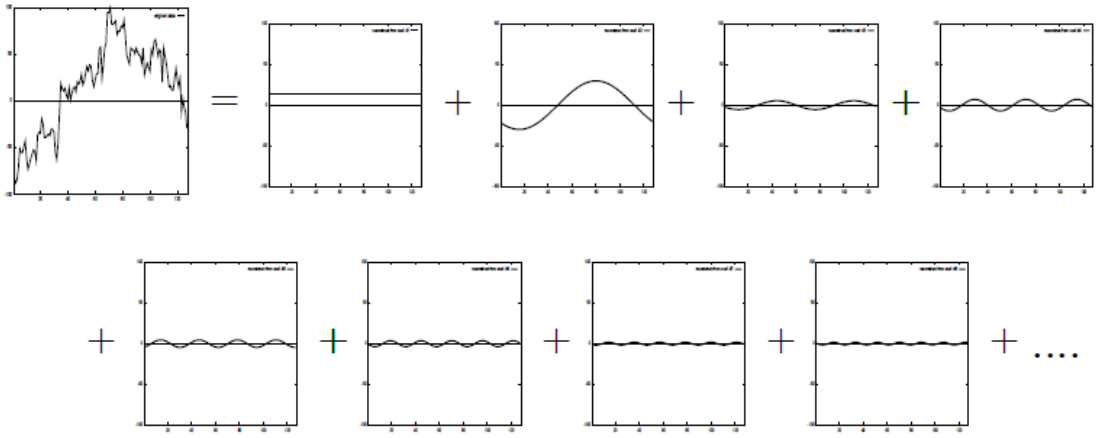


Figura 3: Decomposizione di una serie storica tramite DFT.

Nella Figura 4 viene invece mostrata la ricostruzione della serie a partire dai primi pochi coefficienti DFT.

La linea continua rappresenta la serie originale. La linea tratteggiata rappresenta la serie ricostruita mediante i primi 8 coefficienti DFT mentre la linea punteggiata rappresenta la serie ricostruita mediante i primi 64 coefficienti DFT. Si noti che le serie ricostruite catturano la forma base della sequenza originale e che più aumenta il numero di coefficienti utilizzati per ricostruire più appaiono dettagli della serie originale.

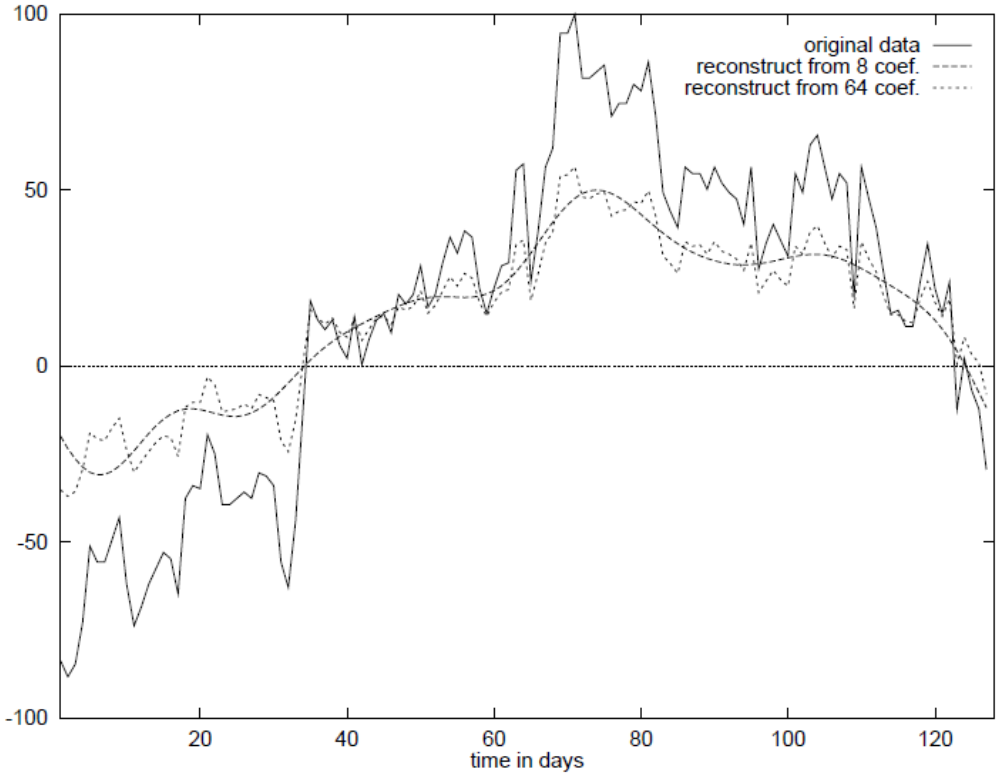


Figura 4: Ricostruzione di una serie storica mediante un certo numero di coefficienti DFT, precisamente 8 e 64.

Calcolare direttamente la DFT richiede una quantità di operazioni aritmetiche $O(n^2)$, con n pari alla dimensione dei dati.

Un salto di qualità nell'utilizzo della DFT è stata la scoperta di una classe di algoritmi veloci, genericamente chiamata FFT (*Fast Fourier Transform*), la quale ottiene lo stesso risultato con un numero di operazioni $O(n\log n)$.

1.2 Discrete Wavelet Trasform

Un'onda (*wave)* è una funzione oscillante di tempo o di spazio ed è periodica. In contrasto, i *wavelet* sono onde localizzate. Quindi, mentre la trasformata di Fourier usa onde per analizzare i segnali, la trasformata wavelet (Chan e Fu, 1999; Wu et al., 2000; Papivanov e Miller, 2002) usa invece *wavelet* (Figura 5).

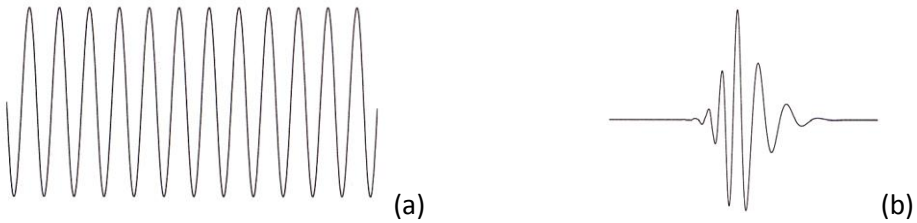
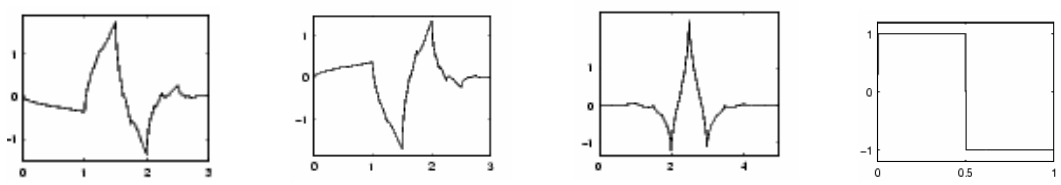


Figura 5: Esempio di *wave (a) e wavelet (b)*

La trasformata wavelet, ψ(t), si riferisce alla rappresentazione di un segnale mediante l'uso di una forma d'onda oscillante di lunghezza finita chiamata *wavelet* madre.
Un elenco di *wavelet* madre è:



Nella Figura 6 viene mostrato come la DWT decompone la serie originale in frequenze diverse. Ciò è simile alla DFT ma molte altre cose differiscono. Per prima cosa, viene usata una classe di *wavelets* chiamata Haar wavelet. Inoltre i segnali decomposti son diversi non solo nella frequenza ma anche nella posizione/tempo del segnale. Ogni risposta tempo/frequenza è rappresentata da un coefficiente DWT.

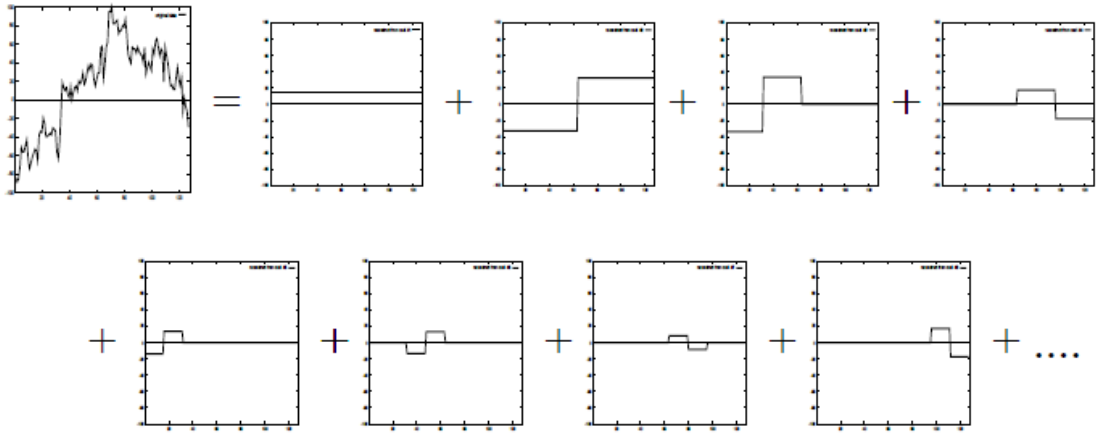


Figura 6: Decomposizione di una serie storica mediante DWT.

La ricostruzione della serie originale viene poi mostrata in Figura 7. La linea continua mostra la serie originale. La linea tratteggiata mostra la ricostruzione avvenuta usando i primi 8 coefficienti DWT, cioè le prime 8 parti tempo/frequenza della sequenza originale. La linea punteggiata invece mostra la ricostruzione mediante i primi 64 coefficienti DWT. Proprio come nella DFT, si può notare come la forma base della sequenza originale venga catturata.

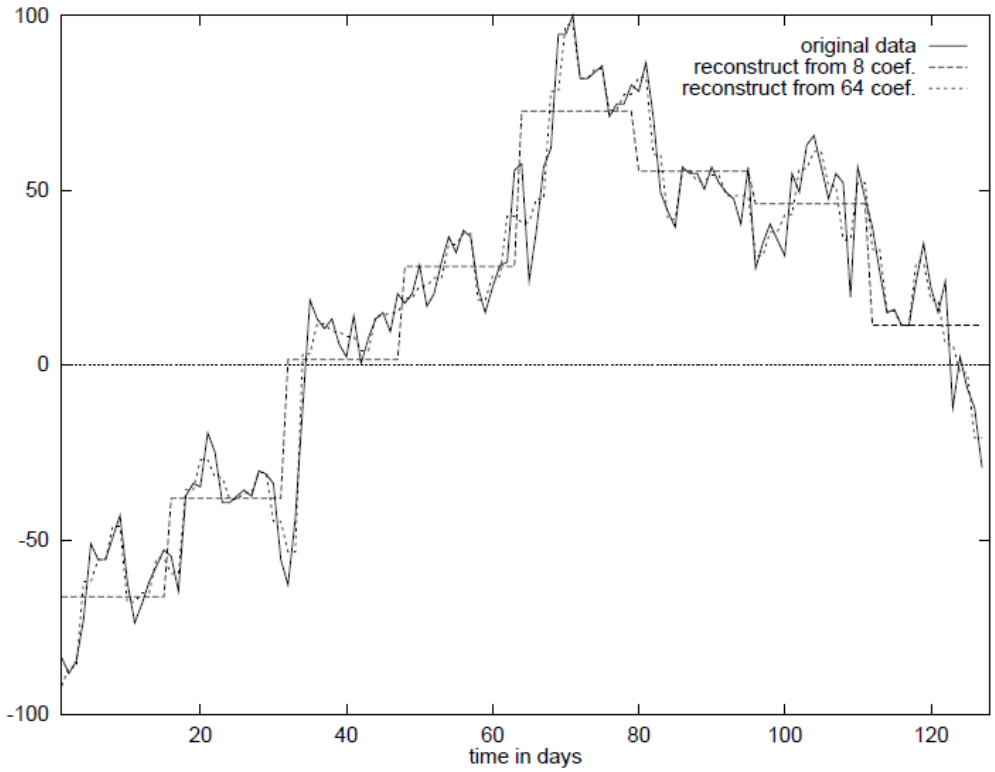


Figura 7: Ricostruzione della serie mediante DWT.

La trasformata wavelet è meno complessa computazionalmente rispetto alla DFT, cioè ha complessità di ordine O(*n*), al contrario della complessità O(*n*log*n*) della FFT (*Fast Fourier Transform*).

1.3 Piecewise Aggregate Approximation

La *Piecewise Aggregate Approximation*, in lettere PAA (Keogh et al., 2001b; Cassisi et al., 2012) è un metodo secondo cui si approssima una serie storica dividendola in segmenti di ugual misura registrando la media dei dati che rientrano nel segmento. La PAA quindi riduce i dati da una dimensione *n* a una dimensione *K*, con $K \leq n$, dividendo la serie in *K* "frammenti" uguali. Viene calcolato poi il valore medio dei dati in ogni segmento e un vettore di questi valori diventa la rappresentazione ridotta dei dati.

Notiamo che, se *K* fosse uguale a *n*, la rappresentazione trasformata sarebbe identica alla rappresentazione originale e che, se *K* fosse pari a 1, la rappresentazione trasformata sarebbe semplicemente la media della sequenza originale.

Questo metodo ha lo svantaggio di fornire un'approssimazione estremamente scadente nel caso di serie temporali altamente non stazionarie. L'evoluzione di questo approccio (chiamato *Adaptive Piecewise Constant Approximation*) consiste nell'applicazione di segmenti di lunghezza variabile in modo da ottenere una migliore approssimazione sia delle variazioni lente che di quelle veloci.

Per facilitare la comparazione del PAA con le altre tecniche di riduzione dei dati è utile visualizzarla come approssimazione di una sequenza con una combinazione lineare di *box function* (Figura 8).

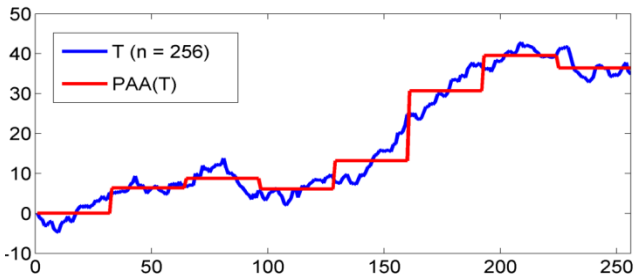


Figura 8: Rappresentazione *Pieacewise Aggregate Approximation (PAA)*

1.4 Adaptive Piecewise Constant Approximation

L'*Adaptive Piecewise Constant Approximation*, in lettere APCA (Keogh et al., 2001a; Cassisi et al., 2012) è un'evoluzione della rappresentazione PAA (*Piecewise Aggregate Approximation)*. Essa può approssimare ogni serie storica con un insieme di segmenti di lunghezza variabile. Questa rappresentazione quindi permette ai segmenti di avere lunghezze arbitrarie che a loro volta hanno bisogno di due numeri per segmento. Il primo numero registra la media di tutti i punti del segmento, il secondo numero registra il punto finale di ogni segmento (Questi due punti sono rappresentati tramite delle frecce nella Figura 9, rispettivamente di colore grigio e nero).

Il suo vantaggio principale è quello di essere capace di posizionare un singolo segmento nella zona di bassa attività e molti segmenti nella zona di alta attività e, così facendo, la precisione nell'adattamento è maggiore rispetto al PAA (Figura 9).

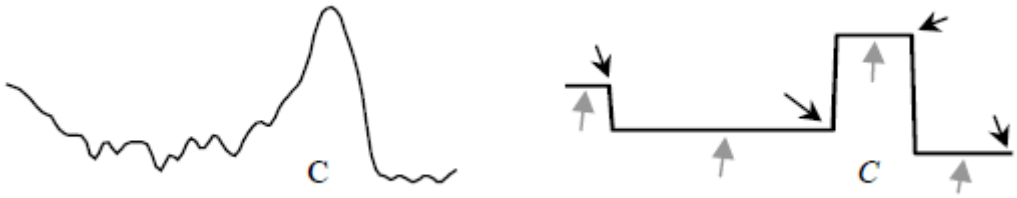


Figura 9: Illustrazione di una serie storica C e la sua rappresentazione APCA in cui vengono evidenziati i punti medi e i punti finali di ogni segmento.

In generale, una rappresentazione ottimale richiede un algoritmo di programmazione dinamica di ordine $O(Kn^2)$. Per la maggior parte degli scopi, comunque, non è richiesta una rappresentazione ottimale. Spesso si utilizza un algoritmo che produce approssimazioni di alta qualità in $O(n\log n)$. Questo algoritmo lavora prima convertendo il problema in un problema di compressione wavelet di Haar; dopodiché converte nuovamente la soluzione nella rappresentazione APCA.

1.5 Symbolic Aggregate Approximation

L'idea base della *Symbolic Aggregate Approximation*, SAX (Lin et al. 2007; Cassisi et al., 2012) è quella di convertire i risultati della PAA (*Piecewise Aggregate Approximation)* in una stringa di simboli.

Lo spazio di distribuzione (asse *y*) è diviso in regioni equiprobabili. Ogni regione viene rappresentata da un simbolo e ogni segmento può essere attribuito al simbolo corrispondente alla regione in cui risiede.

Più specificatamente, per convertire una serie storica T di lunghezza *n* in simboli, essa viene prima normalizzata e poi viene divisa in *w* segmenti di ugual misura. Aggregando questi *w* coefficienti, essi formano la rappresentazione *Piecewise Aggregate Approximation* (PAA) di T. Successivamente, per convertire i coefficienti PAA in simboli, si determinano i *breakpoints* che dividono lo spazio di distribuzione in α regioni equiprobabili, dove α è la dimensione dell'alfabeto specificato dall'utente. In altre parole, i *breakpoint*s sono determinati in modo tale che la probabilità che un segmento cada all'interno di una regione sia approssimativamente la stessa. Se i simboli non fossero equiprobabili, alcune delle sottostringhe sarebbero più probabili di altre.

Una volta determinati i *breakpoints*, ad ogni regione è assegnato un simbolo. I coefficienti PAA possono poi essere facilmente mappati con i simboli che corrispondono alla regione in cui risiedono. I simboli sono assegnati con una strategia *bottom-up,* cioè il coefficiente PAA che cade nella regione più bassa viene convertita in "a" , in quella sopra in "b" e così via (Figura 10).

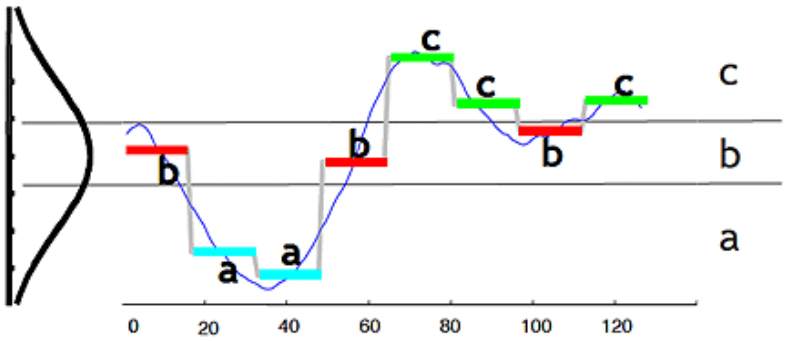


Figura 10: Rappresentazione Symbol Aggregate Approximation (SAX).

La stringa di simboli viene detta PAROLA. Nel caso della figura 9 avremo quindi che la parola è: *baabccbc.*

Minimizzare la dimensionalità con la media dei segmenti di ugual misura ha molti vantaggi quali la velocità, la flessibilità e la facilità di implementazione. Però ha anche alcuni svantaggi per certi tipi di serie storiche, specialmente per quelle finanziarie. Questo perché il suo tipo di riduzione della dimensionalità, basato sui valori medi, ha un'alta possibilità di perdere alcuni importanti caratteristiche (Figura 11).

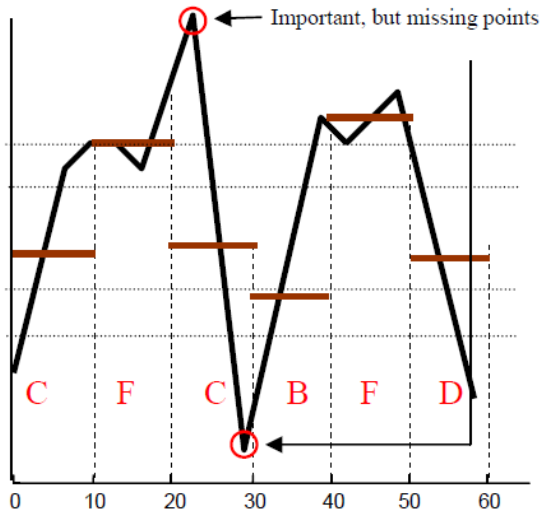


Figura 11: Perdita di punti importanti in una serie finanziaria con metodo SAX.

Per le serie finanziarie questi punti sono molto significativi. Possiamo vedere dalla Figura 11 che il valore medio del terzo segmento *equal-sized* è rappresentato dal simbolo "C", ma nel segmento ci sono punti che dovrebbero essere rappresentati dal simbolo "F", il massimo punto del segmento, e "A", il minimo punto del segmento. Proprio per dare una rappresentazione a questi punti è stato introdotto un nuovo approccio: L'*Extended SAX* (Lkhagva et al., 2006).

Il SAX originale viene esteso aggiungendo due nuovi speciali punti, cioè, i punti di massimo e di minimo per ogni segmento.

Nel seguente esempio (Figura 12) viene mostrato il valore massimo con un cerchio rosso e il valore minimo con un quadrato blu, mentre il valore medio viene evidenziato tramite un triangolo marrone.

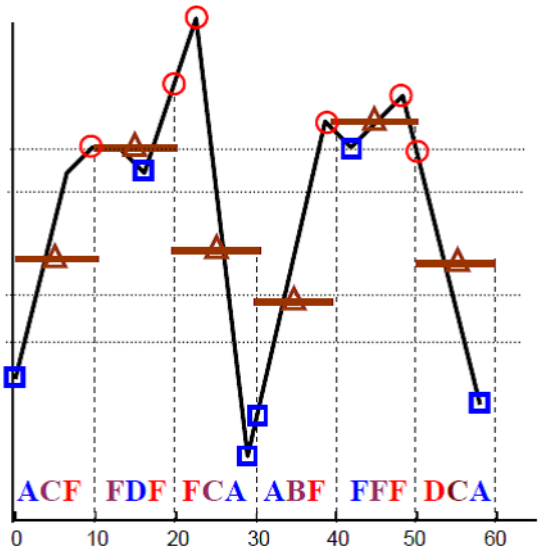


Figura 12: *Extended Sax*

Capitolo 2 - Misure di similarità

La misura di similarità è di fondamentale importanza per una grande varietà di analisi delle serie storiche e per i *tasks* del *data mining*. Essa è la misura di quanto simili sono due oggetti di dati. Nei database tradizionali questa misura si basa su un'esatta corrispondenza. Ma per quanto riguarda i dati delle serie storiche, che sono caratterizzati dalla loro natura numerica e continua, la misura di similarità è tipicamente effettuata in maniera approssimata.

La similarità nel contesto del *data mining* è solitamente descritta come una distanza le cui dimensioni rappresentano le caratteristiche degli oggetti. Una breve distanza indica un alto grado di similarità e un'ampia distanza indica invece un basso grado di similarità.

Più specificatamente, date due serie storiche Q e C, una funzione di similarità (*D*) calcola la distanza tre le due serie, ed è indicata da *D(Q, C).*

Una sintesi delle misure di similarità più comuni [Wang et al., 2012] è data dalla Figura 1.

- **Lock-step Measure**
  - L_p-norms
    - L_1-norm (Manhattan Distance)
    - L_2-norm (Euclidean Distance)
    - L_{inf}-norm
  - DISSIM
- **Elastic Measure**
  - Dynamic Time Warping (DTW)
  - Edit distance based measure
    - Longest Common SubSequence (LCSS)
    - Edit Sequence on Real Sequence (EDR)
    - Swale
    - Edit Distance with Real Penalty (ERP)
- **Threshold-based Measure**
  - Threshold query based similarity search (TQuEST)
- **Pattern-based Measure**
  - Spatial Assembling Distance (SpADe)

Figura 1: sintesi delle misure di similarità più comuni.

2.1 Lock-step Measure

Con il *LOCK-STEP MEASURE* ci riferiamo alle misure di distanza che confrontano il punto *i*-esimo di una serie con il punto *i*-esimo dell'altra serie (*"one-to-one"*). A questa categoria appartengono le L_p-norms. Date due serie $Q=\{q_1,\ldots,q_n\}$ e $C=\{c_1,\ldots, c_n\}$ di lunghezza *n*, si può definire la classe di metriche L_p nel seguente modo: $(\sum_{i=1}^{n} |q_i - c_i|^p)^{1/p}$, con p reale e maggiore o uguale a 1.

La più semplice misura di distanza per le serie storiche è la distanza euclidea, che corrisponde all' L_2-norm. Un'altra importante distanze facente parte dell' L_p-norms è la distanza di Manhattan, L_1-norm.

2.1.1 La distanza euclidea

La distanza Euclidea è probabilmente la metrica più intuitiva per le serie storiche e di conseguenza è la più comunemente usata. La distanza tra due serie, Q e C di lunghezza *n*, è calcolata come somma delle distanze tra i punti corrispondenti delle due serie.

$$D(Q, C) = \sqrt{\sum_{i=1}^{n} (q_i - c_i)^2}$$

La distanza euclidea e le altre sue varianti (L_p norms), oltre ad essere relativamente semplici da comprendere intuitivamente, hanno anche molti altri vantaggi. Un vantaggio particolarmente importante è che la complessità è lineare: O(*n*). La distanza euclidea è sorprendentemente competitiva con altri approcci più complessi, specialmente se la dimensione del dataset/database è relativamente grande. Però, dato che la corrispondenza tra i punti delle due serie storiche è fissa, queste distanze sono molto sensibili al rumore e non sono capaci di gestire i *local time shifting*. Quest'ultima significa che i segmenti simili sono fuori fase e sono quindi sensibili a piccole traslazioni lungo l'asse del tempo.

Questi problemi di deformazione e dei valori anomali richiedono l'intervento di tecniche più sofisticate, come vedremo in seguito.

2.2 Elastic Measure

Questo tipo di misure permettono una mappatura "*one-to-many*" dei punti, ma ogni punto deve essere preso in considerazione. In questo elaborato illustreremo due tipologie di

Elastic Measure, precisamente la *Dynamic Time Warping* e la *longest common subsequence similarity*.

2.2.1 La Dynamic Time Warping

Per entrambi i problemi visti in precedenza una buona soluzione è l'utilizzo di misure elastiche quali la *Dynamic Time Warping*, DTW (Keogh, 2002; Keogh e Ratanamahatana, 2005; Xi et al., 2006; Cassisi et al., 2012).

(2.a) (2.b)

Figura 2: Rappresentazione di due serie storiche, Q e C, fuori fase in cui viene calcolata la distanza euclidea (2.a) e la DTW (2.b). Si può notare come la DTW gestisca bene il *local time shifting,* al contrario della distanza euclidea.

Succede spesso che due sequenze abbiano approssimativamente la stessa forma ma che non siano allineate all'asse *x*. Al fine di trovare la somiglianza tra tali sequenze possiamo "distorcere" (*warp*) l'asse temporale di una (o entrambe) le sequenze per raggiungere un miglior allineamento. Il DTW è la tecnica per realizzare efficacemente questa deformazione.

Prima di tutto prendiamo in considerazione due serie storiche (possibilmente di lunghezza differente), $Q=\{q_1,\ldots,q_n\}$ e $C=\{c_1,\ldots, c_m\}$. L'algoritmo DTW calcola la distanza trovando il percorso (*path*) nella matrice di distanza (D) tra due serie storiche. Bisogna quindi prima costruire una matrice *nxm*. Ogni elemento della matrice (cella) contiene la distanza $d(q_i, c_j)$ tra i due punti q_i e c_j, con $i= 1, \ldots, n$ e $j=1, \ldots ,m$ (si utilizza tipicamente la distanza euclidea).

Nella Figura 3 ciascuna riga della matrice corrisponde al tempo lungo la serie Q (la serie rossa) e ciascuna colonna corrisponde invece al tempo lungo la serie C (serie blu).

Il *warping path*, W, è un insieme di elementi contigui della matrice che definiscono una mappatura tra Q e C. La matrice confronta quindi ogni punto della serie Q con ogni punto della serie C. Proprio per questo motivo, il numero di possibili *warping path* è molto alto.

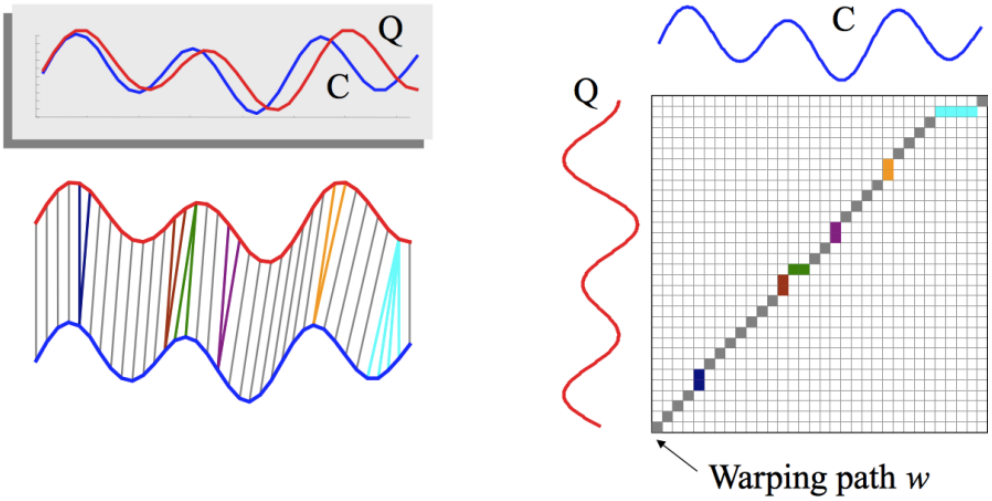
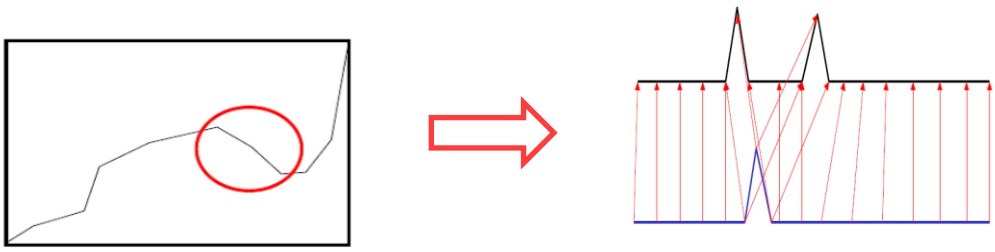


Figura 3: Rappresentazione grafica del DTW tra due serie storiche, Q e C, fuori fase (a sinistra). A destra dell'immagine è illustrata la matrice delle distanze con il *warping path* tra le due serie.

Per selezionare il *warping path* che minimizza la distanza totale, esso deve soddisfare i seguenti vincoli. Abbiamo, inoltre, inserito per ogni vincolo una figura che rappresenta cosa succederebbe se il vincolo in questione non fosse rispettato.

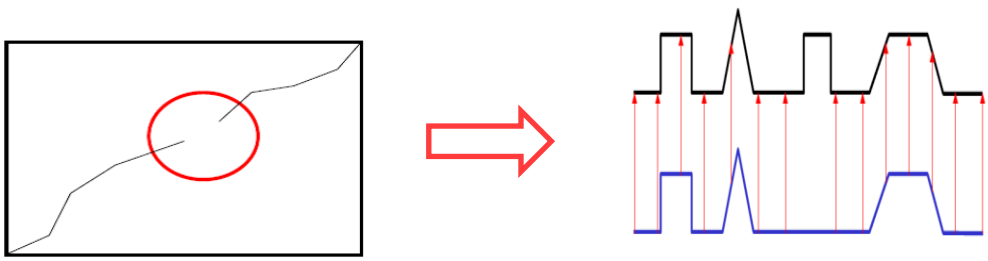
- ***Monotonia:*** $q_{i-1} \leq q_i$ e $c_{j-1} \leq c_j$.

  Il *path* non tornerà mai "indietro nel tempo". Sia l' *i*-esimo che il *j*-esimo indice o rimangono gli stessi o aumentano, non decrescono mai.



- ***Continuità***: $q_i - q_{i-1} \leq 1$ e $c_j - c_{j-1} \leq 1$.

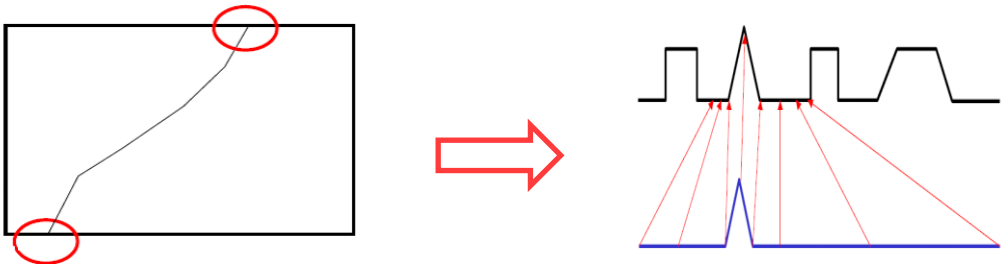
Il *path* avanza di una cella per volta. Ciò garantisce che non vengano omesse importanti caratteristiche.



- ***Boundary condition***: Le condizioni limite sono $w_1 = (1,1)$ e $w_K = (m,n)$.

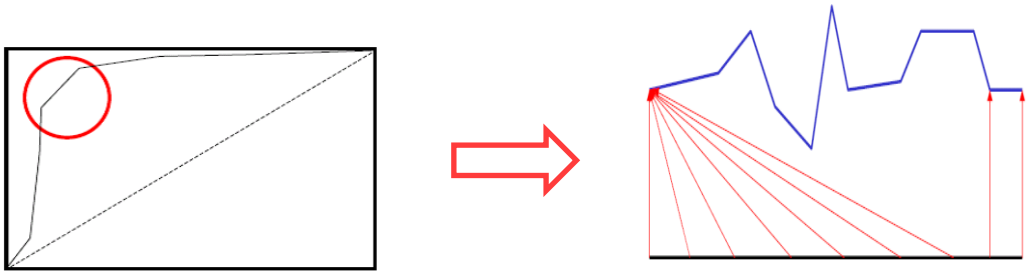
Questo obbliga il *warping path* a iniziare e finire diagonalmente.

Ciò garantisce che il *path* non consideri parzialmente una delle sequenze.



- ***Warping window condition***: $|q_i - c_j| \leq r$, con $r>0$ che è la lunghezza della finestra.

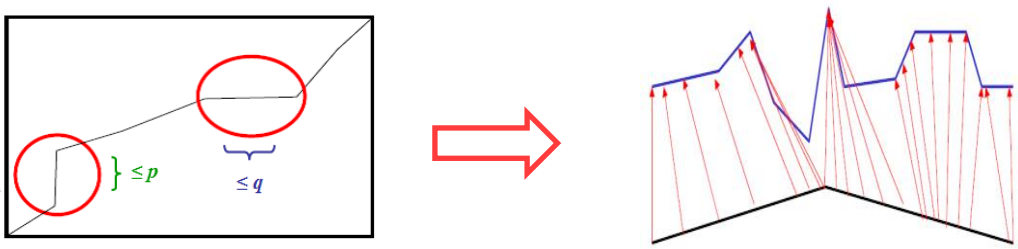
Un buon *path* è improbabile che si discosti troppo dalla diagonale. La distanza di scostamento ammissibile è data dall'ampiezza della finestra.



- ***Slope constraint (S)***: Il *path* non dovrebbe essere troppo rigido né troppo profondo.

Se $\boldsymbol{q \geq 0}$ è il numero di passi nella direzione di dell' asse x e $\boldsymbol{p \geq 0}$ è il numero di passi nella direzione dell' asse y, dopo **q** passi verso x deve essercene uno verso y, e viceversa: $\boldsymbol{S = p/q} \in [0 , \infty[$.

Ciò impedisce che una piccola parte di una sequenza venga collegata con una parte molto lunga dell'altra.



Il DTW può essere calcolato con la programmazione dinamica di complessità O(*mn*) con *m* pari alla lunghezza della serie C e *n* pari alla lunghezza della serie Q.
La distanza cumulata *D(i,j)* può essere definita come la distanza (L_1 o L_2-norm) $d(q_i, c_j)$ trovata nell'attuale cella (*i,j*) e la minima distanza tra quelle cumulate delle celle adiacenti (*i-1,j-1),(i-1, j*), (*i, j-1*), ottenendo quindi:

$$D(i,j) = \begin{cases} 0 & i = j = 0, \\ \infty & i = 0, j > 0 \text{ or } i > 0, j = 0, \\ d(q_i, c_j) + min \begin{cases} D(i-1, j-1) \\ D(i-1, j) \\ D(i, j-1) \end{cases} & i, j \geq 1. \end{cases}$$

Usando la programmazione dinamica, il DTW è un algoritmo più lento della distanza euclidea. Questo è un serio problema se vogliamo usare l'algoritmo per cercare in un grande database. La via più semplice per velocizzare l'algoritmo è quello di calcolare solo una piccola frazione della matrice. In altre parole, vogliamo che la *warping path* stia relativamente vicina alla diagonale. Se essa stesse esattamente sulla linea diagonale, ciò vorrebbe dire che le due serie corrisponderebbero esattamente. Per questo motivo sono stati introdotti dei vincoli globali, che saranno illustrati nei prossimi paragrafi.

2.2.2 Edit distance Based Measure: Longest Common Subsequence Similarity

Un altro gruppo di misure di similarità per serie storiche è stato sviluppato basandosi sul concetto di *edit distance* per stringhe. La distanza di edit (anche detta "distanza di Levenshtein") tra due stringhe A e B è il numero minimo di modifiche elementari che consentono di trasformare la A nella B. Per modifica elementare si intende la cancellazione di un carattere, la sostituzione di un carattere con un altro, o l'inserimento di un carattere.
Essa permette quindi una mappatura *"one-to-many"* dei dati, come per le misure elastiche. Tuttavia, in aggiunta, c'è in questo caso la possibilità di non collegare alcuni (uno o più) punti.

Il miglior esempio conosciuto di questa categoria è la *Longest Common Subsequence Similarity*, LCSS (Cassisi et al., 2012; Kurbalija et al., 2011) che si basa sul modello della più lunga sottosequenza comune.

Il vantaggio di questo metodo è che alcuni elementi possono non essere collegati o presi in considerazione (come per esempio gli *outliers*), cosa che invece non può avvenire nella distanza euclidea e nella DTW in cui tutti gli elementi di entrambe le sequenze devono essere utilizzati, anche se sono *outliers*.

Per due serie $Q=\{q_1,\ldots,q_n\}$ e $C=\{c_1,\ldots,c_m\}$ di lunghezza rispettivamente *n e m*, la sottosequenza comune più lunga $L(i , j)$ è data da:

$$L(i,j) = \begin{cases} 0 & i = 0 \text{ or } j = 0, \\ 1 + L(i-1, j-1) & i,j > 0 \text{ and } q_i = c_j, \\ max\big(L(i-1,j), L(i,j-1)\big) & i,j > 0 \text{ and } q_i \neq c_j. \end{cases}$$

Per adattare questa misura all'ambiente delle serie storiche, viene introdotto un parametro soglia ε. La condizione $q_i = c_j$ è così sostituita con una condizione parametrica:

$q_i\,(1 - \varepsilon) < c_j < q_i\,(1 + \varepsilon)$, in cui $0 < \varepsilon < 1$. Due punti delle serie sono considerati collegati se la loro distanza è minore di ε.

Definiamo la dissimilarità (disparità) tra Q e C come:

$$\text{LCSS (Q, C)} = \frac{m+n-2l}{m+n}$$

dove l è la lunghezza della sottosequenza comune più lunga.

Intuitivamente, questa quantità determina il minimo numero di elementi che dovrebbero essere rimossi e inseriti in Q per trasformare Q in C.

Anche la LCSS, come il DTW, si basa sulla programmazione dinamica. Cambia però il tipo di matrice: la DTW esamina la matrice delle distanze tra i punti, mentre la LCSS esamina la matrice delle sottosequenze comuni più lunghe. Come conseguenza dell'utilizzo della programmazione dinamica, entrambi gli algoritmi sono quadratici e hanno alcune limitazioni quando si tratta di grandi dataset. Comunque, l'introduzione di vincoli globali può migliorare significativamente la performance di questi algoritmi (come si era accennato precedentemente).

I vincoli globali restringono la ricerca nella matrice che si traduce in una significativa diminuzione del numero di calcoli da effettuare. I vincoli globali di uso più frequente sono: la *banda di Sakoe- Chiba* e il *parallelogramma di Itakura* (Figura 4).

Figura 4: Illustrazione dei vincoli globali più comunemente utilizzati: la banda di Sakoe- Chiba(a) e il parallelogramma di Itakura(b).

Altre distanze di edit sono: *Edit Distance on Real sequence (EDR*, Chen et al., 2005a) , *Edit distance with Real Penalty (ERP*, Chen e Ng, 2004) *e Sequence Weighted ALignmEnt model (Swale*, Wang et al., 2012). Simile alla LCSS, anche la EDR utilizza un parametro di soglia ε, ma qui il suo ruolo è quello di quantificare la distanza tra una coppia di punti a 0 o 1. A differenza della LCSS invece, la EDR assegna una penalità ai segmenti non collegati ("gap") secondo la lunghezza di questi gap.

La distanza ERP cerca di combinare i pregi del DTW e dell'EDR, usando un punto costante di riferimento g per calcolare la distanza tra due serie. Sostanzialmente, se volessimo calcolare la distanza tra due punti r_i e s_j delle serie storiche R e S e, ad esempio, s_j fosse un gap, la ERP semplicemente calcolerebbe la distanza tra r_i e g, con g valore costante. Quindi, tutte le volte che un punto delle due serie corrisponde a un gap, la ERP lo sostituisce con g e calcola la distanza tra quest'ultimo e l'altro punto della serie.

Il modello di similarità di Swale si propone invece di premiare i punti collegati e di penalizzare i gap. Oltre alla soglia corrispondente ε, Swale richiede quindi la definizione di altri due parametri: il peso del "premio" r e il peso della "penalità" p. La Swale non utilizza la programmazione dinamica, a differenza di tutte le altre misure elastiche.

2.3 Threshold-based Measure e Pattern-based Measure

Più recentemente sono state proposte altre misure di (dis)similarità, *TQuEST e Spatial Assembling Distance (SpADe)*, che sono state classificate come misure rispettivamente *threshold-based* e *pattern-based*.

Per quanto riguarda il TQuEST (Aßfalg et al., 2006; Wang et al., 2012), l'idea base è che, dato un parametro di soglia τ, una serie storica è trasformata in una sequenza di intervalli chiamati *threshold-crossing*, in cui i punti che rientrano in ogni intervallo hanno un valore maggiore di un dato τ. Ogni intervallo di tempo è poi trattato come un punto a due dimensioni: X è il tempo di partenza e Y è il tempo finale. La similarità tra due serie è allora definita dalla somma di Minkowski delle due sequenze dei punti di intervallo.

L'ultimo approccio considerato in questo lavoro è la SpADe (Chen et al., 2007b) che è una misura di similarità *pattern-based* per serie storiche. L'idea base dietro questo algoritmo è quella di trovare segmenti corrispondenti all'interno dell'intera serie storica, chiamati *patterns*, consentendo lo spostamento e il ridimensionamento in entrambe le dimensioni temporali e di ampiezza. Il problema di calcolare il valore di similarità tra serie storiche è allora trasformato in quello di trovare l'insieme più simile di *patterns* collegati. Il principale svantaggio della SpADe è che essa richiede di fissare un numero alto di parametri, quali il fattore di scala temporale, il fattore di scala dell'ampiezza, la lunghezza del *pattern*, ecc.

Capitolo 3- Tecniche di indicizzazione

L'indicizzazione è una tecnica usata per velocizzare l'accesso ai dati memorizzati. Memorizzando i dati in maniera intelligente e salvando informazioni aggiuntive sui dati è possibile trovare elementi senza esaminare l'intero insieme dei dati. Senza tecniche di indicizzazione tutti i sistemi di database di oggi sarebbero più o meno inutili dal momento che prenderebbero troppo tempo per esaminare l'intero dataset ogni volta che ci fosse da estrarre dei dati.

Ci sono alcune proprietà che gli schemi di indicizzazione dovrebbero rispettare, in particolare, ogni metodo di indicizzazione dovrebbe essere molto più veloce di una scansione sequenziale (o lineare), richiedere poco spazio (*overhead*), essere in grado di gestire *query* di varie lunghezze (dove con *query* si intende l'interrogazione di un database da parte di un utente) e infine permettere l'inserimento e l'eliminazione senza dover ricostruire l'indice. Inoltre, tale indice dovrebbe poter essere costruito entro un "tempo ragionevole" ed essere in grado di gestire diverse misure di distanza.

Una serie storica X di lunghezza *n*, può essere considerata un punto in uno spazio *n*-dimensionale. Questo suggerisce che la serie storica può essere indicizzata dagli *Spatial Access Methods* (SAMs). Questi ultimi permettono di suddividere lo spazio in regioni lungo una struttura gerarchica per un recupero efficiente delle serie storiche.

3.1 R-Tree

Uno dei più influenti *Spatial Access Methods* è l'R-Tree (Guttman, 1984, Sasha e Zhu, 2004). Le sequenze originali vengono compattate utilizzando un metodo di riduzione della dimensionalità e i vettori multidimensionali che ne risultano possono essere raggruppati e rappresentati in regioni, o più precisamente, in rettangoli minimi di delimitazione (*Minimum Buonding Rectangles, MB*R). La "R" in R-Tree sta per Regioni.

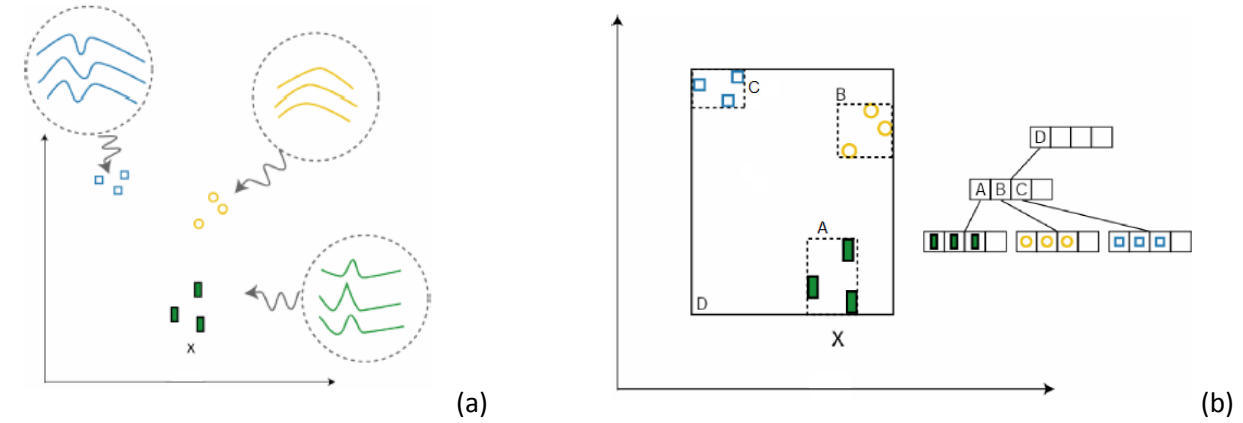


Figura 1: Riduzione della dimensionalità delle serie storiche in due dimensioni (a) e la loro rappresentazione gerarchica tramite R-Tree (b).

I vettori vengono memorizzati in nodi in modo tale che quelli spazialmente adiacenti risiedano nello stesso nodo. I nodi sono poi organizzati in una struttura di directory gerarchica ed ogni nodo di directory indica un insieme di sotto-alberi. All'apice c'è un solo nodo, chiamato nodo radice (*root*), che serve come punto di partenza per la *query* e per il processo di aggiornamento.

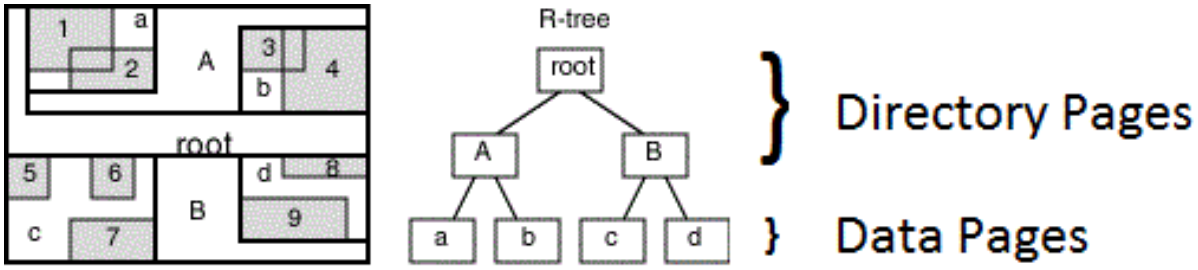


Figura 2: Struttura gerarchica dell'indice. Le *Data Pages* sono i nodi foglia dell'indice R-Tree e contengono gli MBR dei singoli vettori. In questo esempio *a* contiene l'MBR dei vettori 1 e 2, mentre *b* contiene l'MBR dei vettori 3 e 4. Man mano che si sale ai livelli superiori, nelle *Directory Pages*, le regioni sono raggruppate gerarchicamente in aggregazioni più ampie. Nell'esempio, *A* contiene l'MBR di *c* e *d*. La *root* contiene l'MBR di *A* e *B*.

La lunghezza del percorso dalla radice alla pagina dei dati (*Data Pages*) si chiama altezza (*height*) dell'indice. La struttura dell'indice è bilanciata in altezza (*height-balanced*) e ciò significa quindi che le lunghezze dei percorsi tra la radice e tutte le pagine dei dati sono uguali.

Una proprietà dell'R-Tree è che ogni pagina possiede una capacità massima *M*, che definisce quanti MBR possono essere memorizzati al massimo in una pagina, e anche una capacità minima, solitamente data da $m = M/2$.

L'R-Tree è una struttura dinamica che permette inserimenti, cancellazioni e ricerche dinamiche senza richiedere una completa ricostruzione dell'indice, in un tempo O(*logn*). Durante l'operazione di inserimento, se l'inserimento nel nodo più adatto fa sì che esso superi il numero consentito di elementi, il nodo è diviso in due e questo cambiamento può propagarsi in tutto l'albero fino alla radice. Ci sono molti metodi per attuare questo meccanismo di divisione (*split*), tra cui il *QuadraticSplit* e il *LinearSplit*.
L'algoritmo *QuadraticSplit* cerca una coppia di rettangoli (*MBR*) che sono meno collegati fra loro, cioè quelli che fanno perdere la maggiore quantità di spazio all'interno dello stesso nodo, e li colloca in due nodi separati, detti *j* e *k*. L'insieme dei rimanenti MBR sono poi esaminati e il rettangolo *i* la cui aggiunta massimizza la differenza di copertura tra i due rettangoli associati a *j* e *k*, viene aggiunto al nodo in cui la copertura viene minimizzata dall'aggiunta. Questo processo è ripetuto per tutti i rimanenti MBR. Questo metodo è così chiamato in quanto richiede un tempo quadratico.
Il *LinearSplit* invece, a differenza del precedente, sceglie i due MBR che sono più lontani fra loro. I restanti MBR nel nodo sono assegnati ai nodi la cui copertura aumenta meno con l'aggiunta dell' MBR. Questo metodo a differenza del precedente richiede un tempo lineare.

Il vantaggio di usare una struttura R-Tree è che essa può scartare istantaneamente dei dati senza nemmeno analizzarli. Gli MBR nell'R-Tree hanno la possibilità di sovrapporsi e per questo dovrà rendersi necessaria la ricerca attraverso differenti rami dell'albero. Queste sovrapposizioni riducono le prestazioni di ricerca, specialmente per spazi di dati di grandi dimensioni. Infatti, la ricerca di uno specifico rettangolo di *query* parte dalla radice dell'albero e, per ogni rettangolo in ciascun nodo, si deve decidere se esso si sovrappone al rettangolo di *query* o no. Se sì, la ricerca dev'essere compiuta anche all'interno del corrispondente nodo figlio. L'indagine prosegue così in maniera ricorsiva fino a quando non si sono attraversati tutti i nodi sovrapposti.
Per delle buone prestazioni quindi sarebbe importante che gli MBR fossero il più separati possibile. Inoltre, si dovrebbe minimizzare la copertura (*coverage*) in quanto una copertura minima riduce l'ammontare di "spazio vuoto" all'interno dei nodi. Anche per questo motivo sono stati introdotti molti metodi alternativi, come l'R^+-Tree, l'R*-Tree e l'X-Tree.

Essi forniscono miglioramenti nella performance rispetto all'R-Tree, migliorando le operazioni quali l'inserimento e lo *splitting* dei nodi.

3.1.1 R^+-Tree

L'R^+-Tree (Sellis et al., 1987) è un'estensione dell'R-Tree. A differenza di quest'ultimo però, nell'R^+-Tree gli MBR in un determinato livello dell'albero non possono essere sovrapposti. Questa caratteristica riduce il numero di rami testati e riduce notevolmente i costi computazionali in termini di tempo.

Se un rettangolo si sovrappone ad un altro può essere scomposto in più rettangoli che non si sovrappongono. Questo è il motivo per cui, se da un lato in termini di tempo i costi diminuiscono, da un altro i costi in termini di spazio, chiaramente, aumentano.

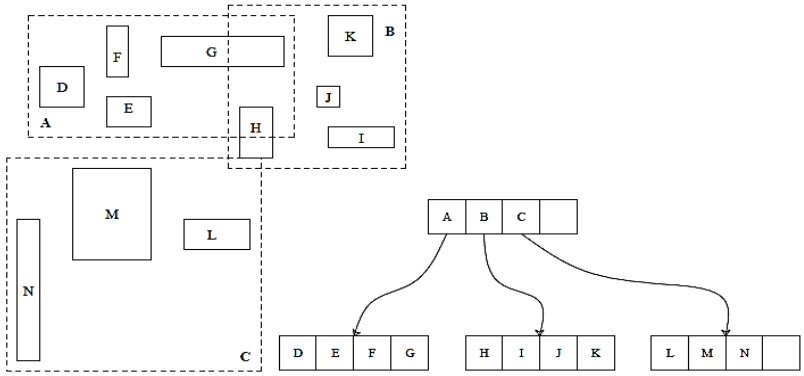


Figura 3: struttura R-Tree. Si noti la sovrapposizione tra il nodo A e il nodo B.

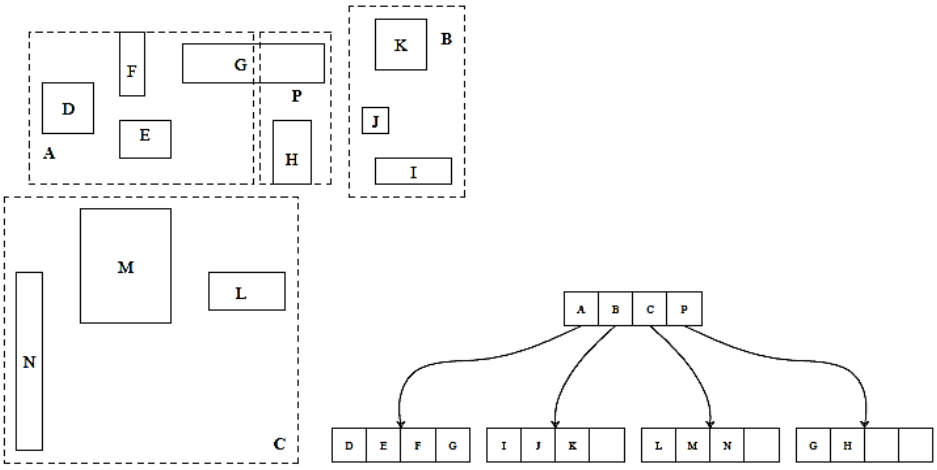


Figura 4: struttura R^+-Tree. In questo caso nessun nodo è sovrapposto ma per far questo si è dovuto introdurre un nuovo nodo P.

3.1.2 R*-Tree

L'R*-Tree (Bechmann et al., 1990) è una variante dell'R-Tree che fa uso di algoritmi più complessi per lo *splitting* dei nodi. L'R*-Tree cerca di ridurre sia le sovrapposizioni che la copertura. In particolare, l'obiettivo primario è quello di ridurre la sovrapposizione favorendo gli split che riducono la copertura e utilizzando gli split che minimizzano il perimetro dei rettangoli dei nodi risultanti. Inoltre, quando un nodo *a* è *overflow*, invece di dividerlo immediatamente, si cerca di vedere se alcuni degli oggetti in *a* potrebbero essere più adatti a stare in un altro nodo. Questo è realizzato reinserendo una porzione di questi oggetti nell'albero (chiamato *forced reinsertion*). Lo split viene effettuato solo se l'*overflow* avviene dopo il reinserimento.

3.1.3 X-Tree

Un'estensione dell'R*-Tree è l'X-Tree (Berchtold et al., 2002). Questo metodo differisce dai precedenti perché enfatizza la prevenzione delle sovrapposizioni degli MBR, che diventa un problema sempre più grande man mano che aumenta la dimensionalità. Nel caso in cui non si possano impedire sovrapposizioni, si fa diventare il nodo in questione un "supernodo" (*supernode*), cioè un nuovo nodo che può contenere non più solo *M* rettangoli, ma bensì *M+M* rettangoli.

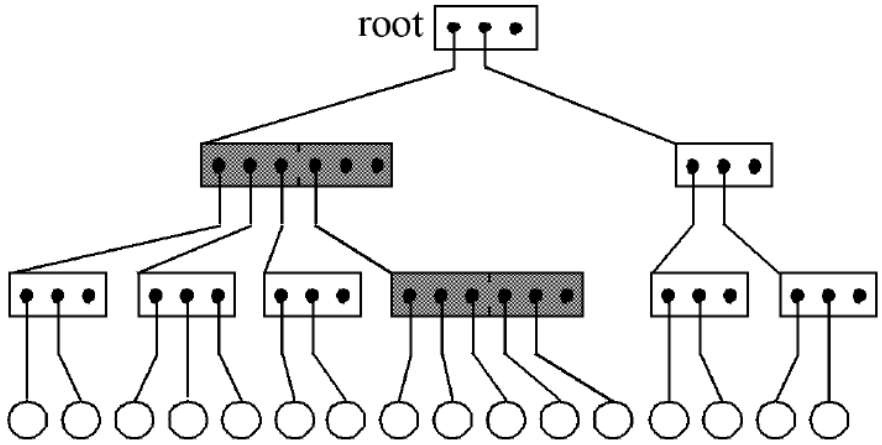


Figura 5: struttura X-Tree. In scuro sono evidenziati i supernodi, i quali risultano più grandi in quanto hanno la possibilità di contenere più rettangoli.

3.2 Altri metodi di indicizzazione

La continua crescita dei dati temporali aumenta l'importanza del recupero e della ricerca di similarità per quanto riguarda i dati delle serie storiche. Per lo specifico caso dei dati delle serie storiche, i metodi precedentemente illustrati sono di solito non adeguati. Questi indici multidimensionali sono noti per fornire guadagni di efficienza solo fino a una certa dimensionalità. Uno dei motivi per cui l'R-Tree può eventualmente fallire è che in spazi di grandi dimensioni gli MBRs si sovrappongono in modo eccessivo. Per quanto riguarda i dati delle serie storiche i problemi sono spesso aggravati. Infatti, sintetizzando i dati in MBR, non viene catturata la natura sequenziale delle serie storiche.

Per questi motivi si è proposta una struttura di indicizzazione per il recupero e la ricerca di similarità efficienti delle serie storiche: il TS-Tree (*Time Series Tree,* Assent et al., 2008).

Il TS-Tree evita la sovrapposizione e fornisce metadati riguardo i sotto-alberi, riducendo lo spazio di ricerca in modo efficace.

A causa della lunghezza della serie storica, tipicamente è richiesta una riduzione della dimensionalità. La serie storica può contenere centinaia di valori che impediscono l'applicabilità della maggior parte delle strutture di indicizzazione così come sono. La *Piecewise Aggregate Approximation* (PAA) e la *Symbolic Aggregate approXimation* (SAX) hanno prestazioni migliori rispetto alle altre tecniche di riduzione, quali la *Discrete Fourier Trasform* (DFT) per le serie storiche.

Le strutture di indicizzazione multidimensionali come, l'R-Tree o l'R*-Tree, lavorano bene per dati spaziali o per dati in un ambiente a minor dimensionalità. Per elevate dimensionalità invece, l'R-Tree e le sue varianti non sono in grado di far fronte alla "*curse of dimensionality"*.

Uno dei punti deboli più significativi per l'estrazione di serie storiche è anche la complessità temporale per costruire l'indice. Una volta che è stata decisa una rappresentazione per ridurre la dimensionalità, l'unico modo per migliorare i tempi di recupero è ottimizzare gli algoritmi di *splitting* per gli indici *tree-based*, poiché una scarsa strategia di *splitting* porta a suddivisioni eccessive e inutili che creano profondi sotto-alberi e causano attraversamenti lunghi.

Per risolvere questi problemi si è introdotta la *indexable Symbolic Aggregate approXimation* (iSAX, Shieh e Keogh, 2008). Questo approccio si basa su una modifica

della rappresentazione SAX che permette un *extendible hashing*, cioè può essere utilizzato un maggior numero di bit per identificare serie diverse. Esso possiede inoltre una proprietà di multirisoluzione che permette di indicizzare la serie storica con zero *overlap* dei nodi foglia a differenza dell'R-Tree e gli altri *spatial access methods.*

Per capire meglio questo metodo reintroduciamo brevemente alcune nozioni della rappresentazione SAX.

Una *parola* SAX è semplicemente un vettore di simboli ed ogni simbolo può essere rappresentato da una lettera o da un intero. Considerando una serie storica T di lunghezza *n* possiamo scrivere:

SAX*(T, w, a)* = $\mathbf{T}^{\mathbf{a}}$ = { t_1, t_2, . . . , t_{w-1}, t_w } in cui *w* è il numero di segmenti PAA utilizzati per rappresentare T ed *a* è la cardinalità, cioè il numero di simboli utilizzati.

Per spiegare meglio iSAX utilizziamo come simboli i numeri binari. Prendiamo per esempio la serie T di lunghezza 16 e la convertiamo in *parole* SAX. In base alla cardinalità che scegliamo possiamo ottenere diverse *parole*; se prendessimo come cardinalità 4 o 2 otterremmo:

SAX (T, 4, 4) = T^4 = {11, 11, 10, 00} e SAX (T, 4, 2) = T^2 = {1, 1, 1, 0} .

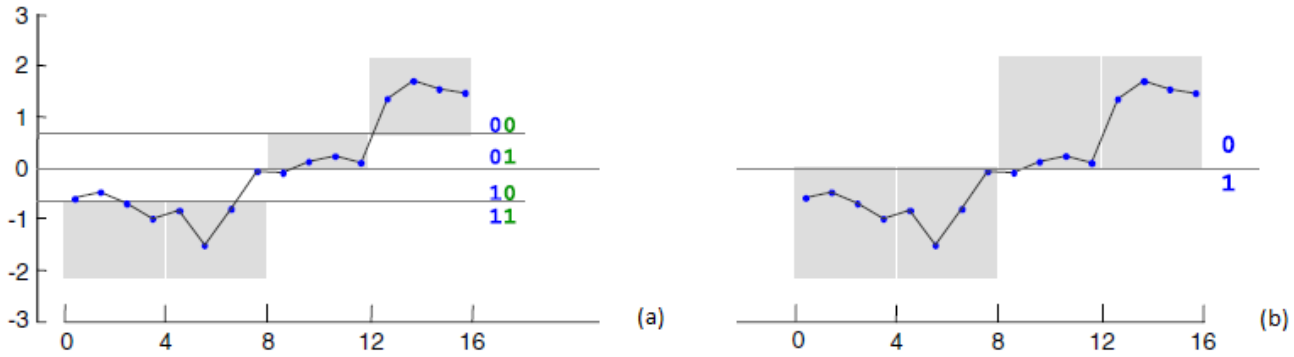


Figura 6: Rappresentazione SAX della serie T di lunghezza 16 con cardinalità 4 (a) e 2 (b).

Si può notare che è possibile ottenere T^2 dalla T^4 semplicemente ignorando i bit finali di quest'ultima. Questo vale anche per tutte le altre cardinalità, come viene mostrato nella Figura 7.

$$\mathrm{SAX}(T, 4, 16) = \mathbf{T^{16}} = \{\mathbf{1100}, \mathbf{1101}, \mathbf{0110}, \mathbf{0001}\}$$
$$\mathrm{SAX}(T, 4, 8) = \mathbf{T^{8}} = \{\mathbf{110}, \mathbf{110}, \mathbf{011}, \mathbf{000}\}$$
$$\mathrm{SAX}(T, 4, 4) = \mathbf{T^{4}} = \{\mathbf{11}, \mathbf{11}, \mathbf{01}, \mathbf{00}\}$$
$$\mathrm{SAX}(T, 4, 2) = \mathbf{T^{2}} = \{\mathbf{1}, \mathbf{1}, \mathbf{0}, \mathbf{0}\}$$

Figura 7: Esempio di riduzione della cardinalità ignorando i bit finali.

Si noti che se si dovesse rappresentare i simboli con caratteri o con interi, dalle semplici *parole* non si riuscirebbe a capire la cardinalità utilizzata. Infatti, prendendo la serie T e una cardinalità 8 avremmo: SAX (T,4,8) = {6,6,3,0}. Da quest'ultima possiamo soltanto capire che la cardinalità è sicuramente maggiore o uguale a sette ma non si può avere la certezza di quale sia quella reale.

Per porre rimedio a questa ambiguità si introduce all'interno della parola la cardinalità scritta in apice, ottenendo così per l'esempio appena visto $\{6^8,6^8,3^8,0^8\}$.

Questa scrittura è molto utile quando si considera l'iSAX. Essa infatti permette di confrontare due *parole* iSAX con cardinalità differenti. Inoltre questa proprietà permette di confrontare *parole* iSAX in cui ogni *parola* ha cardinalità diversa, come ad esempio $\{111,11,101,0\}=\{7^8,3^4,5^8,0^2\}$. Questa caratteristica dell'iSAX permette a una struttura dell'indice di effettuare uno split lungo qualsiasi simbolo. È proprio questa flessibilità che permette all'iSAX di essere indicizzabile.

Definendo i valori della cardinalità a e della lunghezza della parola w, possiamo produrre un insieme di a^w parole iSAX mutuamente esclusive. Possiamo poi creare un file, che chiamiamo ad esempio "2.4_3.4_3.4_2.4.txt", in cui inseriamo tutte le serie storiche che presentano la sequenza $\{2^4,3^4,3^4,2^4\}$. Prendiamo una soglia *th* definita come il numero massimo di serie storiche contenute in un file. Immaginiamo di attuare il processo di costruzione dell'indice e di scegliere *th*=100. In un certo istante del processo possono esserci esattamente 100 serie storiche mappate in una parola iSAX, che nell'esempio risulta $\{2^4,3^4,3^4,2^4\}$. Se poi, continuando a costruire l'indice, troviamo un'altra serie da mappare nella stessa posizione, abbiamo un *overflow* e quindi dobbiamo dividere il file. L'idea è quella di scegliere un simbolo iSAX, esaminare un'ulteriore bit e utilizzare il suo valore per creare due nuovi file. In questo caso, essendo il file originale pari a

$\{2^4,3^4,3^4,2^4\}$, lo si suddivide in: $\{4^8,3^4,3^4,2^4\}$ che diventa il *Childfile1* salvato nel file "4.8_3.4_3.4_2.4.txt", e $\{5^8,3^4,3^4,2^4\}$ che diventa quindi il *Childfile2* salvato nel file "5.8_3.4_3.4_2.4.txt".

L'idea che sta sotto questo procedimento è simile a quella dell'*extendible hashing.*

L'uso dell'indice iSAX permette la creazione di strutture dell'indice che sono gerarchiche e contenenti regioni *non-overlapping.*

4. Tasks del data mining per serie storiche

I *tasks* del *data mining* forniscono una descrizione concisa e sintetica delle serie storiche e presentano interessanti proprietà generali delle serie. Gli strumenti e le tecniche di *data mining* sono quindi utilizzati per riconoscere, anticipare e imparare il comportamento delle serie storiche di interesse.

Nei paragrafi successivi andremo a chiarire l'idea che sta alla base dei seguenti *tasks*:

- **Query by content**: Data una serie storica di *query* Q e una misura di similarità D(Q,T), questo task trova la serie storica più simile alla *query* Q nel database DB.

- **Classification**: Data una serie storica non etichettata T, essa viene assegnata a una di due o più classi predefinite.

- **Clustering**: Trova i raggruppamenti naturali delle serie storiche nel database DB in base a una certa misura di similarità D(T,C).

- **Anomaly detection**: Data una serie storica T, trova tutte le sezioni di T che contengono anomalie o "sorprendenti, interessanti, inaspettati" eventi.

- **Motif discovery**: Data una serie storica T, trova tutti i *patterns* simili che si ripetono all'interno della serie.

4.1 Query by content

Il *query by content* (Chakrabarti et a., 2002; Chan et al., 1999; Ratanamahatana et al., 2005; Yi e Faloutsos, 2000) è un *task* del *data mining* che si basa sul recupero di un insieme di serie storiche che sono più simili a una *query* fornita dall'utente.

Data una serie storica $T=\{t_1,\ldots,t_n\}$, una *query* $Q=\{q_1,\ldots,q_n\}$ e una misura di similarità D(Q,T), la *query by content* trova la lista ordinata $L=\{T_1,\ldots,T_n\}$ di serie storiche in un database DB, tale che $\forall Tk, Tj \in L, k > j \Leftrightarrow D(Q, Tk) > D(Q, Tj)$.

Il contenuto del set di risultati dipende poi dalle tipologie di *query* eseguite sul database.

La definizione precedente è infatti una formalizzazione generalizzata di una *query by content.*

Dato un database di serie storiche, la via più semplice per trovare la corrispondenza più vicina a una determinata sequenza di *query* Q è quella di eseguire un scansione lineare (o anche detta sequenziale) dei dati. Ogni serie viene recuperata dal disco e la sua distanza dalla *query* Q viene calcolata secondo la misura di similarità precedentemente scelta. Dopo che la *query* è stata confrontata con tutte le serie nel database, quella che presenta la distanza minore viene restituita all'utilizzatore come la corrispondenza più vicina.

Questa tecnica però è costosa da realizzare, primo perché sono richiesti molti accessi al disco e secondo perché essa opera sulle serie grezze, le quali potrebbero essere piuttosto lunghe.

Per superare il primo di questi problemi introduciamo qui due tipologie di *query by content*: la *ε-range query* e la *k-Nearest Neighbors query*.

La *ε-range query* (Figura 1b) si basa sull'idea che è possibile specificare una soglia ε e recuperare tutte le serie la cui similarità con la *query* D(Q,T) è minore di ε.

Più formalmente, data una query $Q=\{q_1,\ldots,q_n\}$, un database di serie storiche DB, una misura di similarità D(Q,T) e una soglia ε, la *ε-range query* trova un insieme di serie $S=\{Ti \mid Ti \in DB\}$ che sono a distanza ε da Q. Più precisamente quindi trova $S=\{Ti \in DB \mid D(Q,Ti) \leq \varepsilon\}$.

La *k-Nearest Neighbors query* (Figura 1c) recupera le *k* serie storiche più vicine alla data *query* Q.

Data una *query* Q={$q_1,\ldots,q_n$}, un database di serie storiche DB, una misura di similarità D(Q,T) e un numero intero *k*, la *k-Nearest Neighbors query* trova un insieme *k* di serie che sono più simili a Q. Più precisamente trova $S=\{Ti|Ti\in DB\}$ tale che $|S|=k$ e $\forall Tj\notin S$, $D(Q,Ti)\leq D(Q,Tj)$.

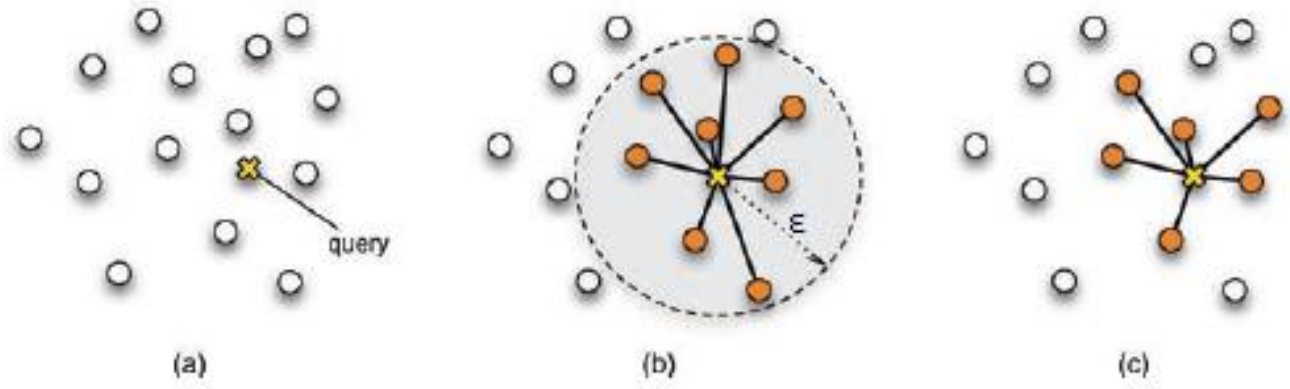


Figura 1: Illustrazione di un tipico procedimento di *query by content* rappresentato in uno spazio di ricerca bidimensionale. Ogni punto nello spazio rappresenta una serie in cui le coordinate sono associate alle sue caratteristiche (*features*). (a) Una *query* è entrata nel sistema. Dopodiché sono calcolati due tipi di *query*: la *ε-range query* (b) e la *k-Nearest Neighbors query* (c).

Per quanto riguarda questo *task* si può distinguere tra *whole series matching* e *subsequence matching.*

Nella *whole series matching* si assume che tutte le serie da confrontare abbiano la stessa lunghezza. Si ricerca quindi la serie storica più simile a una determinata *query*.

Essa richiede che la *query* venga confrontata con ogni sequenza candidata valutando una funzione di distanza e prendendo la serie che ha la distanza minore dalla *query.*

Nella *subsequence matching* si cerca invece di trovare un piccola sottosequenza di *query* all'interno di una più lunga serie storica.

Essa richiede che la sottosequenza di *query* sia posizionata in ogni possibile scostamento all'interno della più lunga serie storica.

Se utilizziamo la *ε-range query* possiamo definire la *subsequence matching* come segue*:*

Data una *query* Q, un database di serie storiche DB, una misura di similarità D(Q,T), la *subsequence matching* trova tutte le sottosequenza T_i' della serie $Ti \in DB$ tali che $D(Q,T_i') \leq \varepsilon$.

4.2 Classification

La classificazione (Lin et al., 2007; Radovanovic et al., 2010; Ratanamahatana et al., 2005; Rodriguez and Kuncheva, 2007; Xi et al.,2006) è una tecnica che permette di associare ad un insieme di serie temporali una particolare classe di appartenenza chiamata anche *etichetta*. Un metodo di classificazione ha il compito di individuare una funzione che calcoli l'etichetta di appartenenza ad una classe tra un insieme di classi possibili. L'etichetta viene calcolata sulla base di un determinato numero di caratteristiche dell'oggetto. La scelta di tali caratteristiche è sempre una fase estremamente critica; non esistono caratteristiche migliori di altre a priori ma devono essere scelte di volta in volta analizzando il problema specifico da risolvere. La classificazione è spesso indicata come "conoscenza supervisionata", e ciò si riferisce al fatto che le classi sono definite dal progettista prima di esaminare i dati. Una "conoscenza non supervisionata" è invece il *clustering* in quanto in questo caso le classi sono apprese autonomamente e automaticamente dal sistema.

Per effettuare la classificazione si suddivide un dataset iniziale in due parti formando così un insieme di addestramento (*training set*) e un insieme di prova (*test set*). Il *training set* è un insieme su cui vengono definite le classi, le quali quindi sono note con certezza. In modo più formale, dato un insieme S di serie e un insieme C di classi, si può considerare un insieme di addestramento:

$D^{train}=\{(s_1,c_1),\ldots,(s_D,c_D)\}$, in cui ogni coppia (s_i,c_i) è chiamata *esempio* ed s_i è una sequenza di classe c_i.

Tramite questo insieme, quindi, un algoritmo di *training* apprende quali caratteristiche discriminano gli elementi appartenenti alle differenti categorie. Si assume che l'algoritmo di apprendimento riuscirà a predire gli output per tutti gli altri esempi che ancora non ha visionato, cioè si assume che il modello di apprendimento sarà in grado di *generalizzare.*

Per valutare questa capacità di generalizzare e anche per valutare la sua accuratezza si esegue l'algoritmo sul *test set*, il quale dev'essere disgiunto dall'insieme di addestramento.

Gli algoritmi di classificazione si dividono in due categorie: metodi *eager* e *lazy*. La differenza principale consiste nella modalità di apprendimento.
Nei metodi *eager* il modello di classificazione viene costruito a partire dall'insieme di addestramento, prima di classificare un nuovo campione.
Nei metodi *lazy*, invece, l'apprendimento consiste semplicemente nel memorizzare le caratteristiche e le classi dei dati dell'insieme di addestramento e successivamente si classificano i nuovi campioni sulla base della loro somiglianza rispetto agli esempi del *training set*.
Verranno qui descritti due metodi; uno di tipo lazy, il *K-Nearest Neighbor (K-NN)* e uno di tipo *eager*, la *Support Vector Machine (SVM).*

4.2.1 K-Nearest Neighbor

L'algoritmo *k-Nearest Neighbor* (*k*-NN; Kim et al., 2012; Wu et al., 2008) si basa sul concetto di classificare un campione (cioè una serie storica) incognito considerando la classe dei *k* campioni più vicini dell'insieme di addestramento. Il nuovo campione verrà assegnato alla classe a cui appartengono la maggior parte dei *k* campioni più vicini. La scelta di *k* è quindi molto importante affinché il campione venga assegnato alla classe corretta. Se *k* è troppo piccolo, la classificazione può essere sensibile al rumore, se *k* è troppo grande invece la classificazione può essere computazionalmente costosa e l'intorno può includere campioni appartenenti ad altre classi.
Per realizzare un classificatore di tipo *k*-NN bisogna calcolare le distanze tra il campione e tutti i campioni di addestramento, identificare i *k* campioni di *training* più vicini e la loro rispettiva etichetta e scegliere infine la classe a cui appartiene la maggioranza dei campioni di *training* più vicini.
Il calcolo delle distanze può risultare computazionalmente inefficiente quando i dati di addestramento sono tanti. Per migliorare questo inconveniente si cerca principalmente di diminuire il numero di distanze da calcolare per la decisione. Anziché mantenere tutti i pattern nel *training set* e calcolare esplicitamente la distanza da ciascuno di essi, si

preferisce quindi in certi casi derivare da essi uno (o più) prototipi per ciascuna classe. Questi prototipi sono poi utilizzati per la classificazione come se fossero i soli elementi del *training set*. Un altro potenziale vantaggio di questo metodo, oltre a diminuire la quantità di distanze da misurare, è che i prototipi sono spesso più affidabili e robusti dei singoli *pattern*.

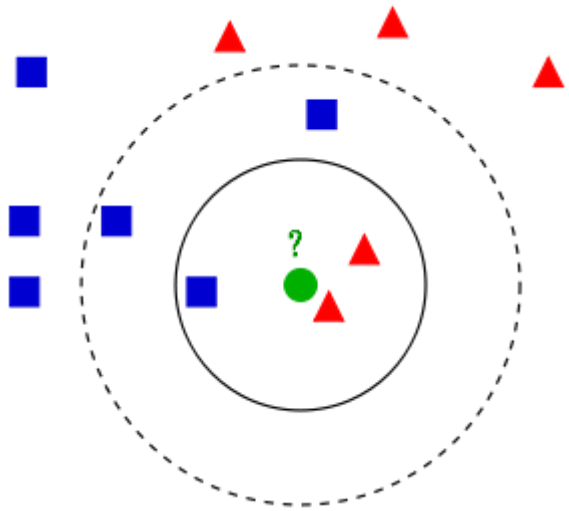


Figura 2: Esempio di una classificazione *k*-NN. La sequenza di prova (cerchio verde) potrebbe essere classificata nella prima classe, rappresentata con quadrati blu, o nella seconda classe, rappresentata con triangoli rossi. Se *k*=3 (cerchio rappresentato con la linea continua) esso verrebbe assegnato alla seconda classe perché all'interno dell'area del cerchio ci sono due triangoli e solo un quadrato blu. Se *k*=5 (cerchio rappresentato con la linea tratteggiata) esso verrebbe assegnato alla prima classe in quanto in questo caso, all'interno della più ampia area del cerchio tratteggiato, sono presenti tre quadrati e due triangoli.

I principali vantaggi di questo metodo sono che esso non richiede l'apprendimento né la costruzione di un modello, può adattare i propri confini di decisione in modo arbitrario, producendo una rappresentazione del modello più flessibile e inoltre garantisce la possibilità di incrementare l'insieme di addestramento.

Questo algoritmo presenta però anche molti svantaggi tra cui il fatto di essere suscettibile al rumore dei dati, di essere sensibile alla presenza di caratteristiche irrilevanti e di richiedere una misura di similarità per valutare la vicinanza.

4.2.2 Support Vector Machines

Gli *support vector machines* (SVM; Kim et al., 2012; Wu et al., 2008) sono metodi di classificazione *eager* che generano quindi un'approssimazione globale del modello di classificazione utilizzando i dati di addestramento.

Consideriamo un esempio di classificazione con due sole classi C={-1,1}. Supponiamo di avere un insieme di addestramento di D campioni ognuno dei quali appartiene ad una delle due classi di C. Quando le classi di possibile appartenenza sono due, come in questo caso,

siamo di fronte ad un problema di classificazione binaria. Un approccio geometrico al problema della classificazione binaria consiste nella ricerca di una superficie che separi lo spazio di input in due porzioni distinte, dove rispettivamente giacciono gli elementi delle due classi.

Nel caso in cui i dati siano linearmente separabili si può effettuare una classificazione lineare in cui si assume che esista almeno un iperpiano in grado di separare i campioni dell'insieme di addestramento di classe -1 da quelli di classe 1. L'obiettivo è quello di trovare l'iperpiano che separa nel modo migliore l'insieme dei campioni, cioè l'iperpiano che rende massimo il margine (μ), il quale rappresenta la distanza minima fra le due classi. Si noti quindi che per l'apprendimento sono importanti solo gli esempi più vicini all'iperpiano e questi vengono chiamati vettori di supporto (*supports vectors*).

La capacità di generalizzazione del classificatore cresce al crescere del margine e quindi la capacità di generalizzazione massima è data dall'iperpiano a margine massimo, detto *Optimal separating hyperplane* (OSH) (Figura 3).

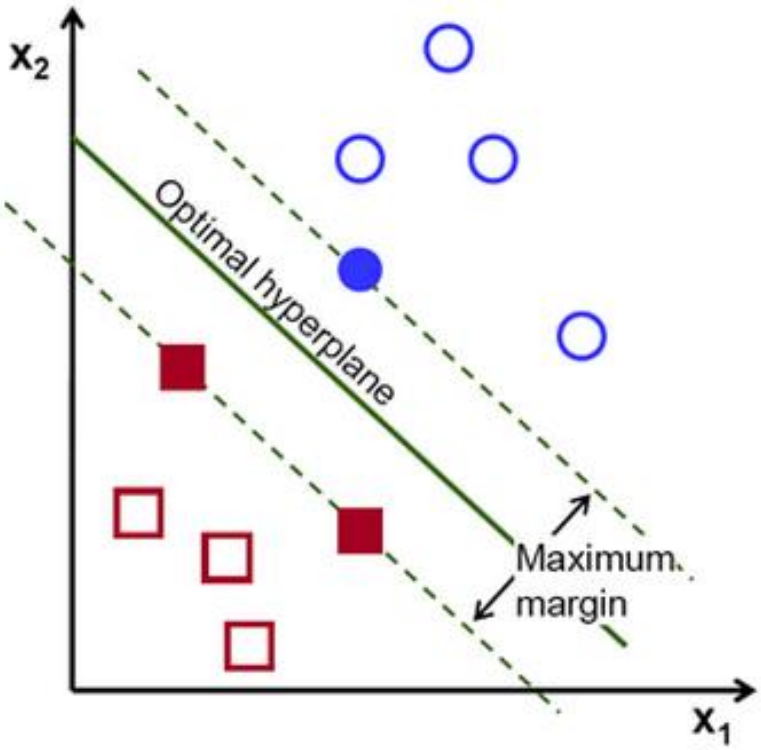


Figura 3: Individuazione dell'*Optimal separating hyperplane*. Le figure piene giacenti sugli iperpiani rappresentati da linee tratteggiate sono i *support vectors.*

Si parla invece di classificazione non lineare se non esiste nessun iperpiano in grado di separare i campioni delle diverse classi. Una soluzione consiste nel proiettare l'insieme di addestramento in uno spazio di dimensione maggiore, in cui sia possibile effettuare una classificazione lineare e trovare l'OSH. Ciò è possibile utilizzando una funzione di *mapping* φ (Figura 4).

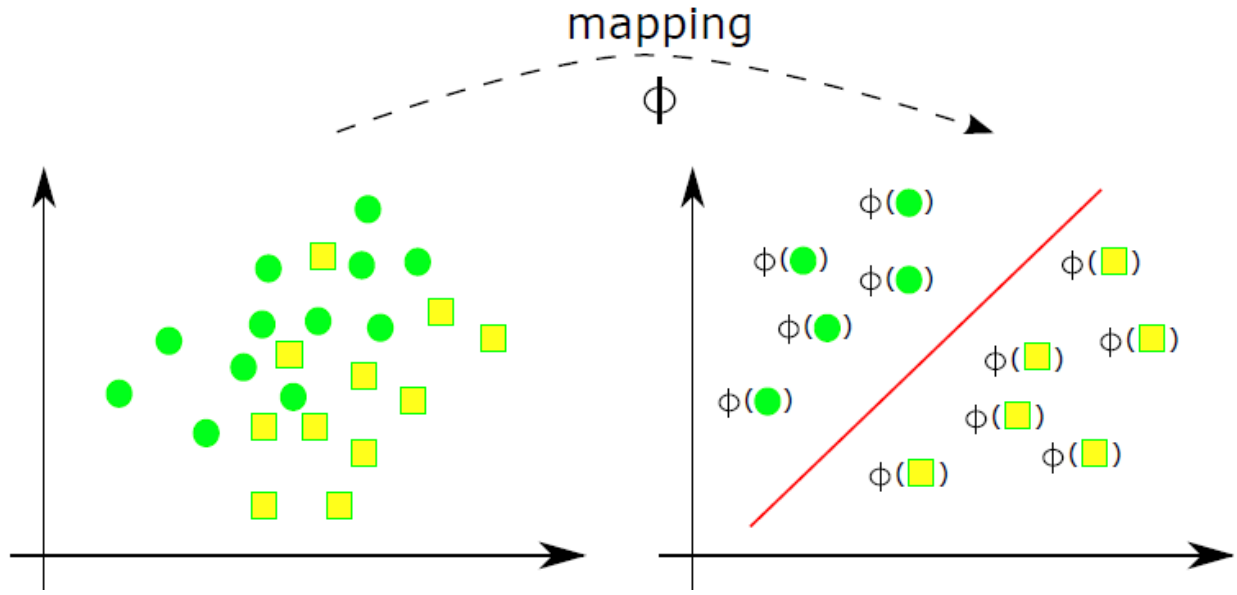


Figura 4: Dato un insieme di dati non linearmente separabili è possibile effettuare una classificazione lineare utilizzando una funzione di *mapping* φ in grado di proiettare l'insieme di addestramento in uno spazio di dimensioni maggiori.

4.3 Clustering

Il *clustering* (Berkhin, 2002; Liao, 2005; Grabusts e Borisov, 2009; Keogh e Kasetty, 2003; Ratanamahatana et al., 2005) è il processo di ricerca di gruppi, chiamati *clusters*, in un dataset.

Come avevamo già accennato nel paragrafo riguardante la classificazione, la ricerca dei gruppi è una conoscenza non supervisionata e ciò significa che l'apprendimento avviene in maniera automatica.

L'obiettivo è quello di trovare gruppi omogenei al loro interno ma che siano il più diversi possibile tra loro. Più formalmente, il raggruppamento dovrebbe massimizzare la varianza fra i *clusters* e minimizzare quella dentro i *clusters*. L'algoritmo dovrebbe quindi automaticamente individuare quali gruppi sono intrinsecamente presenti nei dati. La difficoltà principale riguardante qualsiasi problema di *clustering* di solito sta nel definire il numero corretto di gruppi.

Il *clustering task* delle serie storiche si può dividere in due sotto obiettivi: *whole series clustering* e *subsequence clustering*.

L'obiettivo del *Whole series clustering* è quello di raggruppare serie storiche in gruppi tali che le serie siano il più simili possibili tra loro all'interno di ogni gruppo.

Più precisamente, dato un database di serie storiche DB e una misura di similarità, il *Whole series clustering* trova un insieme di gruppi C= $\{c_i\}$, con i=1,… ,*k* , il quale massimizza la distanza *inter-cluster* e minimizza la varianza *intra-cluster.*

Per quanto riguarda il *subsequence clustering* i gruppi sono creati estraendo sottosequenze da una o molteplici serie storiche più lunghe.

Più formalmente, data una serie storica T=$\{t_1,\ldots,t_n\}$ e una misura di similarità, il *subsequence clustering* trova un insieme di clusters *C*=$\{c_i\}$ dove c_i è un insieme di sottosequenze che massimizzano la distanza *inter-cluster* e la coesione *intra-cluster*.

Il *clustering* nel *data mining* è caratterizzato da grandi insiemi di dati con molti attributi di tipo diverso.

La scelta dell'algoritmo da utilizzare in un dato contesto dipende dal tipo di dati disponibili, dal particolare scopo e dall'applicazione. Tradizionalmente le tecniche di *clustering* sono divise in *hierarchical* e *partitioning.*

4.3.1 Hierarchical clustering

Il *clustering* gerarchico (*Hierarchical clustering*) costruisce una gerarchia di gruppi o, in altre parole, un albero di gruppi, conosciuto come *dendrogramma* (Figura 5). Tale approccio permette di esplorare i dati in differenti livelli di granularità.

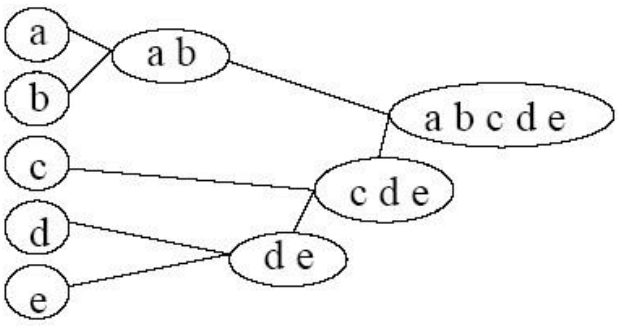


Figura 5: Clustering gerarchico, un esempio di dendrogramma.

Si distinguono spesso due tipi di metodi di *clustering* gerarchici: *agglomerativo (agglomerative)* e *divisivo (divisive)*, a seconda che si sia seguita una strategia rispettivamente *bottom-up* o *top-down*.

Un *agglomerative clustering* inizia posizionando ogni oggetto in un suo proprio gruppo e poi fonde questi gruppi in gruppi sempre più grandi, fino a quando gli oggetti non sono contenuti tutti in un unico gruppo o finché non sono rispettate determinate condizioni di terminazione.

Un *divisive clustering* parte con un singolo gruppo e ad ogni livello suddivide in sottogruppi gli elementi più diversi. Il processo continua fino a che non si raggiunge un criterio di stop il quale spesso è il numero richiesto *k* di gruppi.

Per decidere quali gruppi devono essere combinati (approccio agglomerativo) o quale gruppo deve essere suddiviso (approccio divisivo) è necessario definire una misura di dissimilarità tra gruppi. Nella maggior parte dei metodi di *clustering* gerarchico si fa uso di metriche specifiche che quantificano la distanza tra coppie di elementi e di un criterio di collegamento (di fusione). Si possono distinguere tre tipi di criteri di fusione: a legame singolo, a legame completo e a legame medio entro i gruppi.

Nel metodo a legame singolo (*Single-Linkage*), detto anche del "vicino più prossimo" (*Nearest Neighbor*), la distanza tra i gruppi è posta pari alla più piccola delle distanze calcolabili a due a due tra tutti gli elementi dei due gruppi.

Ad esempio (Figura 6) se C e D sono due gruppi composti rispettivamente da n_1 e n_2 elementi, la loro distanza, secondo questo metodo, è definita come la più piccola (il minimo) tra tutte le n_1n_2 distanze che si possono calcolare tra ciascuna unità *i* di C e ciascuna unità *j* di D:

$d(C,D)= min(d_{ij})$, per ogni *i* appartenente a C e *j* appartenente a D. Una caratteristica di questo metodo è che esso privilegia l'omogeneità tra gli elementi del gruppo a scapito della differenziazione netta tra gruppi.

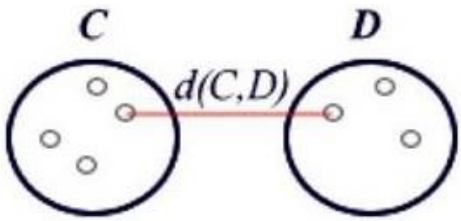


Figura 6: Metodo del legame singolo.

Nel metodo a legame completo (*Complete-Linkage*), detto anche del "vicino più lontano" (*Furthest Neighbor*), si considera la maggiore delle distanze calcolate a due a due tra tutti gli elementi dei due gruppi (Figura 7). Si avrà quindi: $d(C,D)=max(d_{ij})$, per ogni *i* appartenente a C e *j* appartenente a D.

Si uniscono i due gruppi che presentano la più piccola distanza così definita. Questo metodo quindi, al contrario del precedente, privilegia la differenza tra i gruppi piuttosto che l'omogeneità degli elementi di ogni gruppo.

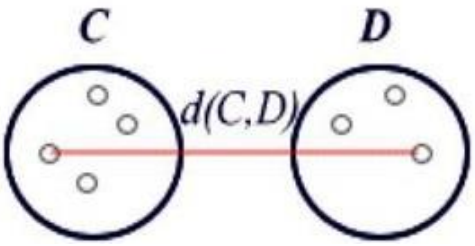


Figura 7: Metodo del legame completo.

Nel metodo a legame medio (*Average-Linkage*), si considera, come distanza tra due gruppi, la media di tutte le distanze calcolate a due a due tra tutti gli elementi dei due gruppi (Figura 8). Si avrà quindi: $d(C,D)= \frac{1}{n_1 n_2} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} d_{ij}$, per ogni *i* appartenente a C e *j* appartenente a D. Si uniscono i due gruppi che presentano la più piccola distanza così definita. Essendo basato sulla media delle distanze, i risultati sono più attendibili e i gruppi risultano più omogenei e ben differenziati tra di loro.

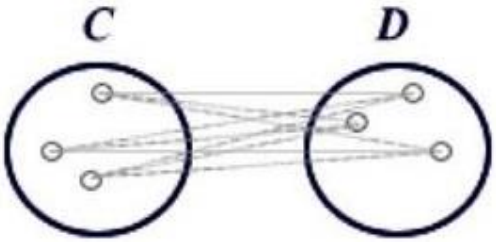


Figura 8: Metodo del legame medio.

I vantaggi del *clustering* gerarchico sono legati alla possibilità di studiare e comprendere il processo agglomerativo che porta le diverse unità ad aggregarsi in gruppi man mano che si risale la gerarchia. Inoltre, non è necessario assegnare alcun numero di gruppi in quanto il numero desiderato di gruppi può essere ottenuto "tagliando" il dendrogramma al livello appropriato.

4.3.2 Partitioning clustering

Il *clustering* partizionale (*Partitioning clustering*) divide gli oggetti in diversi gruppi e ogni oggetto può appartenere ad un solo gruppo (Figura 9). Questo metodo, a differenza del *clustering* gerarchico, necessità anche del numero *k* di gruppi come input.
Dato quindi *k*, un metodo partizionale crea innanzitutto un partizionamento iniziale. Su tale partizionamento viene, successivamente, applicata una tecnica di rilocazione iterativa che tenta di migliorarlo spostando oggetti da un gruppo ad un altro.

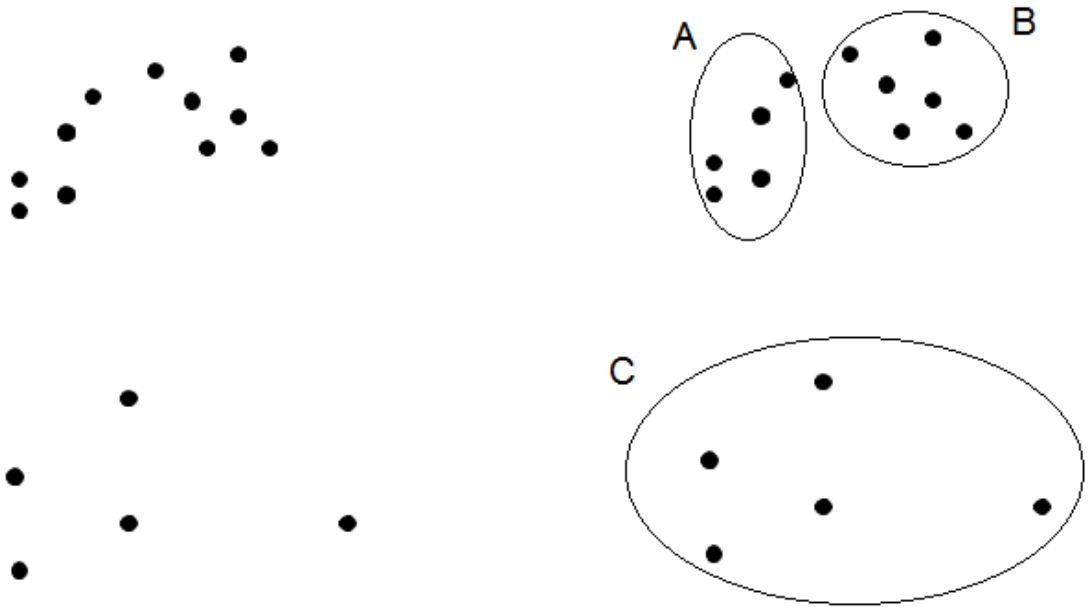


Figura 9: Esempio di *partitioning clustering*. Ogni punto rappresenta una serie storica. Stabilito il numero di gruppi *k*=3 si possono individuare quindi tre sottoinsiemi: A,B e C.

Il criterio generale di un buon partizionamento è che gli oggetti nello stesso gruppo devono essere "vicini", o correlati, l'un l'altro, mentre gli oggetti di gruppi differenti sono molto distanti tra loro.

Le distanze o le similarità tra coppie di oggetti possono essere usate per calcolare le misure delle relazioni tra e all'interno dei gruppi.
Il *clustering* partizionale utilizza algoritmi euristici iterativi e tra essi citiamo: l'algoritmo *k-means*, dove ciascun gruppo è rappresentato dal valore medio degli oggetti nel gruppo e l'algoritmo *k-medoids*, dove ciascun gruppo è rappresentato da uno dei suoi oggetti più rappresentativi.

Nel caso del *k-means*, un gruppo è rappresentato dal suo *centroide*, che è la media degli oggetti all'interno del gruppo. Più precisamente, l'algoritmo *k-means* riceve in input un parametro *k* e partiziona un insieme di *n* oggetti in *k* gruppi in modo tale che la similarità *intra-cluster* risultante sia alta mentre la similarità *inter-cluster* sia bassa. La similarità dei gruppi è misurata rispetto al valore medio degli oggetti in un gruppo (*centroide*).
Innanzitutto, quindi, l'algoritmo seleziona in modo casuale *k* oggetti, ciascuno dei quali rappresenta la media o il centro di un gruppo. Ciascuno degli oggetti rimanenti viene associato al gruppo più simile basandosi sulla distanza tra l'oggetto e la media del gruppo. Viene poi calcolata la nuova media per ciascun gruppo. Tale processo viene iterato fino a quando non si raggiunge la convergenza di una determinata funzione criterio. Tipicamente viene utilizzato come criterio l'errore quadratico, definito come:

$$V(U,C) = \sum_{i=1}^{K} \sum_{X_j \in P_i} ||X_j - C_i||^2$$

, in cui $P=P_1, ...,P_k$ ($i=1,...,k$) è l'insieme dei gruppi, X_j è l'oggetto j-esimo ($j=1,..,n$) e C_i è il *centroide* del gruppo P_i.
L'obiettivo dell'algoritmo è quindi quello di determinare *k* partizioni che minimizzino la funzione errore quadratico. Questo criterio cerca di rendere i *k* gruppi risultanti quanto più compatti e separati possibili.
Il metodo è relativamente efficiente nel processare grandi insiemi di dati perché la complessità computazionale dell'algoritmo è O(*nkt*), dove *n* è il numero totale di oggetti, *k* è il numero di gruppi e *t* è il numero di iterazioni. Normalmente, $k \ll n$ e $t \ll n$, dove << significa strettamente minore.
Il metodo, tuttavia, può essere applicato soltanto quando è definita la media di un gruppo. Questo può non accadere in molte applicazioni, ad esempio quando sono coinvolti dati con

attributi categorici. Inoltre, la necessità per gli utenti di specificare all'inizio *k*, ovvero il numero dei cluster, può essere vista come uno svantaggio.
Un altro svantaggio del *k-means* è che esso non è adatto per scoprire gruppi con forme non convesse o gruppi di dimensioni molto differenti e in aggiunta è sensibile al rumore e agli *outlier* dal momento che un piccolo numero di questi dati può influenzare sostanzialmente il valore medio.

Per esempio, si supponga di avere un insieme di oggetti localizzati nello spazio secondo quanto mostrato in Figura 10. Si supponga che $k = 3$, ovvero che l'utente voglia suddividere gli oggetti in 3 gruppi.
Seguendo l'algoritmo *k-means* scegliamo arbitrariamente 3 oggetti come i 3 centri iniziali dei gruppi, dove i centri sono marcati con un "+". Ciascun oggetto viene poi assegnato al cluster a cui è più vicino (Figura 10a). Al termine di tale raggruppamento è necessario ricalcolare i centri dei gruppi basandosi sugli oggetti che correntemente fanno parte del gruppo. Usando questi nuovi centri, gli oggetti vengono ridistribuiti tra i vari gruppi basandosi sulle loro distanze dai nuovi centri (Figura 10b). Il processo viene, quindi, iterato nuovamente ottenendo infine i gruppi definiti nella Figura 10c.

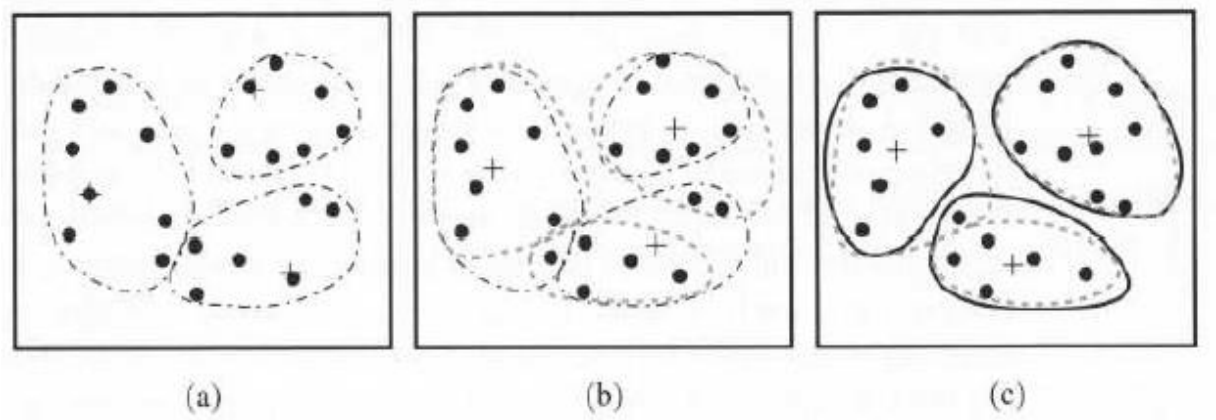


Figura 10: Esempio di partizione degli oggetti mediante l'algoritmo *k-means*.

Il *k-medoids* è stato ideato per sopperire ai limiti del *k-means*. L'idea chiave fonda le sue basi sull'utilizzo del *medoid* come elemento rappresentativo anziché la media degli oggetti del gruppo. Il *medoid* non è un oggetto centrale del gruppo ma un oggetto dello stesso che meglio lo rappresenta. Questo algoritmo somiglia molto al *k-means*. Inizialmente, sono scelti *k* oggetti in maniera casuale come *medoid* dei gruppi. Successivamente, ai gruppi vengono assegnati i rimanenti oggetti che sono più simili ai *medoids* individuati. Per ogni iterazione vengono generati nuovi *medoids* di partenza per le iterazioni successive.
Il principale vantaggio di questo metodo è che il *k-medoids* è più resistente agli *outlier*. Tuttavia, l'elaborazione di *k-medoids* è più costosa rispetto al metodo *k-means* e inoltre, come per il *k-means*, all'utente è richiesto di specificare *k*, cioè il numero di gruppi.

4.4 Anomaly detection

L'*anomaly detection* (Chandula et al., 2009; Chen e Zhan, 2008; Keogh et al., 2006) si riferisce al problema di trovare patterns nei dati non conformi al comportamento atteso della serie storica. L'importanza dell'*anomaly detection* è dovuta al fatto che le anomalie nei dati si traducono in significative (e spesso critiche) informazioni utilizzabili in una vasta gamma di applicazioni.

Data una serie storica T=$\{t_1,\ldots,t_n\}$ di lunghezza *n*, l'*anomaly detection* trova tutte le sottosequenze T'$\in$ S_T^m che contengono anomalie (l'insieme S_T^m definisce tutte le sottosequenza di lunghezza *m* ($m<n$) della serie storica T).

L'usuale approccio per rilevare anomalie è quello di creare prima di tutto un modello per la serie che ne catturi l'andamento generale, e poi qualificare come anomale tutte quelle sottosequenze che si allontanano troppo dal modello (Figura 11).

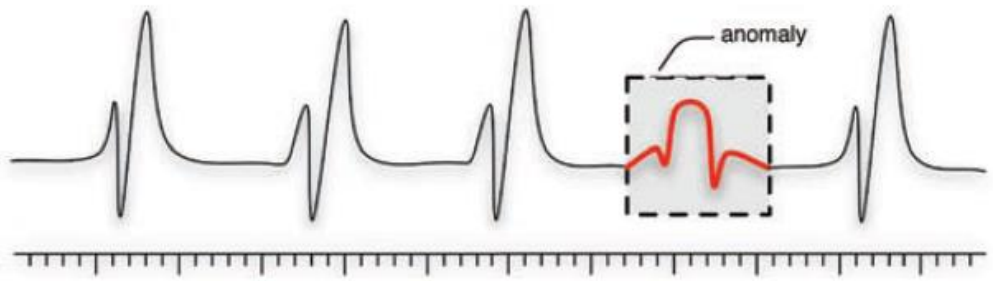


Figura 11: Esempio di un *anomaly detection.*

Il problema dell'*anomaly detection* è particolarmente difficile in quanto ciò che costituisce una anomalia differisce notevolmente a seconda del tipo di studio che si sta conducendo.

Proprio per questo motivo le ricerche esistenti sull'*anomaly detection* per le serie storiche sono state frammentate nei vari domini di applicazione.

In Chen e Zhan (2008) gli autori hanno proposto due algoritmi per trovare le anomalie in serie storiche: il PAV (*Pattern Anomaly Value*) e l'MPAV (*Multi-scale Pattern Anomaly Value*). Il PAV rileva le anomalie direttamente nella serie storica originale mentre l'MPAV le estrae nell'approssimazione wavelet della serie storica. Più precisamente, l'MPAV deriva dalla combinazione dell'algoritmo PAV con la trasformata Haar Wavelet.

4.4.1 PAV

Per definire l'algoritmo PAV è necessario introdurre alcuni concetti:

Definizione 1 (*Support count*): Data una serie storica $X=(x_1, x_2, \ldots, x_n)$ di lunghezza n e un *pattern* $Y_i=(x_i,x_{i+1})$ con $i=1,2, \ldots,n-1$, se il *pattern* Y_i si presenta δ volte nella serie storica X, il *support count* è proprio δ.

Definizione 2 (*Anomaly value, AV*): Se δ_i è il *support count* per il *pattern* Y_i e δ_i viene normalizzato nell'intervallo [0,1] tramite la seguente formula : $\delta_i^* = \frac{\delta i - min}{max - min}$ in cui *min* e *max* sono rispettivamente il minimo e il massimo del vettore contenente i *support counts* di tutti i *patterns*, l'*anomaly value* AV_i ($i=1,2, \ldots,n-1$) del *pattern* Y_i è definito come:

$AV_i = 1-\delta_i^*$.

Un *anomaly pattern* può essere definito come un *pattern* infrequente o raro, cioè il *support count* per l'*anomaly pattern* è più piccolo ripetto agli altri *pattern* della serie.

Il grado dell'anomalia di un *pattern* può essere quindi misurato in termini del suo *anomaly value*. Esso determina quanto un *pattern* appare anomalo in una data serie storica.

Definizione 3 (*anomaly pattern*): Data una serie storica *X*, il pattern Y_i ($i= 1,2, \ldots, n-1$) avente $AV_i \geq minvalue$ è chiamato *anomaly pattern*, in cui il *minvalue* è un'*anomaly value* preso come soglia.

Più alto è l'*anomaly value* dell'*i*-esimo *pattern*, maggiore è la probabilità che questo *pattern* sia un *anomaly pattern.*

In alcuni casi è difficile specificare un'appropriata soglia. In questo caso dobbiamo classificare ogni pattern in ordine decrescente sulla base del loro valore AV e dichiarare che presentano anomalie quelli in cima alla classifica.

L'algoritmo PAV trova l'*anomaly pattern* calcolando l'AV del *pattern.*

L'algoritmo include principalmente due fasi:

1] Estrazione delle caratteristiche dei *patterns*. Prima di tutto, i due punti x_i e x_{i+1} della serie storica sono collegati per formare il *pattern* $Y_i= (x_i, x_{i+1})$ con $i=1,2, \ldots,n-1$. Dopodiché si estraggono due caratteristiche del *pattern* Y_i, necessarie per determinare se due *patterns* siano uguali: la pendenza S_i e la lunghezza l_i. Essi sono calcolati con le formule:

$S_i = \frac{x_{i+1}-x_i}{t_{i+1}-t_i}$ e $l_i = \sqrt{(x_{i+1}-x_i)^2+(t_{i+1}-t_i)^2}$, in cui x_i è un punto della serie storica X e t_i il tempo in cui si trova il punto x_i. Se la serie fosse ad intervalli regolari (cioè, t_{i+1}-t_i= 1) allora le due formule diventerebbero:

$S_i = x_{i+1} - x_i$ e $l_i = \sqrt{(x_{i+1}-x_i)^2+1}$. Da queste notiamo che, in questo caso, se due *patterns* avessero la stessa pendenza, essi avrebbero anche la stessa lunghezza.

2] Calcolare *l'anomaly value* dei patterns.

Per prima cosa confrontiamo tutti i patterns della serie storica e se due *patterns* sono uguali, cioè hanno la stessa pendenza e la stessa lunghezza, il *support count* aumenterà di uno. Procedendo in questo modo si ottiene il *support count* di ogni *pattern* e successivamente, seguendo il procedimento visto in precedenza nella definizione 2, si calcola l'*anomaly value* di ogni *pattern* (AV_i).

Ottenuti questi valori AV, essi vengono ordinati in modo decrescente e si indentificano come *anomaly patterns* i primi *k patterns* corrispondenti agli AV di valore più elevato oppure quei *patterns* che hanno l'*anomaly value* maggiore o uguale a un *minvalue.*

Nel peggiore dei casi, il PAV necessita di effettuare $\frac{n(n-1)}{2}$ volte il confronto fra i *patterns* (con *n* pari alla lunghezza della serie *X*), così che la complessità computazionale sia dell'ordine $O(n^2/2)$.

Trasformando i dati con una wavelet di Haar per comprimere la serie storica possiamo rilevare l'*anomaly pattern* utilizzando poi l'algoritmo PAV. Questo nuovo metodo viene chiamato *multi-scale abnormal pattern mining algorithm* MPAV.

Questo algoritmo ha principalmente due vantaggi; la trasformata wavelet può eliminare il rumore e comprimere i dati della serie storica e ciò può migliorare l'efficienza e l'accuratezza dell'*anomaly detection*. La complessità in termini computazionali della trasformata di Haar è dell'ordine $O(n)$ e inoltre possono essere rilevate dalla serie storica i *multi-scale anomaly patterns* mediante un tasso di compressione differente. Si noti che quando il tasso di compressione è pari a zero l'MPAV e il PAV sono lo stesso algoritmo. Più aumenta il tasso di compressione, più aumenta l'efficienza dell'algoritmo MPAV.

4.5 Motif discovery

La *Motif discovery* (Yankov et al., 2007; Mueen et al., 2009; Patel et al., 2002) consiste nel trovare tutte le sottosequenze, chiamate *motif*, precedentemente sconosciute che si manifestano ricorrentemente in una lunga serie storica.

Per definire correttamente che cos'è un *motif,* dobbiamo prima dare la definizione di *trivial match*.

Data una serie storica $T=\{t_1,\ldots,t_n\}$ contenente una sottosequenza C che inizia alla posizione *p* e una sottosequenza M simile a C che inizia alla posizione *q*, e un *range* R, diciamo che M è un *trivial match* rispetto a C se $p=q$ o se non esiste una sottosequenza M' che inizia in *q*' tale che D(C, M')>R con $q<q'<p$ o $p<q'<q$.

Possiamo quindi definire un *k-motif* come:

Data una serie storica T, una sottosequenza di lunghezza *n* e un *range* R, il *motif* più significativo in T (chiamato *1-motif*) è la sottosequenza C_1 che ha il più alto numero di *non-trivial matches*. Il *k*-esimo *motif* più significativo in T (chiamato *k-motif*) è la sottosequenza C_k che ha il più alto numero di *non-trivial matches* e soddisfa la seguente condizione:

$D(C_k, C_i)>2R$, per ogni $1 \leq i < k$.

Notiamo che la definizione forza l'insieme delle sottosequenze ad essere mutuamente esclusive. Ciò è importante perché altrimenti due *motifs* potrebbero condividere la maggior parte dei loro elementi, e quindi essere essenzialmente gli stessi.

La nozione di *motifs* può essere applicata a molti *tasks* differenti. La modellazione del normale comportamento per rilevare anomalie implica di trovare il *motif* ricorrente di una serie. Per la classificazione delle serie storiche invece, può essere ottenuta un'accelerazione significativa degli algoritmi costruendo *motifs* per ogni classe. Questo *task* è molto utile anche per algoritmi di *clustering* come il *k-means,* in cui i *motifs* potrebbero essere utilizzati per risolvere i problemi di come scegliere più accuratamente i punti iniziali e di come scegliere un opportuno numero di gruppi, *k*.

Bibliografia

Aßfalg, J., Kriegel, H-P., Kröger, P., Kunath, P., Pryakhin, A., Renz, M. 2006. Similarity search on time series based on threshold queries. *Advances in Database Technology - EDBT 2006*, 276-294.

Agrawal, R., Faloutsos, C., Swami A. 1993. Efficient similarity search in sequence databases. In *Proceedings of the 4th Conference on Foundations of Data Organization and Algorithms*, 69–84.

Assent, I., Krieger, R., Afschari, F., Seidl, T. 2008. The TS-tree: Efficient time series search and retrieval. In *Proceedings of the 11th International Conference on Extending Database Technology*. 25–29.

Beckmann, N., Kriegel, H., Schneider, R., Seeger, B. 1990. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, *19*, 2, 322–331.

Berchtold, S., Keim, D., Kriegel, H. 2002. The X-tree: An index structure for high-dimensional data. *Read. Multimedia Comput. Netw. 4*, 1, 451–463.

Berkhin, P. 2002. Survey Of Clustering Data Mining Techniques. *Grouping Multidimensional Data*. 25-71

Cassisi, C., Montalto, P., Aliotta, M., Cannata, A., Pulvirenti, A. 2012. Similarity Measures and Dimensionality Reduction Techniques for Time Series Data Mining

Chakrabarti, K., Keogh, E., Pazzani, M., Mehrotra, S. 2002. Locally adaptive dimensionality reduction for indexing large time series databases. *ACM Transactions on Database Systems*. 27, 2, 188-228.

Chan, K. e Fu, A. 1999. Efficient time series matching by wavelets. In *Proceedings of the 15th IEEE International Conference on Data Engineering*. 126–133.

Chandula, V., Banerjee, A., Humar, V. 2009. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*. 41,3.

Chen, L. e Ng, R.T. 2004. On the marriage of lp-norms and edit distance. In *Proceedings of the Thirtieth international conference on Very large data bases* , 792-803.

Chen, L., Ozsu, M., Oria, V. 2005a. Robust and fast similarity search for moving object trajectories. In *Proceedings of the ACM SIGMOD International Conference on Management of Data.* ACM, 491–502.

Chen, Y., Nascimento, M.A., Ooi, B.C., Tung, A.K.H. 2007b. SpADe: On shape-based pattern detection in streaming time series. In *ICDE 2007*, 786 – 795.

Chen, X, Zhan, Y. 2008. Multi-scale anomaly detection algorithm based on infrequent pattern of time series. *Journal of Computational and Applied Mathematics*. 227 – 237.

Esling, P. e Agon, C. 2012. Time-series data mining. In *ACM Computing Surveys (CSUR)*,45,1.

Fu, T. 2011. A review on time series data mining. *Engineering Applications of Artificial Intelligence*. 164-181.

Grabusts, P., Borisov, A. 2009. Clustering methodology for time series mining. *Scientific journal of Riga technical university*. 40,81-86

Keogh, E. 2002. Exact indexing of dynamic time warping. In *Proceedings of the 28th international conference on Very Large Data Bases*, 406-417.

Keogh, E., Chakrabarti, K., Pazzani, M. 2001a. Locally adaptive dimensionality reduction for indexing large time series databases. In *Proceedings of ACM Conference on Management of Data*. 151–162.

Keogh, E., Chakrabarti, K., Pazzani, M., Mehrotra, S. 2001b. Dimensionality reduction for fast similarity search in large time series databases. *Knowl. Info. Syst. 3,* 3, 263–286.

Keogh, E. e Kasetty, S. 2003. On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration. *Data Mining and Knowledge Discovery*, 7, 349–371.

Keogh, E., Lin, J., Lee, S-H., Herle, H. V. 2006. Finding the most unusual time series subsequence: algorithms and applications. Knowl Inf Syst. 11, 1, 1–27.

Keogh, E. e Ratanamahatana, C. 2005. Exact indexing of dynamic time warping. *Knowl. Info. Syst*. 7, 3, 358–386.

Kim, J., Kim, B-S., Savarese, S. 2012. Comparing image classification methods: K-nearest-neighbor and support-vector-machines. In *Proceedings of the 6th WSEAS international conference on Computer Engineering and Applications, and Proceedings of the 2012 American conference on Applied Mathematics*. 133-138.

Kurbalija, V., Radovanović, M., Geler, Z., Ivanović, M. 2011. The Influence of Global Constraints on Similarity Measures for Time-Series Databases. In *Third International Conference on Software, Services and Semantic Technologies S3T 2011*., 67-74.

Liao, T. W., 2005. Clustering of time series data- a survey. Pattern Recognition. 38, 1857 – 1874.

Lin, J., Keogh, E., Wei, L., Lonardi, S. 2007. Experiencing SAX: a novel symbolic representation of time series. *Data Min. Knowl. Disc.*15, 2, 107-144.

Lkhagva, B., Suzuki, Y., Kawagoe, K. 2006. Extended sax: extension of symbolic aggregate approximation for financial time series data representation. In *Proceedings of Data Engineering Workshop*.

Mueen, A., Keogh, E., Zhu, Q., Cash, S. 2009. Exact Discovery of Time series Motifs.

Patel, P., Keogh, E., Lin, J., Lonardi, S. 2002. Mining motifs in massive time series databases. In *Proceedings 2002 IEEE International Conference on Data Mining*. 370-377.

Popivanov, I. e Miller, R. 2002. Similarity search over time-series data using wavelets. In *Proceedings of the International Conference on Data Engineering*. 212–224.

Radovanovic, M., Nanopoulos, A., Ivanovic, M. 2010. Time-Series Classification in Many Intrinsic Dimensions. In *Proceedings of SDM*. 677-688.

Ratanamahatana, C., Lin, J., Gunopulos, D., Keogh, E., Vlachos, M., Das, G. 2010. Mining Time Series Data. *Data Mining and Knowledge Discovery Handbook,* 1049-1077.

Rodriguez, J. and Kuncheva, L. 2007. Time series classification: Decision forests and SVM on interval and DTW features. In *Proceedings of the Workshop on Time Series Classification, 13th International Conference on Knowledge Discovery and Data mining*.

Sellis, T., Roussopoulos, N., Faloutsos, C. 1987. The R+tree: a dynamic index for multidimensional objects. In *Proceedings of the 13th International Con- ference on Varge Large Data Bases*, 507–518.

Shasha, D. e Zhu, Y. 2004. High Performance Data mining in Time Series: Techniques and Case Studies.

Shieh, J. e Keogh, E. 2008. iSAX: Indexing and Mining Terabyte Sized Time Series. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, 623-631.

Wang, X., Mueen, A., Ding, H., Trajcevski, G., Scheuermann, P., Keogh, E. 2012. Experimental comparison of representation methods and distance measures for time series data. *Data Min. Knowl. Disc.* 26, 2, 275-309.

Wu, Y., Agrawal, D., El Abbadi, A. 2000. A comparison of DFT and DWT based similarity search in time-series databases. In *Proceedings of the 9 th International Conference on Information and Knowledge Management,* 488-495.

Wu, X., Kumar, V., Quinlan, R., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G. J., Ng, A., Liu, B., Yu, P. S., Zhou, Z-H., Steinbach, M., Hand, D. J., Steinberg, D. 2008. Top 10 algorithms in data mining. J. *Knowledge Information Systems*. 4,1, 1-37.

Xi, X., Keogh, E., Shelton, C., Wei, L., Ratanamahatana, C. 2006. Fast time series classification using numerosity reduction. In *Proceedings of the 23rd International Conference on Machine Learning*, 1033- 1040.

Yankov, D., Keogh, E., Medina, J., Chiu, B., Zordan, V. 2007. Detecting time series motifs under uniform scaling. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. 844-853.

Yi, B., Faloutsos, C. 2000. Fast time sequence indexing for arbitrary lp norms. In *Proceedings of the 26th Int'l Conference on Very Large Databases*. 385-394.