Scuola di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Informatica

Tesi di Laurea

TIME SERIES K-MEANS: UN NUOVO ALGORITMO DI CLUSTERING PER SERIE TEMPORALI

TIME SERIES K-MEANS: A NEW CLUSTERING ALGORITHM FOR TIME SERIES

ELIA MERCATANTI

Relatore: *Donatella Merlini*

Anno Accademico 2016-2017

INDICE

ELENCO DELLE FIGURE

"Tutti sanno che una cosa è impossibile da realizzare, finché arriva uno sprovveduto che non lo sa e la realizza"
— Albert Einstein

INTRODUZIONE

In questo ultimo decennio abbiamo assistito ad un crescente interesse verso le tecniche di *clustering* e in particolare, verso quelle dedicate alle serie temporali, utilizzate per facilitare lo sviluppo di applicazioni avanzate in svariati campi. Con l'aumento costante della potenza di calcolo e delle capacità di memorizzazione dei computer, infatti, molteplici applicazioni hanno trovato l'opportunità di memorizzare e conservare dati per lunghi periodi di tempo, e proprio per questo, in molte applicazioni tali dati vengono conservati sotto forma di serie temporali. Possiamo citare ad esempio i dati sulle vendite di un prodotto, sulle variazioni di quote azionarie, sui tassi di cambio, ma anche dati meteorologici, biomedici ecc. Negli ultimi anni, inoltre, si sono verificati un numero significativo di cambiamenti e di nuovi sviluppi nell'area del clustering per serie temporali causati dall'emergere di nuovi concetti come i "*Big Data*" e il "*Cloud computing*" che hanno aumentato in modo esponenziale le dimensioni dei dati da trattare. Di conseguenza le tecniche di clustering hanno dovuto far fronte a queste nuove valanghe di dati per preservare l'integrità della loro reputazione come strumenti di *data mining* per la ricerca di pattern e informazioni utili da estrarre da grandi set di dati [1, 2].

Che cosa si intende per Clustering?

Il "*Clustering*" o la "*Cluster Analysis*" è un particolare insieme di tecniche di *data mining*, ovvero, la disciplina che ha come oggetto l'estrazione di informazioni a partire da grandi quantità di dati e l'utilizzo scientifico, industriale o operativo di questo sapere. In particolare, la *cluster analysis* ha il compito di selezionare e raggruppare elementi omogenei in un insieme di dati, e le tecniche di clustering, quindi, si basano su misure relative alla somiglianza degli elementi [4].

Nel nostro caso di studio, il clustering per serie temporali, miriamo dunque ad identificare strutture nascoste e ad organizzare grandi set di dati in gruppi (*clusters*) omogenei, dove le somiglianze fra gli oggetti di uno stesso gruppo sono massimizzate e le somiglianze tra oggetti appartenenti a gruppi diversi sono minimizzate. Le strutture e i pattern nascosti in un set di dati, composto da serie temporali, possono essere identificati andando a studiare i *cluster* scoperti dagli algoritmi di clustering [2].

La maggior parte degli algoritmi di clustering per serie temporali dipendono in modo critico dalla scelta della misura di similarità, e queste tecniche spesso calcolano le similarità tra i dati tenendo conto dell'intera sequenza temporale in cui sono definiti. Tuttavia, risulta molto più ragionevole calcolare la similarità fra due serie temporali in riferimento ad uno specifico intervallo di tempo, invece che considerare l'intera sequenza temporale, soprattutto per serie temporali di grandi dimensioni. Pertanto è importante identificare gli intervalli di tempo più rilevanti (o sottospazi) quando dei pattern simili vengono riconosciuti nel set di serie temporali e successivamente eseguire il clustering del set di dati a seconda dei rispettivi sottospazi rilevati. Le caratteristiche estratte da un sottospazio ricavato utilizzando algoritmi di clustering tradizionali non hanno una relazione sequenziale, tuttavia per dati basati su serie temporali un sottospazio fa parte, in genere, di un arco temporale più lungo e le caratteristiche estratte dai vari sottospazi sono interconnesse invece che indipendenti come nei normali set di dati. In generale, i sottospazi prodotti dai tradizionali algoritmi di clustering non possono essere direttamente applicati a dati basati su serie temporali [1].

L'obiettivo di questa tesi è proprio quello di presentare e sperimentare un nuovo algoritmo di clustering su sotto sequenze per serie temporali chiamato "Time Series K-means" (*TSkmeans*). Questo nuovo algoritmo è stato proposto per la prima volta da X. Huang et al. (2016) [1], un gruppo di ricercatori delle Università di Jiaotong e Hong Kong e dell'Istituto di Tecnologia di Harbin in Cina, pubblicando un interessante articolo sulla rivista *Information Sciences* e discutendone le varie caratteristiche. Il TSkmeans si basa sul ben noto algoritmo di clustering K-means ed è in grado di assegnare automaticamente un peso ai vari istanti di tempo (*time stamps*) che caratterizzano le serie temporali presenti nel set di dati che si vuole esaminare in base alla rilevanza che ha l'intervallo di tempo durante il processo di clustering. Inoltre, l'algoritmo cerca di assegnare pesi simili a istanti di tempo vicini tra loro per indurre il processo di clustering a generare sottospazi il più omogenei possibile (*smooth subspace*) [1]. Testando l'algoritmo su set di dati reali e sintetici possiamo mostrare come il TSkmeans riesce a superare in prestazioni gli algoritmi di clustering classici per serie temporali rispetto ad alcune metriche fondamentali usate nella valutazione dei risultati del clustering.

Per prima cosa, dunque, andremo ad offrire una panoramica sui concetti e sugli algoritmi utilizzati nella *cluster analysis* per poi discutere in

particolare delle tecniche e gli aspetti fondamentali che riguardano il caso specifico del clustering per serie temporali. Successivamente andremo ad analizzare l'algoritmo TSkmeans in tutti i suoi particolari ed infine mostreremo e valuteremo come l'algoritmo si comporta rispetto ad alcuni tra i più noti algoritmi di clustering per serie temporali.
Il resto di questo documento è strutturato come segue:

- **Capitolo 1**: parleremo dei concetti di base che caratterizzano la *Cluster Analysis*, mostreremo i vari approcci e i tipi di algoritmi utilizzati dalle tecniche di clustering. In particolare ci soffermeremo a descrivere nei suoi dettagli l'algoritmo *K-means*.
- **Capitolo 2**: parleremo delle tecniche di clustering applicate a dati in forma di serie temporale, ci soffermeremo a descriverne le reali applicazioni, presenteremo una classificazione di tali tecniche e introdurremo una nuova misura di prossimità, il DTW.
- **Capitolo 3**: presenteremo l'algoritmo di clustering per serie temporali TSkmeans, offrendo una sua implementazione utilizzando il software MATLAB e discuteremo delle sue principali caratteristiche.
- **Capitolo 4**: andremo a testare l'algoritmo TSkmeans su diversi set di dati basati su serie temporali, misurando le sue prestazioni rispetto ad alcune fra le metriche più utilizzate per la valutazione delle tecniche di clustering.

SOFTWARE UTILIZZATO

Per lo sviluppo dell'algoritmo TSkmeans proposto in questa tesi è stato utilizzato il software *MATLAB* sviluppato dalla *MathWorks* [11]. La piattaforma MATLAB è un ambiente di sviluppo ottimizzato per risolvere problemi scientifici e di progettazione, in particolare, per il calcolo numerico e l'analisi statistica. Consente di manipolare matrici, visualizzare funzioni e dati, implementare algoritmi, creare interfacce utente e interfacciarsi con altri programmi. Durante la stesura della tesi tale software è stato utilizzato per l'implementazione dei test per valutazione degli algoritmi di clustering ed anche per la creazione dei grafici e di alcune figure mostrate nei Capitolo 3 e 4 del documento.

1

CLUSTERING: CONCETTI DI BASE E ALGORITMI

Il *Clustering* o la *Cluster Analysis* ha il compito di suddividere un set di dati in gruppi (*clusters*) in modo significativo ed utile. Se abbiamo come obiettivo quello di ottenere dei gruppi significativi, allora i *cluster* dovrebbero essere in grado di catturare la naturale struttura del set di dati. In alcuni casi però la cluster analysis è solo un utile punto di partenza per altri scopi, come per esempio il riassunto o la compressione dei dati. Sia che venga utilizzata per la comprensione o per l'utilità, la cluster analysis ha sempre avuto un ruolo importante in una grande varietà di campi: psicologia e altre scienze sociali, biologia, statistica, riconoscimento di pattern, *information retrivial*, *machine learning* e *data mining*.

Gli argomenti trattati in questo capitolo sono in parte ripresi da *Introduction to Data Mining* (2006) [3]. In questo primo capitolo presenteremo dunque i concetti di base e le tecniche legati al clustering per set di dati generici. Vediamo innanzitutto in cosa consistono il clustering per la comprensione e quello per l'utilità.

Clustering per la comprensione

Nel contesto della comprensione dei dati, il concetto di classe assume un ruolo fondamentale. Le classi sono gruppi concettualmente significativi di oggetti che condividono caratteristiche comuni. Le persone sono molto abili nel dividere gli oggetti in gruppi (clustering) e ad assegnare particolari oggetti a questi gruppi (classificazione). Per esempio, una persona è perfettamente in grado, con estrema facilità, di riconoscere in una fotografia quali oggetti rappresentano gli edifici, i veicoli, gli animali ecc. Dunque, nel clustering per comprensione dei dati, i cluster sono potenziali classi e la cluster analysis è lo studio delle tecniche per identificare automaticamente queste classi. Alcuni dei numerosi campi di applicazione per questo contesto sono presentati di seguito.

- *La Biologia*. I biologi hanno speso molti anni per creare una tassonomia di ogni essere vivente: regno, phylum, classe, ordine, famiglia, genere e specie. La cluster Analysis può essere dunque utilizzata per trovare in automatico queste classificazioni.
- *L'Information Retrivial*. Il World Wide Web è formato da miliardi di pagine Web e il risultato di una *query* per un motore di ricerca può restituire migliaia di pagine. In questo caso il clustering può essere utilizzato per raggruppare questi risultati di ricerca in un piccolo numero di cluster, ognuno dei quali cattura un particolare aspetto della query.
- *L'Economia*. Le imprese collezionano vaste quantità di informazioni sugli attuali o potenziali clienti. Le tecniche di clustering possono essere utilizzate per suddividere i clienti in un piccolo numero di gruppi per successive analisi e operazioni di marketing.

Clustering per l'utilità

La cluster analysis fornisce un astrazione dai singoli oggetti ai cluster in cui tali oggetti risiedono. Inoltre, alcune tecniche di clustering caratterizzano ogni cluster in base ad un prototipo, ovvero un oggetto che rappresenta nel modo migliore l'intero cluster. I prototipi possono essere utilizzati come base per varie analisi o per l'utilizzo di particolari tecniche sui dati. Di conseguenza, nel contesto dell'utilità, la cluster analysis è lo studio delle tecniche per la ricerca in ogni cluster dei prototipi più rappresentativi. Alcuni esempi sono:

- *Summarization*. Un insieme di tecniche utilizzate per la riduzione di un set di dati ad un suo riassunto, per poi permettere di utilizzare quest'ultimo su vari algoritmi di analisi.
- *Compression*. L'insieme delle tecniche di elaborazione dati che permettono la riduzione della quantità di bit necessari alla rappresentazione in forma digitale di un'informazione.

Nel resto del capitolo daremo una panoramica sui concetti di base, sui vari approcci che si possono seguire per la cluster analysis e presenteremo alcuni tipi di algoritmi per il clustering. In particolare, ci soffermeremo a descrivere nei suoi dettagli l'algoritmo *K-means*, poiché verrà utilizzato come base per l'implementazione dell'algoritmo *TSkmeans* presentato in questa tesi.

1.1 INTRODUZIONE ALLA CLUSTER ANALYSIS

La cluster analysis raggruppa gli oggetti basandosi solo sulle informazioni trovate nei dati che li descrivono e che specificano le loro relazioni. L'obiettivo che si vuole raggiungere è quello di ottenere dei gruppi (cluster) in cui gli oggetti presenti sono tra loro simili o hanno una qualche relazione ma sono diversi o non correlati con oggetti appartenenti a gruppi differenti. Se quindi la somiglianza (o l'omogenità) tra gli oggetti di uno stesso gruppo e le differenze tra i gruppi sono alte avremo ottenuto un buon raggruppamento (clustering).

In molte applicazioni però, la nozione di cluster non è sempre ben definita, ad esempio prendendo in considerazione la Figura 1 possiamo vedere come per uno stesso set di dati possano esistere diversi tipi di raggruppamenti. Infatti, un raggruppamento possibile ed evidente è sicuramente quello mostrato nella Figura 1 (B), ma il set di dati può essere diviso anche in più di due cluster come mostrato dalle Figure 1 (C e D). Tuttavia, l'apparente divisone di ciascuno dei due cluster più grandi in tre sotto cluster potrebbe essere un semplice artefatto del nostro occhio umano. La figura mostra dunque che in generale la definizione di un cluster può essere imprecisa e che la sua migliore descrizione dipende dalla natura dei dati e dal tipo di risultati che vogliamo ottenere.

La cluster analysis può essere confrontata con altre tecniche che suddividono oggetti in gruppi. Ad esempio, il clustering potrebbe essere visto come una forma di classificazione, dato che assegna un cluster e quindi una sorta di classe ad ogni oggetto esaminato. Tuttavia, il clustering deriva questi assegnamenti utilizzando solo le informazioni contenute nel set

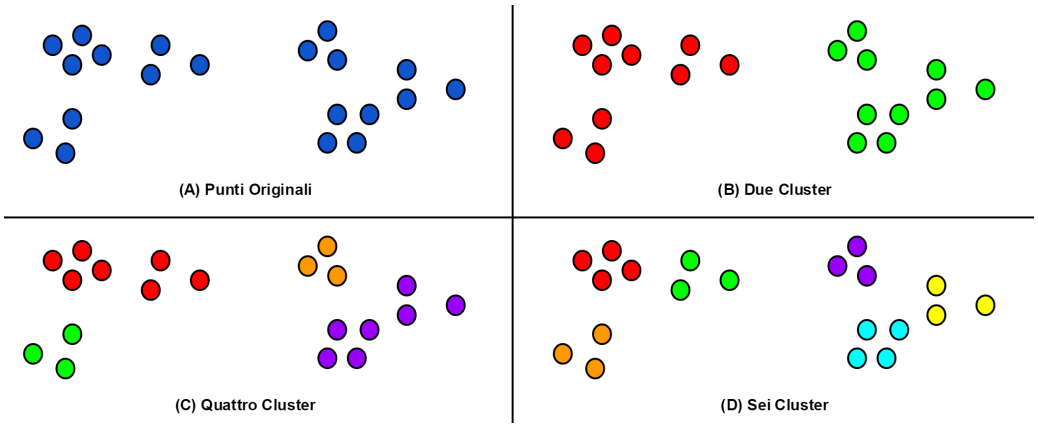

Figura 1: Modi diversi di raggruppare (clustering) lo stesso set di punti.

di dati, mentre la classificazione assegna le classi in base ad un modello sviluppato a partire da oggetti la cui classe è nota. Per questa ragione quando parliamo di cluster analysis possiamo fare riferimento al termine "apprendimento non supervisionato" (*unsupervised classification*) mentre nel caso della classificazione parliamo di "apprendimento supervisionato" (*supervised classification*).

1.2 TIPI DIFFERENTI DI ALGORITMI DI CLUSTERING

Esistono varie classificazioni delle tecniche di clustering comunemente utilizzate. La più comune categorizzazione tiene conto del tipo di procedura utilizzata per dividere il set di dati, ovvero, a seconda se il set di cluster ottenuto è annidato o meno. In questo caso, possiamo distinguere il tipo di clustering in due modi: partizionale o gerarchico.

Clustering Partizionale

Un clustering partizionale, detto anche non gerarchico o *K-clustering*, si basa semplicemente sulla divisione del set di dati in sottoinsiemi non sovrapposti (clusters) in modo tale che ogni dato (oggetto) si trovi in un unico sottoinsieme. Presa individualmente, ogni collezione di cluster mostrata nella Figura 1 (b-d) rappresenta un clustering partizionale. Alcuni tra gli algoritmi più usati in questa categoria sono il noto *K-means* oppure il suo stretto parente *K-medoids*.

Clustering Gerarchico

Se permettiamo ai cluster di avere delle gerarchie di partizioni, e quindi dei sotto cluster, allora otteniamo un clustering gerarchico, che rappresenta un set di cluster annidati avente una struttura ad albero. Ogni nodo (cluster) dell'albero (eccetto i nodi foglia) è formato dall'unione dei sui figli (sotto cluster) e la radice dell'albero è il cluster contenete tutti gli oggetti. Spesso, ma non sempre, le foglie dell'albero rappresentano un *singleton*, ovvero, contengono un solo oggetto appartenete al set di dati. Inoltre questo tipo di algoritmi possono essere divisi a loro volta in due classi.

- *Agglomerativi*: quando la strategia scelta è di tipo *bottom up*, in cui si parte dall'inserimento di ciascun oggetto in un cluster differente e si procede quindi all'accorpamento graduale di cluster a coppie.

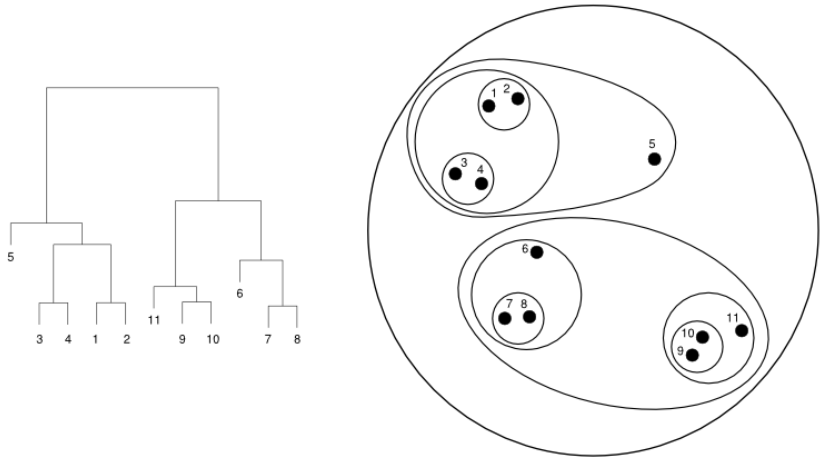

Figura 2: Esempio di clustering gerarchico.

- *Divisivi*: quando la strategia scelta è di tipo *top down*, in cui tutti gli oggetti si trovano inizialmente in un singolo cluster che viene poi suddiviso ricorsivamente in sotto cluster.

Un clustering gerarchico viene in genere rappresentato da un diagramma strutturato ad albero chiamato dendrogramma [5], che mostra le relazioni fra i vari sotto cluster e l'ordine in cui sono stati accorpati (caso agglomerativo) o suddivisi (caso divisivo). In Figura 2 viene fornito un esempio di dendrogramma prendendo in esame un set di undici punti (set a due dimensioni).

Un altra categorizzazione può dipendere dalla possibilità che un oggetto possa o meno essere assegnato a più cluster. In questo caso possiamo distinguere il tipo di clustering in tre modi: esclusivo, sovrapposto o fuzzy.

Clustering Esclusivo o Sovrapposto

Possiamo parlare di clustering esclusivo quando ogni oggetto viene assegnato ad uno ed a un solo cluster, i gruppi risultanti dunque non possono avere elementi in comune. La Figura 1 mostra un clustering di tipo esclusivo. Ci sono però molte situazioni in cui un oggetto possa ragionevolmente essere assegnato a più di un cluster. Nel caso più generale un clustering sovrapposto o non esclusivo è usato per riflettere il fatto che un oggetto possa simultaneamente appartenere a più di un gruppo (classe). Per esempio una persona può essere sia uno studente sia un

lavoratore. Inoltre, il clustering non esclusivo è spesso usato quando un oggetto si trova, rispetto ad una qualche misura di similarità, in mezzo a due o più cluster e quindi possa essere assegnato a ciascuno dei gruppi.

Fuzzy Clustering

Da un punto di vista matematico un *fuzzy set* è un insieme dove ogni oggetto appartiene ad esso in base ad un peso che varia da 0 a 1. Di conseguenza in un fuzzy clustering ogni oggetto appartiene ad ogni cluster con un'appartenenza pesata che varia da 0 a 1, dove lo 0 rappresenta la completa non appartenenza e 1 la massima appartenenza. Questo approccio è utilizzato quando si vuole sottolineare la possibilità che un oggetto possa essere "vicino" a molteplici cluster.

Infine possiamo fare un ultima distinzione fra clustering a seconda del numero di oggetti che vogliamo considerare durante il processo di raggruppamento.

Clustering Completo o Parziale

Un clustering completo assegna ogni oggetto del set di dati ad un cluster mentre un clustering parziale assegna un cluster solo ad un selezionato gruppo di oggetti. La necessità di un clustering parziale può essere giustificata quando alcuni oggetti appartenenti al set di dati non possono essere raggruppati in gruppi ben definiti, ad esempio quando gli oggetti in questione rappresentano delle anomalie, dei rumori o non sono rilevanti.

1.3 TIPI DIFFERENTI DI CLUSTER

Il processo di clustering mira a cercare gruppi di oggetti (cluster) che siano utili, dove il concetto di utilità è definito in base all'obiettivo dell'analisi che vogliamo effettuare. Ci sono dunque molteplici e differenti nozioni di cluster che possono avere una qualche utilià pratica. Di seguito riporteremo i principali tipi di cluster.

Cluster Ben Separati

Un cluster ben separato è un set di oggetti in cui ogni elemento è più vicino (o più simile) ad ogni altro elemento del cluster rispetto ad oggetti al di fuori del gruppo. Molto spesso viene stabilito un confine o un

limite con cui specificare quanto devono essere vicini (o simili) gli oggetti appartenenti ad uno stesso cluster. Questa definizione idealistica di un cluster è soddisfatta solo quando i dati contengono nella loro struttura dei cluster naturali che sono abbastanza lontani fra loro, ma in contesti reali è arduo trovare dei dati strutturati e divisi in modo preciso.

Cluster basati su Prototipi

Un cluster basato su prototipi è un set di oggetti in cui ogni elemento è più vicino (o più simile) ad un prototipo che definisce il cluster rispetto ai prototipi che definiscono gli altri gruppi. Per i dati che hanno attributi continui, il prototipo di un cluster è spesso un *centroide*, ovvero, una media (*mean*) di tutti gli oggetti contenuti nel cluster (media dei punti, serie temporali o altro). Quando però un centroide non è significativo, ad esempio quando abbiamo a che fare con dati aventi attributi categorici, il prototipo è spesso un *medoide*, o in altre parole, l'oggetto più rappresentativo del cluster. Per molti tipi di dati, i prototipi possono essere visti come gli oggetti più centrali e in questi casi si parla di *center-based clusters*.

Cluster basati su Grafi

Se i dati sono rappresentati per mezzo di un grafo, dove i nodi sono gli oggetti e gli archi rappresentato le varie connessioni fra di loro, allora un cluster può essere definito a componenti connesse (*connected component*), ovvero un gruppo di oggetti che sono collegati fra di loro ma che non hanno connessioni con gli oggetti al di fuori del gruppo. Un importante esempio di questa categoria sono i cluster basati sulla contiguità (*contiguity based cluster*), dove due oggetti sono connessi solo se sono entro una certa distanza fra di loro.

Cluster basati sulla Densità

In questa categoria un cluster è una densa regione di oggetti circondata da una regione con bassa densità di altri oggetti. Una definizione basata sulla densità è spesso impiegata quando i cluster sono irregolari o intrecciati, oppure quando sono presenti anomalie o rumori. Un esempio di tecnica di clustering appartenete a questa categoria è l'algoritmo *DBSCAN*, che produce un clustering partizionale in cui il numero di cluster da trovare viene determinato automaticamente. Inoltre tale algoritmo va ad omettere nel processo di clustering tutti quei dati che appartengono a regioni con bassa densità e quindi il risultato finale sarà un clustering parziale.

Cluster Concettuali

Più in generale possiamo definire un cluster come un set di oggetti che condividono una qualche proprietà. Questa definizione comprende tutte quelle precedenti che abbiamo presentato, ad esempio gli oggetti in un cluster basato su prototipi condividono la proprietà di essere gli oggetti più vicini allo stesso centroide o medoide. Tuttavia, i cluster concettuali includono anche nuovi tipi di cluster basati su proprietà più sofisticate.

1.4 L'ALGORITMO K-MEANS

L'algoritmo *K-means* è una tecnica di clustering partizionale basata su prototipi, il cui scopo è quello di suddividere un insieme di oggetti in K gruppi (cluster) sulla base dei loro attributi, dove il parametro K viene scelto dall'utente. Un'altra tecnica molto famosa e correlata strettamente al K-means è l'algoritmo K-medoids. Il K-means definisce un prototipo in termini di un centroide che di solito rappresenta la media di un gruppo di punti, ma può essere applicato tranquillamente anche ad oggetti in un spazio continuo ad n dimensioni. Il K-medoids, invece, definisce un prototipo in termini di un medoide, che equivale al punto più rappresentativo di un dato gruppo di oggetti e che può essere applicato ad un ampia gamma di dati, poiché richiede solo una misura di prossimità per una coppia di oggetti. Mentre un centroide raramente corrisponde ad un vero e proprio oggetto del set di dati, un medoide, per definizione, deve essere un oggetto presente nel set.

In questa sezione però ci concentreremo solo sul K-means, studiandone ogni sua caratteristica, dato che risulta uno dei più vecchi e più usati algoritmi di clustering e per il suo fondamentale utilizzo nell'algoritmo *TSkmeans* proposto in questa tesi. Al fine di illustrare l'algoritmo nel modo più chiaro possibile, scegliamo di usare come dati di partenza un semplice set di punti (set a due dimensioni). Tutti i vari ragionamenti possono essere estesi anche a differenti tipi di dati essendo il K-means un algoritmo molto generale, come ad esempio documenti o serie temporali.

1.4.1 *L'Algoritmo K-means di Base*

L'algoritmo di clustering K-means è molto semplice, cominciamo dunque con la descrizione della sua tecnica di base. Prima di tutto, vengono scelti K centroidi iniziali, dove K è un parametro specificato dall'utente, ossia, il numero di cluster che vogliamo trovare. Avendo a che fare con un

set di punti i centroidi iniziali verranno scelti fra i punti appartenenti al set. Ogni punto del set è poi assegnato al centroide più vicino e ogni collezione di punti assegnati ad un centroide rappresenterà dunque un cluster. Il centroide di ogni cluster viene poi aggiornato utilizzando la media dei punti assegnati al cluster che rappresenta. Infine ripetiamo i passi di assegnamento e di aggiornamento fino a che nessun punto cambia cluster, oppure in modo del tutto equivalente, fino a che i centroidi non cambiano.

Algorithm 1: K-means di base

1 Seleziona K punti come centroidi iniziali.
2 **repeat**
3 Forma K cluster assegnando ogni punto al centroide più vicino.
4 Ricalcola i centroidi di ogni cluster.
5 **until** *Centroidi non cambiano.*

Il K-means di base è descritto dall'Algoritmo 1, mentre le operazioni che il K-means svolge durante la sua esecuzione sono illustrate nella Figura 3, che mostra come, partendo da tre centroidi, i cluster finali vengono trovati in dieci iterazioni. Ogni sotto figura mostra lo stato dei cluster e dei centroidi all'inizio di ogni iterazione e quali sono i punti che sono stati assegnati ad essi. Il simbolo X viene utilizzato per marcare i centroidi in ogni iterazione e tutti i punti appartenenti allo stesso cluster sono mostrati con il medesimo colore.

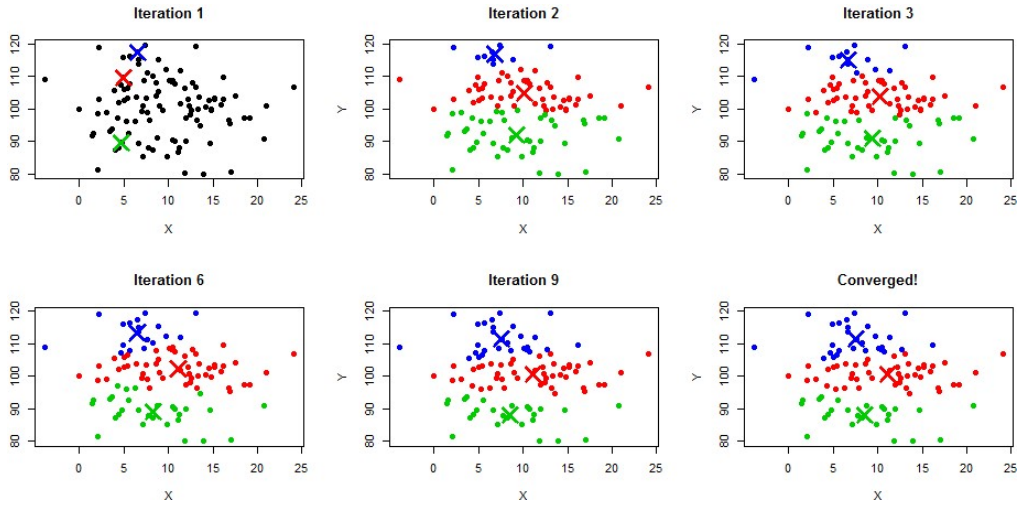

Figura 3: Uso del K-means per trovare tre cluster in un set di punti.

Nella prima iterazione vengono scelti tre centroidi tra i punti del set di dati. Dopo che i punti sono stati assegnati ad un centroide, quest'ultimi vengono aggiornati e si passa all'iterazione successiva. Questo procedimento viene così ripetuto fino a che arriviamo alla decima iterazione dove non essendoci nessun cambiamento l'algoritmo si ferma, i centroidi hanno identificato i loro gruppi di punti.

Per alcune funzioni di prossimità e alcuni tipi di centroidi l'algoritmo K-means converge sempre ad una soluzione, ovvero si raggiunge una situazione in cui i punti non passano più da un cluster ad un altro e quindi anche i centroidi non cambiano. Dato però che la maggior parte delle volte l'algoritmo converge nelle prime iterazioni, spesso viene scelta una condizione di convergenza più debole, ovvero, l'esecuzione viene portata avanti fino a che solo l'1% dei punti cambia cluster.

Successivamente andremo a esaminare più da vicino ogni passo del K-means dando qualche altro dettaglio aggiuntivo, inoltre esamineremo anche la complessità dell'algoritmo rispetto al tempo e allo spazio.

1.4.2 *Misure di Prossimità*

Per assegnare un punto al centroide più vicino, abbiamo bisogno di una misura di prossimità (similarità) che quantifichi il concetto di "vicino" per lo specifico tipo di dati che stiamo considerando. Presenteremo dunque di seguito alcune tra le misure più utilizzate con il K-means, le varie notazioni sono prese dalla Tabella 1.

Per dati in uno spazio Euclideo viene spesso utilizzata la distanza Euclidea nella sua forma standard

$$\text{Distanza Euclidea} = \sqrt{\sum_{i=1}^{n} (X_i - C_i)^2} \tag{1.1}$$

oppure nella sua forma quadratica [8].

$$\text{Distanza Euclidea Quadratica} = \sum_{i=1}^{n} (X_i - C_i)^2 \tag{1.2}$$

Un altra misura di prossimità utilizzata spesso per dati nello spazio Euclideo è la distanza di *Manhattan* (*city block*) [19].

$$\text{Distanza di Manhattan} = \sum_{i=1}^{n} |X_i - C_i| \tag{1.3}$$

Simbolo	Descrizione
X	Un oggetto.
X_i	L'i-esimo attributo dell'oggetto.
$\mathcal{C}_i$	L'i-esimo cluster.
$\mathbb{C}_i$	Il centroide del cluster C_i.
C_i	L'i-esimo attributo del centroide.
m_i	Il numero di oggetti nell'i-esimo cluster.
m	Il numero di oggetti nel set di dati.
n	Il numero di attributi dell'oggetto.
K	Il numero di cluster.

Tabella 1: Tabella delle notazioni

Il coefficiente di correlazione di *Pearson* invece, formalizzato di seguito

$$\text{Correlazione di Pearson} = \frac{\sum_{i=1}^{n}(X_i - \overline{X})(C_i - \overline{C})}{\sqrt{\sum_{i=1}^{n}(X_i - \overline{X})^2}\sqrt{\sum_{i=1}^{n}(C_i - \overline{C})^2}} \quad (1.4)$$

dove $\overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$ e $\overline{C} = \frac{1}{n}\sum_{i=1}^{n} C_i$, può essere utilizzato su vari tipi di dati per la sua invarianza alla posizione e alla scala di quest'ultimi [8].

Un ultimo esempio è dato dalla similiarità del coseno che viene in genere utilizzata quando abbiamo a che fare con dei documenti [19],

$$\text{Similarità del Coseno} = 1 - \frac{\sum_{i=1}^{n} X_i C_i}{\sqrt{\sum_{i=1}^{n} X_i^2}\sqrt{\sum_{i=1}^{n} C_i^2}} \quad (1.5)$$

Ad ogni modo esistono vari altri tipi di misure di prossimità a seconda del tipo di dati che abbiamo a disposizione. Solitamente le misure di similarità usate dal K-means sono relativamente semplici dato che l'algoritmo per ogni iterazione deve ricalcolarsi la similarità di ogni punto rispetto ai vari centroidi. In alcuni casi, tuttavia, quando i dati hanno uno spazio dimensionale basso, avremo un numero di similarità notevolmente minore da calcolare, e questo andrà a velocizzare significativamente l'algoritmo K-means.

1.4.3 *Centroidi e Funzioni Obiettivo*

I centroidi calcolati da un algoritmo di clustering possono variare a seconda della misura di prossimità scelta e in base all'obiettivo del

clustering. Tale scopo è tipicamente espresso da una funzione obiettivo che dipende dalle prossimità fra i punti o dai centroidi dei cluster. Di seguito illustreremo alcuni esempi di funzioni obiettivo

Dati nello spazio Euclideo

Consideriamo dei dati per cui la misura di prossimità sia la distanza Euclidea. Per la nostra funzione obiettivo, che misura la qualità del cluster, usiamo la somma degli errori al quadrato, l'*SSE* (*sum of squarred error*). In altre parole, calcoliamo l'errore di ogni punto, ovvero, la sua distanza Euclidea dal più vicino dei centroidi e successivamente calcoliamo la somma totale degli errori al quadrato. Dati due set differenti di cluster prodotti da due diverse esecuzioni del K-means, preferiamo quello con il minore SSE, per via del fatto che i prototipi (centroidi) di questo clustering sono la migliore rappresentazione dei punti nei loro cluster. L'SSE è formalmente definito come segue:

$$SSE = \sum_{i=1}^{K} \sum_{X \in \mathcal{C}_i} dist(\mathbb{C}_i, X)^2 \tag{1.6}$$

dove *dist* è la distanza Euclidea standard 1.1 tra due oggetti in uno spazio Euclideo e le restanti notazioni sono prese dalla Tabella 1. Con queste assunzioni è possibile dimostrare che il centroide che minimizza l'SSE del cluster è quello medio [3]. Usando sempre la notazione della Tabella 1, il centroide medio dell'i-esimo cluster è definito nel seguente modo.

$$\mathbb{C}_i = \frac{1}{m_i} \sum_{X \in \mathcal{C}_i} X \tag{1.7}$$

I passi 3 e 4 dell'algoritmo K-means tentano dunque di minimizzare l'SSE o più in generale la funzione obiettivo, garantendo di trovare un minimo locale per tale funzione.

Dati in forma di Documenti

Per illustrare che il K-means non è ristretto ai soli dati nello spazio Euclideo, consideriamo dei dati in forma di documento e la similarità del coseno 1.5. Assumiamo che un il set di documenti sia rappresentato da una matrice in cui ogni riga si riferisce ad un documento diverso ed ogni colonna fa riferimento ad un tipo diverso di termine presente nei documenti. Il valore di ogni componente della matrice sarà il numero

di occorrenze di un dato termine all'interno di un documento del set. Il nostro obiettivo è quello di massimizzare la similarità fra i documenti in un cluster e il centroide di quest'ultimo. Questa quantità è chiamata coesione del cluster e per questo obiettivo possiamo affermare che il centroide del cluster viene calcolato, come nel precedente caso, tramite la media. In questo caso dunque possiamo calcolarci una quantità simile all'SSE chiamata coesione totale, formalizzata nel seguente modo.

$$\text{Coesione Totale} = \sum_{i=1}^{K} \sum_{X \in \mathcal{C}_i} cosine(\mathbb{C}_i, X)^2 \tag{1.8}$$

Il Caso Generale

In generale ci sono varie scelte per la funzione di prossimità, per i centroidi e per la funzione obiettivo che possono essere usate dall'algoritmo K-means e che ne garantiscono la convergenza. Le misure di prossimità introdotte nella Sezione 1.4.2 garantiscono tutte la convergenza dell'algoritmo [3].

1.4.4 *Scegliere i Centroidi Iniziali*

La scelta dei centroidi iniziali risulta un passo molto importante per l'algoritmo K-means. Un approccio comune è quello di scegliere i centroidi iniziali in modo causale. Nel caso in cui ripetessimo l'esecuzione del K-means più volte questa scelta porterebbe alla produzione di differenti valori totali dell'SSE, ma i cluster risultanti potrebbero non essere soddisfacenti.

Una tecnica utilizzata per risolvere il problema di scegliere i cluster in modo casuale consiste nell'effettuare molteplici esecuzioni dell'algoritmo, utilizzando ogni volta un diverso set di centroidi generato casualmente, per poi scegliere il set di cluster che risulta avere il minimo SSE. Nonostante la tecnica sia molto semplice, questa strategia potrebbe non funzionare molto bene, per via della sua forte dipendenza dal tipo di data set e dal numero di cluster cercati.

Un altra tecnica per l'inizializzazione dei centroidi consiste nel prendere un campione di punti del set di dati e utilizzarlo con un algoritmo di clustering gerarchico. I K cluster verranno estratti dal clustering gerarchico e i centroidi di questi cluster vengono utilizzati per inizializzare i centroidi del K-means. Con questo approccio, in genere, si ottengono buoni risultati ma può essere messo in atto o solo quando il campione di

dati è relativamente piccolo, da qualche centinaio a poche migliaia di dati (questo perché un clustering gerarchico è abbastanza costoso), oppure quando il parametro K è relativamente piccolo rispetto alla dimensione del campione.

Un'ultima tecnica per selezionare i centroidi iniziali si basa sulla seguente idea. Inizialmente come centroide viene scelto un punto casuale o la media di tutti i punti. Dopodiché, per ogni successivo centroide iniziale, verrà scelto il punto più lontano da ogni altro centroide iniziale selezionato. In questo modo otteniamo un set iniziale di centroidi che non solo risulta generato casualmente ma anche ben separato. Sfortunatamente questo approccio può selezionare dei punti isolati invece che punti provenienti da regioni dense (cluster). Inoltre, è abbastanza costoso calcolare il punto più distante dal set corrente di centroidi iniziali. Per superare questi problemi tale approccio è spesso applicato ad un campione di punti, in modo tale da velocizzare il processo e includere più facilmente dei punti provenienti da regioni dense, almeno che le dimensioni del campione non siano troppo piccole.

1.4.5 *Complessità*

Lo spazio richiesto dal K-means risulta modesto perché vengono salvati solo dati e centroidi necessari per l'esecuzione. In modo specifico, la complessità in spazio è pari a $O((m + K)n)$, dove i vari parametri sono ripresi dalla Tabella 1. Il tempo richiesto dal K-means è anch'esso modesto, praticamente lineare nel numero di punti (dati). In particolare il tempo richiesto è $O(I * K * m * n)$, dove I è il numero di iterazioni richieste per la convergenza. Come già accennato, I è spesso relativamente piccolo e può essere limitato, dato che tipicamente i cambiamenti avvengono nelle prime iterazioni. Il K-means dunque è lineare per m, il numero di punti (dati), ed è inoltre efficiente purché venga scelto un numero di cluster K significativamente minore di m.

1.5 K-MEANS: DEBOLEZZE E PUNTI DI FORZA

In questa sezione andremo a discutere brevemente alcune delle principali debolezze e punti di forza dell'algoritmo di clustering K-means. Come prima cosa descriveremo due problemi dell'algoritmo e in seguito andremo a tirare le somme sulle sue caratteristiche.

Cluster Vuoti

Uno dei problemi del K-means di base riguarda i cluster vuoti che possono essere ottenuti se durante la fase di assegnazione nessun punto (dato) viene assegnato ad un cluster. Se questo accade, è necessario trovare una strategia per scegliere una sostituzione per il centroide, dato che altrimenti l'SSE risulterebbe più grande del necessario e in ogni caso non sarebbe possibile calcolare il centroide del cluster vuoto. Un approccio è quello di scegliere il punto più lontano dal suo centroide, questo infatti va ad eliminare il punto che più contribuisce all'SSE totale. Un altro approccio è quello di scegliere un sostituto per il centroide dal cluster che ha l'SSE più alto, provocando la divisione del cluster e riducendo l'SSE totale del clustering.

Dati Anomali

Quando è usato il criterio dell'SSE i dati anomali possono influenzare i cluster trovati. In particolare, quando sono presenti dei dati anomali, i centroidi dei cluster risultanti potrebbero non essere così rappresentativi come lo sarebbero in condizioni normali e anche l'SSE risulterà più alto. Proprio per questo, è spesso utile trovare i dati anomali prima dell'esecuzione dell'algoritmo ed eliminarli. Tuttavia è importante specificare che in alcune applicazioni i dati anomali non andrebbero eliminati in quanto utili, come ad esempio per risultati di analisi finanziarie dove questi dati potrebbero essere dei vantaggiosi clienti e quindi dati interessanti. Alternativamente i dati anomali possono essere identificati dopo l'esecuzione dell'algoritmo. Per esempio, possiamo tener traccia del contributo all'SSE di ciascun punto ed andare ad eliminare quelli che contribuiscono al suo innalzamento, specialmente dopo molteplici esecuzioni. Inoltre, possiamo procedere all'eliminazione dei cluster piccoli dato che in genere rappresentano gruppi di dati anomali.

I maggiori punti di forza dell'algoritmo K-means sono invece la sua semplicità e la sua grande versatilità di uso su vari tipi di dati. Inoltre, risulta un algoritmo molto efficiente anche se spesso risultano necessarie molteplici esecuzioni, ed alcune sue varianti mirano ancora di più ad ottimizzare questa sua efficienza. Il K-means però non è adatto a tutti i tipi di dati, ad esempio non può gestire correttamente cluster non globulari o con differenti dimensioni e densità, tuttavia può in genere trovare sotto cluster puri se viene indotto a cercare un numero sufficientemente grande di cluster. Come abbiamo accennato, il K-means ha dei problemi con i

dati anomali e quindi risulta necessario saperli gestire correttamente, ed infine, il K-means è ristretto a quei dati che hanno una nozione di centro (centroide). Una tecnica molto affine, il K-medoids clustering, non ha queste restrizioni, ma è molto più costoso in termini di prestazioni.

2

CLUSTERING PER SERIE TEMPORALI

Un campo speciale della clustering analysis è quello basato sull'analisi di serie temporali. Per definizione, una serie temporale è una sequenza di osservazioni ordinate rispetto al tempo, che in genere esprime la dinamica di un certo fenomeno o parametro nel tempo. Le serie temporali vengono studiate sia per interpretare un fenomeno, individuando componenti di trend, ciclicità, stagionalità e accidentalità, sia per prevederne il suo andamento futuro [2]. Sono esempi di serie temporali l'intensità del traffico su una strada nell'arco di un anno oppure l'andamento mensile del prezzo di un determinato prodotto.

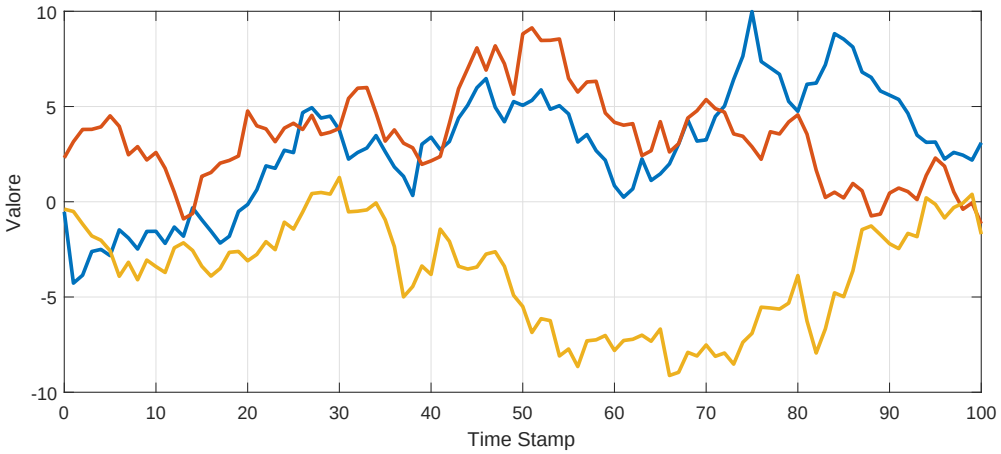

Figura 4: Esempio di serie temporali.

Esse rivestono molto interesse per via della loro ubiquità in varie aree che spaziano dalla scienza, all'ingegneria, all'economia, alla medicina e in molti altri campi. In genere le serie temporali sono rappresentate utilizzando grafici in cui il tempo viene posto sulle ascisse e le osservazioni sulle ordinate, un esempio viene mostrato nella Figura 4.

Le serie temporali sono un tipo di dati che per loro natura hanno grandi dimensioni essendo composte in genere da un alto numero di osservazioni e molto spesso fanno parte a loro volta di set di dati molto grandi. Inoltre, per la loro natura temporale richiedono continui aggiornamenti per rappresentare in modo corretto il fenomeno che descrivono. Le serie temporali possono essere viste anche come oggetti singoli caratterizzati da sequenze di osservazioni ordinate rispetto al tempo. Essendo dei tipi di dati aventi grandi dimensioni molto spesso mettono a dura prova l'esecuzione degli algoritmi di clustering per la loro analisi, sia in termini di spazio che di tempo, ma nonostante questi problemi il loro studio può portare a scovare informazioni interessanti.

Applicare le tecniche della *cluster analysis* alle serie temporali è particolarmente vantaggioso per i seguenti motivi:

- I database di serie temporali contengono informazioni preziose che possono essere ottenute mediante la scoperta di pattern. Il clustering dunque è una soluzione comune utilizzata per la ricerca di questi pattern.

- I database di serie temporali sono spesso molto grandi e non possono essere gestiti molto bene dalle persone senza l'ausilio dei computer. Perciò molti utenti preferiscono avere a che fare con set di dati strutturati e con una dimensione ridotta rispetto a set con grandi dimensioni.

- Il clustering di serie temporali è l'approccio più usato come tecnica di esplorazione e viene spesso utilizzato come sotto routine di complessi algoritmi di data mining, come ad esempio quelli finalizzati alla classificazione, alla scoperta di regole, all'indicizzazione e al rilevamento di anomalie.

- Rappresentare le strutture dei cluster di serie temporali con immagini visive può aiutare gli utenti a capire velocemente la struttura dei dati, dei cluster, delle anomalie e di altre regolarità nei set di dati.

Nel resto del capitolo parleremo del clustering per serie temporali, introducendo nuovi concetti e riprendendone alcuni presentati nel Capitolo 1. In particolare, ci soffermeremo a descrivere alcune applicazioni reali di tecniche di clustering per questa categoria di dati, presenteremo come si classificano tali tecniche, introdurremo una nuova misura di prossimità, il DTW, pensata appositamente per il suo utilizzo con serie temporali

e vedremo in generale quali sono gli algoritmi più utilizzati di questa categoria. Gli argomenti trattati in questo capitolo sono in parte ripresi da *Time-series clustering, A decade review* (2015) [2].

2.1 INTRODUZIONE AL CLUSTERING PER SERIE TEMPORALI

Dato un set di n serie temporali $S = \{S_1, S_2, \ldots, S_n\}$, il processo di partizionamento non supervisionato di S in un insieme di cluster $C = \{C_1, C_2, \ldots, C_k\}$, creato in modo tale che serie temporali omogenee vengano raggruppate insieme basandosi su una certa misura di similarità, viene chiamato clustering per serie temporali. Ogni C_i rappresenta un cluster, mentre $S = \cup_{i=1}^{k} C_i$ e $C_i \cap C_j = \emptyset$ per $i \neq j$.

Il clustering di serie temporali, per una serie di motivi, risulta molto impegnativo. Prima di tutto per via della grandezza dello spazio che i set di dati basati su serie temporali occupano per loro natura, tali dimensioni, infatti, sono spesso molto più grandi della memoria disponibile e conseguentemente devono essere salvati su dischi rigidi. Questo causa dunque un decremento esponenziale delle prestazioni del processo di clustering. Un altro problema è causato dal fatto che le serie temporali sono spesso molte estese dal punto di vista dimensionale, contenendo un grandissimo numero di osservazioni, e questo mette in difficoltà molti algoritmi di clustering, rendendo la loro esecuzione molto lenta. Infine, un ulteriore complicazione riguarda le misure di prossimità usate per ottenere i cluster. Per ricavare quest'ultimi infatti, è necessario ricercare le similarità fra le varie serie temporali disponibili e in genere questo processo viene eseguito rispetto alla loro completa interezza. Tale processo è conosciuto come "confronto sull'intera sequenza" (*whole sequence matching*), dove durante i calcoli delle distanze viene considerata l'intera lunghezza delle serie temporali. Tuttavia, questo processo risulta molto complesso, dato che le serie temporali includono spesso anomalie, rumori o dilatazioni temporali. Questi problemi comuni hanno reso la scoperta di nuove misure di prossimità una vera e propria sfida per i ricercatori.

2.2 APPLICAZIONI DEL CLUSTERING PER SERIE TEMPORALI

Il clustering di serie temporali viene utilizzato in larga parte per la scoperta di pattern interessanti all'interno di set composti da serie temporali. Questo compito divide i vari tipi di clustering in due categorie. Il primo gruppo è quello che consiste nella ricerca di pattern che appaiono fre-

quentemente in un set di dati. Mentre il secondo gruppo, comprende i metodi per la scoperta di pattern anomali o inaspettati.

In particolare, la ricerca di cluster di serie temporali può essere vantaggiosa in diversi contesti per rispondere ad alcuni problemi pratici. Di seguito vengono riportati vari esempi.

- *Anomalie, novità o rilevamenti di discordanza*: i metodi per il rilevamento di anomalie sono utilizzati per scoprire pattern insoliti o inaspettati che si verificano in set di dati in modo sorprendente. Per esempio in un database basato su sensori, i risultati di un clustering per serie temporali può essere molto interessante per la scoperta di determinati eventi particolari.

- *Riconoscimento di cambiamenti dinamici*: per l'individuazione di una correlazione nell'andamento di alcune serie temporali. Per esempio, in database finanziari, può essere utilizzato per cercare le società che hanno un simile andamento di quote azionarie.

- *Predizioni e consigli*: alcune tecniche ibride che combinano il clustering all'approssimazione di funzioni possono aiutare l'utente a predire alcuni eventi rilevanti. Ad esempio, in database scientifici, possono affrontare problemi di carattere meteorologico, come la previsione della temperatura o della pioggia in un determinato giorno.

- *Scoperta di modelli*: per ricercare i gruppi di pattern più interessanti nei database. Per esempio, in database per il marketing, possono essere scoperti differenti pattern giornalieri sulle vendite di un determinato prodotto in un negozio.

2.2.1 *Un Esempio Pratico*

Presentiamo infine un esempio di problema pratico risolvibile grazie ad algoritmi di clustering per serie temporali presentato in [10].

Supponiamo di avere a disposizione un set di immagini che raffigurano foglie di specie diverse (aventi quindi forme differenti), e di doverle raggruppare fra loro, trovando dunque dei cluster, in base alla loro forma, per poterle successivamente classificare in modo automatico attraverso appositi algoritmi di classificazione. Questo problema può essere risolto utilizzando un algoritmo di clustering per serie temporali. L'idea iniziale è quella di utilizzare un metodo per l'elaborazione delle immagini per

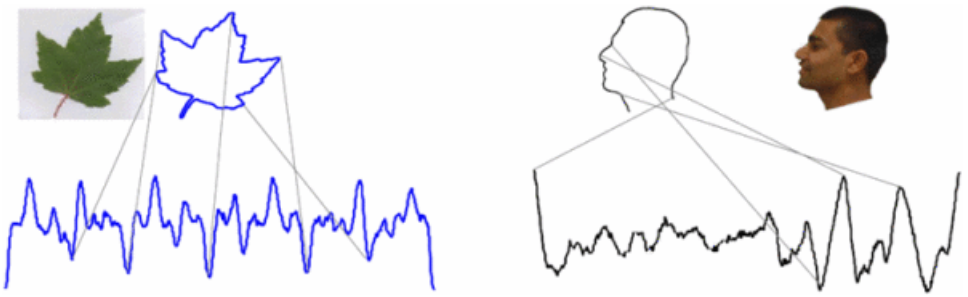

Figura 5: Trasformazione del perimetro di un immagine in una serie temporale.

convertire il perimetro di ogni foglia, raffigurata in un immagine, in una sequenza di valori, e utilizzare quest'ultimi per la creazione di una serie temporale. Tale processo viene mostrato in Figura 5. Dopodiché una volta che abbiamo convertito ogni forma delle foglie disponibili in una serie temporale possiamo usare il set risultante di serie temporali come input per un algoritmo di clustering. L'algoritmo infatti ci restituirà un insieme di cluster, dove ogni gruppo è formato da foglie aventi forme simili fra loro (rappresentate in forma di serie temporali). Questi cluster finali possono essere poi utilizzati per classificare correttamente tutto il set di foglie. Un altro esempio simile al precedente consiste invece nel dover raggruppare fra loro profili simili di persone differenti per scoprire quali persone hanno facce somiglianti, tale caso è riportato sempre in Figura 5.

2.3 CLASSIFICAZIONE DEL CLUSTERING PER SERIE TEMPORALI

Il clustering per serie temporali, a seconda di come viene eseguito, può essere classificato in tre categorie: clustering sull'intera sequenza (*whole sequence clustering*), clustering su sotto sequenza (*subsequence clustering*) e clustering sul punto temporale (*time point clustering*) [2], come mostrato in Figura 6.

2.3.1 *Whole Sequence Clustering*

Il clustering sull'intera sequenza (*whole sequence clustering*) mira a scoprire le serie temporali che hanno pattern simili rispetto all'intera sequenza temporale su cui sono definite, raggruppandole in cluster differenti a seconda della loro somiglianza basata su una misura di similarità. In generale però, questo tipo di clustering non prende in considerazione i

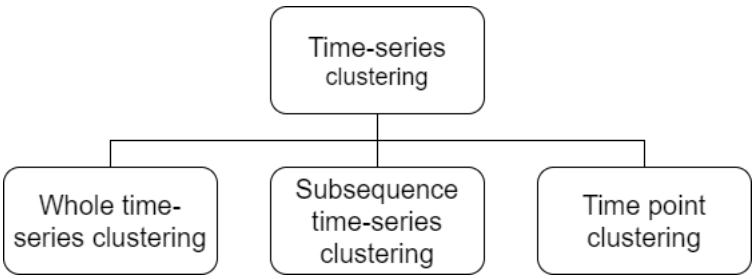

Figura 6: Classificazione delle serie temporali.

possibili sottospazi (sotto sequenze) in comune e similari che possono presentare i dati basati su serie temporali. Gli algoritmi di clustering basati sulla distanza Euclidea 1.1 o sul coefficiente di correlazione di Pearson 1.4 sono spesso utilizzati per affrontare il clustering per questa categoria. Tuttavia, questi algoritmi non possono gestire efficientemente la mutevolezza e le dilatazioni delle serie temporali.

2.3.2 *Subsequence Clustering*

Il clustering su sotto sequenze (*subsequence clustering*) cerca di identificare differenti intervalli di tempo (sottospazi), ovvero singoli segmenti, delle intere sequenze su cui sono definite le serie temporali, per poi eseguire gli algoritmi di clustering rispetto a questi intervalli. Questo approccio viene utilizzato quando abbiamo a che fare con serie temporali molto lunghe e che presentato pattern simili o dati rilevanti solo in determinate sotto sequenze o sottospazi. Molti algoritmi basati su questo approccio, però, possono solo risolvere problemi che sono caratterizzati da alcuni parametri predefiniti la cui variabilità è abbastanza ridotta. Recentemente, nuovi studi effettuati da vari ricercatori hanno portato alla scoperta di algoritmi di clustering su sotto sequenze che consentono risultati soddisfacenti per vari tipi di problemi, come l'algoritmo che presenteremo in questa tesi, il TSkmeans.

2.3.3 *Time Point Clustering*

Un altra categoria di clustering per serie temporali è quella basata sui punti temporali, che consiste nel generare dei cluster in base ad una combinazione tra la prossimità dei punti (coppie istante di tempo - valore, che caratterizzano una serie temporale) rispetto al tempo e alla similarità dei loro corrispondenti valori. Questo approccio viene in genere applicato

ad una singola serie temporale e ha come obiettivo la ricerca di cluster di punti temporali, inoltre, non tutti i punti necessitano di essere assegnati ad un cluster, dato che alcuni di essi, durante il processo di clustering, possono essere considerati come rumori [2].

2.4 MISURE DI PROSSIMITÀ PER SERIE TEMPORALI

Esistono varie misure di prossimità che possono essere utilizzate per valutare la similarità fra serie temporali, e le tecniche di clustering per questa tipologia di dati ne fanno un uso molto massiccio. In un clustering tradizionale, le distanze tra oggetti statici vengono calcolate in base alla corrispondenza dei loro attributi, ma in un clustering per serie temporali, la distanza è calcolata spesso in modo più approssimato. In particolare, al fine di confrontare serie temporali con irregolari lunghezze e intervalli di tempo, è estremamente importante determinare in modo adeguato la similarità fra esse. Esistono diverse misure di prossimità progettate per specifici tipi di similarità fra serie temporali. Le misure di prossimità più famose e utilizzate per serie temporali sono la distanza Euclidea 1.1, che abbiamo già introdotto nel Capitolo 1, e il *Dynamic Time Warping* (DTW) che presenteremo tra poco in questa sezione.

Uno dei modi più semplici e utilizzati per calcolare la distanza fra due serie temporali consiste nel considerarle "univariate", e calcolare la distanza attraverso i punti temporali. Una serie temporale univariata è la più semplice forma di dato temporale e consiste in una sequenza di numeri reali raccolti ad intervalli di tempo regolari, dove ogni numero rappresenta un valore. Se prendiamo l'insieme $S_i = \{s_{i1}, \ldots, s_{it}, \ldots, s_{iT}\}$, che rappresenta una serie temporale di lunghezza T, la distanza fra due serie temporali è definita di seguito:

$$\mathrm{dist}(S_i, S_j) = \sum_{t=1}^{T} \mathrm{dist}(s_{it}, s_{jt}) \tag{2.1}$$

quindi la 2.1 è la somma delle distanze fra i punti che definiscono le rispettive serie temporali.

In generale però, la scelta della giusta misura di prossimità da utilizzare durante il processo di clustering dipende dalle caratteristiche delle serie temporali che dobbiamo analizzare, dalla loro lunghezza e dall'obiettivo che vogliamo raggiungere attraverso il processo di clustering. Tipicamente, esistono due obiettivi diversi che rispettivamente richiedono approcci differenti per il calcolo della distanza fra serie temporali.

Di seguito andremo a parlare brevemente di questi due approcci e mostreremo quali misure di prossimità vengono utilizzate per questi metodi, soffermandoci in particolare sulla misura DTW.

2.4.1 *Prossimità Rispetto al Tempo*

In questo approccio, la similarità fra serie temporali viene calcolata comparando i valori che assumono in ogni istante di tempo. Tutte le misure di prossimità descritte nella Sezione 1.4.2 possono essere utilizzate per raggiungere questo obiettivo. In particolare le misure più utilizzate risultano la distanza Euclidea, sia nella suo forma standard 1.1 sia nella sua forma quadratica 1.2, e la distanza basata sul coefficiente di correlazione di Pearson 1.4.

2.4.2 *Prossimità Rispetto alla Forma*

Le occorrenze rispetto agli istanti di tempo non sono importanti se vogliamo trovare serie temporali che siano simili rispetto alla loro forma. Per questo obiettivo vengono utilizzate delle misure più "elastiche" come ad esempio il *Dynamic Time Warping* (DTW). Con questo approccio dunque, i cluster di serie temporali che hanno un andamento simile, sono costruiti a prescindere dai punti temporali, ad esempio, l'andamento dei prezzi delle azioni di due società diverse possono risultare simili se le due serie temporali che li rappresentato hanno pattern di andamento somiglianti ma risultano indipendenti dall'intervalli di tempi in cui si manifestano. Inoltre, risulta evidente che le misure di prossimità rispetto al tempo sono un caso speciale di quelle relative alla forma.

Dynamic Time Warping

Il *Dynamic Time Warping*, o DTW, è un algoritmo che permette l'allineamento tra due sequenze e che può portare ad una misura di distanza tra le due sequenze allineate. Tale algoritmo è particolarmente utile per trattare sequenze in cui singole componenti hanno caratteristiche che variano nel tempo e per le quali la semplice espansione o compressione lineare delle due sequenze non porta risultati soddisfacenti. È stato utilizzato in diversi campi di applicazione, dal riconoscimento vocale alla cluster analysis. In generale, il DTW è un metodo che permette di trovare una corrispondenza ottima tra due sequenze, ad esempio due serie temporali, attraverso una distorsione non lineare rispetto al tempo [7]. La Figura 7

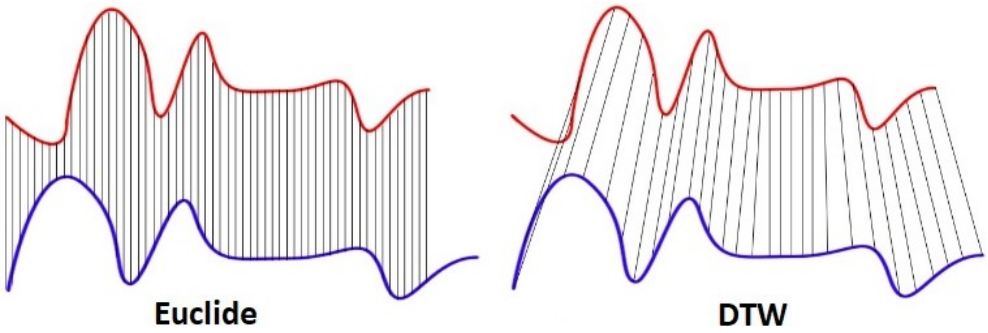

Figura 7: Differenza fra la distanza Euclidea e la distanza DTW.

mostra la differenza fra la distanza DTW e quella Euclidea 1.1. Purtroppo, data la complessità quadratica rispetto alla lunghezza delle serie temporali, il DTW può risultare molto costoso per effettuare il clustering di serie temporali, ma in alcuni contesti può dare risultati molto soddisfacenti rispetto ad altre misure, esistono inoltre alcune varianti che ne migliorano le prestazioni.

Date due serie temporali, definite da $Q = \{q_1, \ldots, q_i, \ldots, q_n\}$ e $R = \{r_1, \ldots, r_i, \ldots, r_m\}$, il DTW allinea le due serie in modo tale da minimizzare la loro differenza (distanza cumulativa). A questo scopo viene costruita una matrice $n \times m$ dove l'elemento (i, j) contiene la distanza $d(q_i, r_j)$ tra i punti q_i e r_i. In questa matrice viene normalmente utilizzata la distanza Euclidea 1.1. Nella matrice, un *warping path*, $W = \{w_1, \ldots, w_i, \ldots, w_K\}$, dove $max(m, n) \leqslant K \leqslant m + n - 1$, rappresenta un set di elementi della matrice che soddisfano tre vincoli: condizione di limite, continuità e monotonicità. La condizione di limite richiede che il *warping path* inizi e finisca nelle celle posizionate negli angoli opposti della matrice, ovvero $w_1 = (1, 1)$ e $w_K = (m, n)$. Il vincolo di continuità restringe i passi ammissibili alle sole celle adiacenti. Il vincolo di monotonicità forza i punti del *warping path* ad essere monotonamente distanziati nel tempo. Il *warping path* che ci interessa trovare è quello che minimizza la distanza cumulativa tra le due serie temporali. In termini matematici:

$$d_{DTW} = min \frac{\sum_{k=1}^{K} w_k}{K} \tag{2.2}$$

Per trovare efficacemente questo percorso, possiamo utilizzare una tecnica di programmazione dinamica, per valutare la seguente ricorrenza, che definisce la distanza cumulativa come la somma della distanza dell'e-

lemento corrente e il minimo delle distanze cumulative degli elementi adiacenti [8], ovvero:

$$d_{cum}(i,j) = d(q_i, r_j) + min\{d_{cum}(i-1, j-1), d_{cum}(i-1, j), d_{cum}(i, j-1)\} \tag{2.3}$$

2.5 ALGORITMI DI CLUSTERING PER SERIE TEMPORALI

In questa sezione parleremo delle tipologie di algoritmi dedicati alle serie temporali. Per le tecniche già descritte nel Capitolo 1 ci limiteremo a parlare di come si comportano se applicati a dati in forma di serie temporali. In generale possiamo classificare gli algoritmi per serie temporali in quattro gruppi: gerarchici, partizionali, basati su un modello o sulla densità.

2.5.1 *Clustering Gerarchico*

Nel clustering gerarchico di serie temporali vengono generate gerarchie annidate di gruppi simili basati su matrici di distanza fra serie temporali. In generale questi tipi di algoritmi generano dei cluster di qualità bassa perchè non possono aggiustare i gruppi creati dopo averli uniti o divisi con altri. Proprio per questo solitamente questi algoritmi vengono combinati con altri ottenendo delle soluzioni ibride. Il clustering gerarchico però a differenza di molti altri algoritmi non richiede come parametro iniziale il numero di cluster da trovare e questo è indubbiamente uno dei suoi punti di forza. Anche nel caso di serie temporali questa particolarità risulta un grande punto di forza dato che spesso è complesso definire il numero di cluster per problemi reali. Inoltre, il clustering gerarchico, non avendo bisogno di prototipi, ha l'abiltà di gestire serie temporali di lunghezze differenti attraverso l'utilizzo di misure di prossimità elastiche come quella basata sul DTW. Tuttavia, non è in grado di gestire efficacemente serie temporali di grandi dimensioni per la sua complessità quadratica e dunque viene spesso ristretto ad operare su piccoli set di dati per la sua bassa scalabilità.

2.5.2 *Clustering Partizionale*

Come visto nel capitolo precedente i principali algoritmi utilizzati in questa categoria sono il K-means, il K-medoids, a cui vengono aggiunti il Fuzzy C-means e il Fuzzy C-medoids, due algoritmi che si basano sui

fuzzy set. Gli algoritmi K-means e K-medoids necessitano come parametro iniziale il numero di cluster da ricercare, questo risulta un loro lato negativo e nel caso specifico di serie temporali, che hanno di natura grandi dimensioni, è un aspetto ancora più grave per la difficoltà di trovare tecniche efficienti per ricavare il numero corretto di cluster. Tuttavia, essendo algoritmi estremamente preformanti rispetto per esempio al clustering gerarchico, risultano tecniche molto indicate per l'analisi di serie temporali e dunque sono spesso utilizzati in molte applicazioni reali che si basano su questi tipi di dati. In generale, se applicati a serie temporali, i precedenti algoritmi insieme ai clustering fuzzy, per la loro dipendenza dai prototipi e dai loro metodi di aggiornamento, sono algoritmi che tendono ad essere più compatibili con misure di prossimità rispetto al tempo e preferibilmente con serie aventi la stessa lunghezza. Difatti definire prototipi per misure di prossimità rispetto alla forma non risulta un processo semplice.

2.5.3 *Clustering Basato su Modelli*

I metodi basati su modelli ipotizzano un modello per ciascuno dei cluster e trovano la migliore disposizione dei dati rispetto al determinato modello. Un algoritmo basato su un modello può localizzare i cluster costruendo una funzione di densità che riflette la distribuzione spaziale dei punti associati ai dati. Esso consente anche di determinare automaticamente il numero di cluster basandosi su statistiche standard, tenendo in considerazione il rumore e ottenendo, pertanto, metodi di clustering robusti. In generale, il clustering su modelli ha due svantaggi: ha bisogno di alcuni paramatri iniziali che sono basati su assunzioni dell'utente, che potrebbero dunque essere false e di conseguenza portare a generare dei cluster inaccurati, e secondariamente ha un esecuzione lenta su set di dati molto grandi.

2.5.4 *Clustering Basato sulla Densità*

Nel clustering basato sulla densità i cluster sono sottospazi densi di oggetti che sono separati dai sottospazi in cui gli oggetti hanno bassa densità. Uno degli algoritmi più famosi di questa tipologia è l'algoritmo DBSCAN, dove un cluster viene esteso se i sui vicini sono densi. Gli algoritmi di questa tipologia non sono tuttavia molto utilizzati con dati in forma di serie temporali per la loro elevata complessità.

3

L'ALGORITMO TIME SERIES K-MEANS

In questo capitolo, presenteremo un nuovo algoritmo di clustering per serie temporali chiamato *Time Series K-means* (TSkmeans), proposto per la prima volta in un interessante articolo da X. Huang et al. (2016) [1], un gruppo di ricercatori delle Università di Jiaotong e Hong Kong e dell'Istituto di tecnologia di Harbin in Cina. L'algoritmo presentato è basato sulla tecnica K-means e sfrutta l'approccio del clustering su sotto sequenze (o sottospazi) per serie temporali, assegnando un peso maggiore agli intervalli di tempo più rilevanti, al fine di ottenere cluster utili e significativi.

Offriremo un'implementazione dell'algoritmo utilizzando il software MATLAB [11] e discuteremo delle caratteristiche principali del TSkmeans. In particolare, parleremo di come è stata sviluppata la sua funzione obiettivo per il processo di clustering, verranno descritte tutte le procedure iterative derivanti dalla funzione obiettivo e che vengono utilizzate durante l'esecuzione dell'algoritmo, parleremo della gestione dei pesi e ne spiegheremo il loro uso, ed infine, analizzeremo la complessità computazionale dell'algoritmo. Inoltre, forniremo anche un semplice esempio pratico per mostrare come si comporta l'algoritmo TSkmeans durante la sua esecuzione, prendendo in riferimento, per semplicità, un set molto piccolo di serie temporali definite su una sequenza temporale abbastanza ristretta.

3.1 CARATTERISTICHE DELL'ALGORITMO TSKMEANS

In modo simile ai tradizionali algoritmi di clustering che operano sull'intera sequenza, l'algoritmo TSkemans è in grado di scoprire iterativamente i sottospazi dell'intera sequenza e successivamente eseguire il clustering delle serie temporali basandosi su quest'ultimi. Seguendo questa idea, l'algoritmo TSkmeans, utilizzando l'approccio del K-means, assegna un peso specifico ad i vari istanti di tempo (*time stamp*) che caratterizzano le

serie temporali in esame, ovvero un valore che specifica l'importanza di un determinato istante di tempo per il processo di clustering.

Sebbene siano già stati proposti in passato degli algoritmi pesati basati sul K-means utilizzando vari metodi di attribuzione del peso, come ad esempio il W-k-Means [9], questi algoritmi puntano a generare dei cluster per dati formati da attributi che non hanno un ordine cronologico, a differenza delle serie temporali. Per esplorare efficacemente le informazioni della sequenza associata a dati in forma di serie temporale, l'algoritmo TSkmeans cerca di assegnare pesi simili ad istanti di tempo adiacenti (*smooth weights*), per cercare di rendere più significativo durante le operazioni di clustering l'ordine cronologico dei dati presenti nelle serie e di conseguenza rendere i sottospazi scoperti molto più rilevanti per il clustering di serie temporali. L'algoritmo TSkmeans offre quindi ottimi risultati quando utilizzato su set di serie temporali che presentano una forte correlazione in specifiche sotto sequenze.

3.2 LA FUNZIONE OBIETTIVO

Per definire la funzione obiettivo su cui l'algoritmo fa affidamento, definiremo di seguito le principali strutture utilizzate dall'algoritmo TSkmeans. Sia $X = \{X_1, X_2, \ldots, X_n\}$ un set di n serie temporali e k il numero di cluster che vogliamo trovare. Ogni serie temporale $X_i = \{x_{i1}, x_{i2}, \ldots, x_{im}\}$ è caratterizzata da m valori corrispondenti ad m istanti di tempo. Il set X delle serie temporali può essere rappresentato da una matrice $n \times m$ dove l'elemento x_{ij} indica il valore associato al j-esimo istante di tempo relativo alla i-esima serie temporale. La matrice di appartenenza U, è invece una matrice binaria di dimensioni $n \times k$, dove l'elemento $u_{ip} = 1$ indica che la serie temporale i è stata assegnata al cluster p, altrimenti, se $u_{ip} = 0$, indica che la serie i non è stata assegnata al cluster p. I centroidi dei k cluster sono rappresentati da un set di k vettori $Z = \{Z_1, Z_2, \ldots, Z_k\}$, che può anch'esso essere rappresentato attraverso una matrice $k \times m$, dove ogni Z_i costituisce la serie temporale, definita come $Z_i = \{z_{i1}, z_{i2}, \ldots, z_{im}\}$, che descrive il centroide del cluster i, mentre $W = \{W_1, W_2, \ldots, W_k\}$ è un set di k vettori che rappresenta i pesi degli istanti di tempo per ogni cluster, rappresentabile con una matrice di dimensioni $k \times m$. Il valore dell'elemento w_{pj} indica il peso associato al j-esimo istante di tempo per il p-esimo cluster.

Con le strutture precedentemente definite possiamo ora mostrare la funzione obiettivo dell'algoritmo TSkmeans, ovvero, la funzione che il nostro metodo cercherà di minimizzare e rappresentante l'obiettivo che

il processo di clustering vuole raggiungere. La funzione obiettivo del TSkmeans è formulata come segue:

$$P(U, Z, W) = \left.\begin{aligned} &\sum_{p=1}^{k}\sum_{i=1}^{n}\sum_{j=1}^{m} u_{ip} w_{pj} (x_{ij} - z_{pj})^2 + \\ &+\frac{1}{2}\alpha \sum_{p=1}^{k}\sum_{j=1}^{m-1} (w_{pj} - w_{pj+1})^2 \end{aligned}\right\} \begin{aligned} &\text{Prima Parte} \\ &\text{Seconda Parte} \end{aligned} \tag{3.1}$$

ed è soggetta a:

$$\begin{cases} \sum_{p=1}^{k} u_{ip} = 1, \quad u_{ip} \in \{0, 1\} \\ \sum_{j=1}^{m} w_{pj} = 1, \quad 0 \leqslant w_{pj} \leqslant 1 \end{cases} \tag{3.2}$$

Il parametro α è un valore che verrà passato in input all'algoritmo, e viene utilizzato per bilanciare gli effetti della funzione obiettivo tra la dispersione delle serie temporali all'interno dei cluster e l'uniformità dei pesi associati a istanti di tempo adiacenti, tale che, la similarità dei pesi tra istanti di tempo adiacenti aumenti con l'aumentare del valore di α. In seguito, quando parleremo delle funzionalità dei pesi, andremo a specificare come l'algoritmo si comporta al variare del valore di α.

La funzione obiettivo 3.1 del TSkmeans si divide sostanzialmente in due parti. La prima parte della 3.1 punta a minimizzare la somma pesata della dispersione di tutti i cluster, tentando dunque di generare dei cluster coesi e compatti, dove $w_{pj}(x_{ij} - z_{pj})^2$ rappresenta la misura di prossimità che l'algoritmo TSkmeans usa per confrontare le serie temporali con i centroidi dei cluster, ovvero, una distanza Euclidea quadratica 1.2 e pesata. La seconda parte della 3.1 punta invece a cercare di rendere simili i pesi per istanti di tempo adiacenti, per cercare di valorizzare maggiormente il carattere cronologico delle serie temporali e per permettere la corretta ricerca di sotto sequenze omogenee. In sintesi, durante il processo di clustering tale funzione cerca, in modo simultaneo, di minimizzare la dispersione all'interno dei cluster e di rendere omogenei i pesi di istanti di tempo adiacenti.

3.3 L'ALGORITMO TSKMEANS

L'algoritmo TSkmeans, essendo basato sul K-means, seguirà in generale le operazioni descritte nell'Algoritmo 1. Inizialmente dunque, una volta ricevuto in input il numero di cluster k da produrre, andrà a selezionare

Algorithm 2: Time Series K-means (TSkmeans)

Input : $X = \{X_1, X_2, \ldots, X_n\}$, k, α.
Output: U, Z, W.

1 **Inizializzazione**: Sceglie in modo casuale i centroidi $Z^0 = \{Z_1, Z_2, \ldots, Z_k\}$ e i pesi $W^0 = \{W_1, W_2, \ldots, W_k\}$ iniziali.
2 **repeat**
3 Fissato Z, W, ricava la matrice di appartenenza U.
4 Fissato U, W, ricava la matrice dei centroidi Z.
5 Fissato U, Z, ricava la matrice dei pesi W.
6 **until** *Le assegnazioni ai cluster non cambiano.*

in modo casuale dal set di input k centroidi iniziali, che in questo caso sono rappresentati in forma di serie temporali. Successivamente il TSkmeans entrerà nel suo ciclo principale ed inizierà ad assegnare ogni serie temporale al centroide più vicino ed a ricalcolare in seguito i centroidi di ogni cluster. L'algoritmo terminerà nel momento in cui verrà raggiunta la convergenza, ovvero, nel momento in cui gli assegnamenti ai cluster non cambiano. La procedura generale del TSkmeans viene descritta dall'Algoritmo 2. Di seguito viene riportata anche l'implementazione in MATLAB dell'Algoritmo 2.

Listing 3.1: ts_kmeans.m

```
function [U, Z, W] = ts_kmeans(X, k, alpha, init_Z)
% inizializzazione
[n, m] = size(X);
[Z, W, H, Aeq, beq, lb, ub] = initialize(X, k, alpha, n, m);
if exist('init_Z', 'var')
   Z = init_Z;
end
U0 = zeros(n, k);
convergence = false;

% loop principale
while ~convergence
   % ricavo la matrice U
   [U] = solve_U(X, Z, W, k, n);
   % ricavo la matrice Z
   [Z] = solve_Z(X, U, k, m);
   % ricavo la matrice W
   [W] = solve_W(X, U, Z, k, n, m, H, Aeq, beq, lb, ub);
```

```
19
20    % controllo il criterio di convergenza
21    if isequal(U0, U)
22       convergence = true;
23    else
24       U0 = U;
25    end
26 end
27 end
```

L'implementazione in MATLAB aggiunge ai parametri di input la variabile "*init_Z*", utilizzata per passare all'algoritmo TSkmeans un set già costruito ed utilizzabile di centroidi iniziali, che devono però essere ricavati in ogni caso dal set di dati disponibile. Questo risulterà utile quando andremo a testare le prestazioni dell'algoritmo confrontandolo con altre tecniche di clustering. A questo punto parleremo dei passi svolti dall'algoritmo e mostreremo la loro implementazione in MATLAB.

Il passo 1, la fase di inizializzazione, dell'Algoritmo 2 è stato implementato utilizzando la funzione "*initialize*". Esso ha il compito di inizializzare le principali variabili utilizzate durante l'esecuzione del TSkmeans. In particolare, ha il compito di generare la matrice dei centroidi iniziali Z, nel caso in cui non sia stata passata in input, si occupa di generare in modo casuale la matrice dei pesi W rispettando le condizioni imposte da 3.2, e infine, inizializza le variabili utilizzate dal risolutore per problemi di programmazione quadratica "*quadprog*" offerto da MATLAB [6], utilizzato per ricavare la matrice dei pesi W nel passo 5 di cui parleremo fra poco. Di seguito il codice utilizzato.

Listing 3.2: initialize.m

```
1  function [Z, W, H, Aeq, beq, lb, ub] = initialize(X, k, alpha, n, m)
2  % inizializzo le matrici principali
3  Z = zeros(k, m);
4  W = zeros(k, m);
5
6  % vengono generati k indici di serie temporali in X in modo random
7  seriesIndex = randperm(n, k);
8  for i = 1:k
9     % scelgo i centroidi iniziali attraverso gli indici delle serie.
10    Z(i,:) = X(seriesIndex(i),:);
11    % genero random i pesi iniziali
12    W(i,:) = randfixedsum(m, 1, 1, 0, 1)';
```

```
end

% ricavo la matrice simmetrica hessiana H per il tool quadprog
H = zeros(k*m, k*m);
for p = 1:k
   for j = 1:m-1
      index1 = j+m*(p-1);
      index2 = j+1+m*(p-1);
      H(index1,index1) = H(index1,index1) + 1;
      H(index2,index2) = H(index2,index2) + 1;
      H(index1,index2) = H(index1,index2) - 1;
      H(index2,index1) = H(index2,index1) - 1;
   end
end
H = H * alpha;

% primo vincolo da applicare al problema di programmazione
% quadratica (Sommatoria da j=1 a m di w_pj deve essere 1)
Aeq = zeros(k, k*m);
for i = 1:k
   Aeq(i, (i-1)*m+1:i*m) = 1;
end
beq = ones(k, 1);

% ricavo il secondo vincolo del problema di programmazione
% quadratica (lb = 0 <= w_pj <= 1 = ub)
lb = zeros(k*m, 1);
ub = ones(k*m, 1);
end
```

Nel passo 3, ricaviamo la nuova matrice di appartenenza U utilizzando la funzione "*solve_U*". Per ricavare la regola di aggiornamento della matrice possiamo fare riferimento alla funzione obiettivo 3.1, infatti per minimizzare tale funzione, possiamo utilizzare un comune metodo di ottimizzazione, ovvero, fissiamo due variabili e minimizziamo la funzione obiettivo 3.1 per ottenere il valore delle altre variabili. In [1] viene dimostrato che fissando le matrici W e Z la funzione obiettivo è minimizzata solo se

$$u_{ip} = \begin{cases} 1, & se\ D_{pj} \leqslant D_{p'j},\ p' \neq p,\ 1 \leqslant p' \leqslant k \\ 0, & altrimenti \end{cases} \tag{3.3}$$

dove

$$D_{pj} = \sum_{j=1}^{m} w_{pj}(x_{ij} - z_{pj})^2. \tag{3.4}$$

Le serie temporali vengono dunque assegnate al cluster dove la distanza pesata tra il centroide del cluster e la serie è minima. Basandosi sulla 3.3, la funzione "*solve_U*" è stata implementata nel seguente modo.

Listing 3.3: solve_U.m

```
function [U] = solve_U(X, Z, W, k, n)
% fissato Z e W trovo la matrice U utilizzando la formula (3)
U = zeros(n, k);
distance = zeros(1, k);
dist_to_Z = zeros(1, n);
for i = 1:n
   for p = 1:k
      distance(p) = sum(W(p, :).*((X(i, :)-Z(p, :)).^2));
   end
   [min_dist, min_dist_idx] = min(distance);
   U(i, min_dist_idx) = 1;
   dist_to_Z(i) = min_dist;
end

% controllo se un cluster e' vuoto, nel caso ci inserisco una s. t.
empty_clusters = find(~any(U));
if ~isempty(empty_clusters)
   for i = 1:length(empty_clusters)
      % viene scelta la s. t. piu' lontana dal suo centroide
      [~, farthest_ts] = max(dist_to_Z);
      original_cluster = find(U(farthest_ts, :));

      % aggiorno le principali variabili di conseguenza
      U(farthest_ts, empty_clusters(i)) = 1;
      U(farthest_ts, original_cluster) = 0; %#ok<FNDSB>
      Z(empty_clusters(i), :) = X(farthest_ts, :);
      dist_to_Z(farthest_ts) = 0;
   end
end
end
```

Nella funzione "*solve_U*" viene aggiunto anche un segmento di codice per gestire il problema dei cluster vuoti del K-means, già discusso nella

Sezione 1.5, utilizzando la tecnica del riassegnamento dell'oggetto più lontano. Nello specifico, nel caso in cui durante l'aggiornamento della matrice di appartenenza U si verificasse che un cluster rimanga vuoto, ovvero, nel caso in cui nessuna serie temporale vi sia stata assegnata, la nostra implementazione del TSkmeans ricercherà la serie temporale più lontana dal suo centroide e la riassegnerà al cluster vuoto, aggiornando di conseguenza la matrice U.

Per il passo 4, ricaviamo la nuova matrice dei centroidi Z utilizzando la funzione "*solve_Z*". In questo caso, in [1] viene dimostrato che fissando le matrici W e U la funzione obiettivo è minimizzata se

$$z_{pj} = \frac{\sum_{i=1}^{n} u_{ip} x_{ij}}{\sum_{i=1}^{n} u_{ip}}. \tag{3.5}$$

La funzione "*solve_Z*" è stata dunque implementata nel modo seguente.

Listing 3.4: solve_Z.m

```
function [Z] = solve_Z(X, U, k, m)
% fissato W0 e U trovo la matrice Z utilizzando la formula (3.5)
Z = zeros(k, m);
for p = 1:k
   for j = 1:m
      Z(p, j) = sum(U(:, p).*X(:, j)) / sum(U(:, p));
   end
end
end
```

Infine, nel passo 5, ricaviamo la nuova matrice dei pesi W utilizzando la funzione "*solve_W*". Fissando le matrici U e Z la funzione obiettivo può essere minimizzata utilizzando un risolutore per problemi di programmazione quadratica, ovvero problemi basati sull'ottimizzazione matematica di una funzione quadratica basata su diverse variabili soggette a vincoli lineari [12]. Nel nostro metodo "*solve_W*" utilizzeremo dunque il risolutore per problemi di programmazione quadratica chiamato "*quadprog*" messo a disposizione dal software MATLAB [6]. Considerando U e Z fissate, la nostra funzione obiettivo 3.1 può essere rappresentata in forma matriciale dall'espressione $f' * w + \frac{1}{2} * w' * H * w$, dove w è un vettore che rappresenta i pesi incogniti da calcolare (i pesi di ogni istante di tempo per ogni cluster), f è un vettore che rappresenta la parte lineare della funzione obiettivo 3.1, ovvero la prima parte (considerando sempre U e Z fissate e dunque W come unica incognita), H è una matrice Hessiana che rappresenta la parte quadratica della funzione, ovvero la seconda, mentre

l'operatore $*$ rappresenta il prodotto fra matrici. Il vettore f comprende dunque i coefficienti dei pesi incogniti w della parte lineare, H invece, è la matrice simmetrica della forma quadratica [14] associata alla seconda parte della funzione obiettivo. Il vettore f verrà ricalcolato ad ogni iterazione dell'algoritmo dalla funzione "*solve_W*", mentre la matrice H e i vincoli lineari 3.2 associati alla matrice dei pesi W, verranno calcolati una sola volta dalla funzione di inizializzazione "*initialize*", essendo costanti rispetto alle iterazioni dell'algoritmo. I precedenti parametri verranno infine usati come input per il risolutore "*quadprog*", mentre quest'ultimo restituirà come output la nuova matrice dei pesi W che minimizza la funzione obiettivo (con U e Z fissate). Di seguito l'implementazione di "*solve_W*".

Listing 3.5: solve_W.m

```
function [W] = solve_W(X, U, Z, k, n, m, H, Aeq, beq, lb, ub)
% ricavo il vettore f, rappresenta i coefficenti della
% parte lineare della funzione obiettivo
f = zeros(k*m, 1);
for p = 1:k
   for i = 1:n
      for j = 1:m
         f(j+m*(p-1)) = f(j+m*(p-1)) + U(i,p)*((X(i,j)-Z(p,j))^2);
      end
   end
end

% risolvo il problema di programmazione quadratica e ricostruisco
% correttamente la matrice W
options = optimset('Display', 'off');
W = quadprog(H, f, [], [], Aeq, beq, lb, ub, [], options);
W = vec2mat(W, m);
end
```

3.4 FUNIONALITÀ DEI PESI

L'algoritmo TSkmeans, come abbiamo visto, fa un ampio uso dei pesi associati agli istanti di tempo, impiegandoli per caratterizzare i sottospazi trovati durante il processo di clustering. Per via della relativa uniformità delle serie temporali, l'algoritmo cerca di estrarre dei sottospazi altrettanto uniformi, cercando dunque di assegnare dei pesi simili a istanti di

tempo adiacenti. I pesi possono aiutare ad identificare gli istanti di tempo che hanno un alto valore discriminante per migliorare le prestazioni e i risultati del clustering, inoltre i pesi prodotti dall'algoritmo possono aiutare ad identificare gli intervalli di tempo dove le serie temporali presentano pattern simili facilitando dunque la loro analisi. L'algoritmo TSkmeans cerca quindi di assegnare pesi più grandi a istanti di tempo adiacenti che presentano una minore dispersione all'interno del cluster, mentre assegnerà pesi più piccoli agli istanti di tempo che invece presentano una dispersione del cluster più alta.

Il Parametro α

L'uniformità fra pesi di istanti di tempo adiacenti viene regolata dal parametro α. Se in input viene passato un valore di α pari a 0, otterremo l'annullamento della seconda parte della funzione obiettivo 3.1. In questo caso, non potremo ottenere dei pesi omogenei per istanti di tempo adiacenti, inoltre all'istante di tempo che presenta la minima dispersione all'interno del cluster verrà associato un peso pari a 1 mentre a tutti gli altri istanti di tempo verrà associato un peso pari a 0. Questo approccio ci garantisce di ricavare il minimo valore che ottimizza la funzione obiettivo ma per il clustering di serie temporali non sarebbe un risultato molto utile per via dell'importanza cronologica delle serie. Se invece assegniamo ad α un valore tale che $\alpha < 0$, avremo che la seconda parte della funzione obiettivo 3.1 risulterà negativa, portando i pesi relativi ad istanti di tempo adiacenti ad oscillare in modo marcato, e questo ovviamente va contro l'idea di ottenere sottospazi omogenei di serie temporali. L'ultima possibilità che abbiamo è quella di utilizzare $\alpha > 0$. In questo caso avremo che il valore della seconda parte della funzione obiettivo 3.1 aumenterà con l'aumentare del valore di α e questo porterà, per le osservazioni fatte in precedenza, ad ottenere istanti di tempo adiacenti con pesi omogenei. In aggiunta, la prima parte della funzione obiettivo 3.1 permetterà di assegnare pesi più grandi agli istanti di tempo adiacenti aventi pesi piccoli. Scegliere quindi $\alpha > 0$ ci consentirà di ottenere un clustering di serie temporali ottimale rispetto agli obiettivi che vogliamo raggiungere.

3.5 COMPLESSITÀ

L'algoritmo TSkmeans è un metodo iterativo che basa la sua esecuzione su tre passaggi fondamentali: l'aggiornamento della matrice di apparte-

nenza U, l'aggiornamento della matrice dei centroidi Z e l'aggiornamento della matrice dei pesi W. Facendo riferimento all'equazione 3.3 possiamo notare che aggiornare la matrice di appartenenza U richiede un costo computazionale pari a $O(k * n * m)$, dove k rappresenta il numero di cluster, n il numero di serie temporali e m il numero di istanti di tempo. Riferendoci invece all'equazione 3.5 possiamo vedere che anche l'aggiornamento della matrice dei centroidi Z richiede un costo pari a $O(k * n * m)$. Viceversa, per aggiornare la matrice dei pesi W dovremo risolvere un problema di programmazione quadratica, in questo caso dunque, il costo computazionale per aggiornare i pesi, dipende dalla complessità in tempo del risolutore utilizzato. Pertanto, se assumiamo che il nostro risolutore abbia una complessità di tempo pari a $f(j)$, dove j rappresenta la dimensione del suo input, possiamo dunque affermare che la complessità totale dei primi due passi è pari a $O(I * k * n * m)$, dove I rappresenta il numero di iterazioni impiegate dall'algoritmo TSkmeans per raggiungere la convergenza, mentre la sua complessità totale risulterà pari a $O(I * (f(j) + k * n * m))$.

3.6 UN ESEMPIO PRATICO

Andiamo ora a presentare un esempio di applicazione dell'algoritmo TSkmeans per dare un'idea un pò più precisa di come l'algoritmo si comporta difronte all'analisi di un set basato su serie temporali.

Per fornire una descrizione chiara e semplice dell'algoritmo sceglieremo di mostrare come si comporta il TSkmeans quando viene eseguito su un set di serie temporali molto piccolo. In particolare, prenderemo in considerazione un set formato da 5 serie temporali definite su una sequenza di 20 istanti di tempo presi ad intervalli regolari. La Figura 8 mostra il set completo che andremo ad analizzare. Possiamo subito notare facilmente che in questo semplice set sono presenti 2 cluster di serie temporali, uno formato da due serie e l'altro da tre, dunque nell'analisi del set di dati andremo ad imporre al TSkmeans di cercare questi due cluster. Il TSkmeans inoltre, come spiegato nelle sezioni precedenti, necessità in input di un parametro α per bilanciare gli effetti tra la dispersione delle serie temporali all'interno dei cluster e l'uniformità dei pesi associati a istanti di tempo adiacenti. Essendo il set in esame molto piccolo e semplice sceglieremo di assegnare ad α il valore della dispersione globale (*global scatter*) del set di dati, definita nella Sezione 4.4.2 dall'equazione 4.12.

A questo punto non ci resta che trovare il modo di rappresentare il nostro set di dati per passarlo in seguito come input all'algoritmo

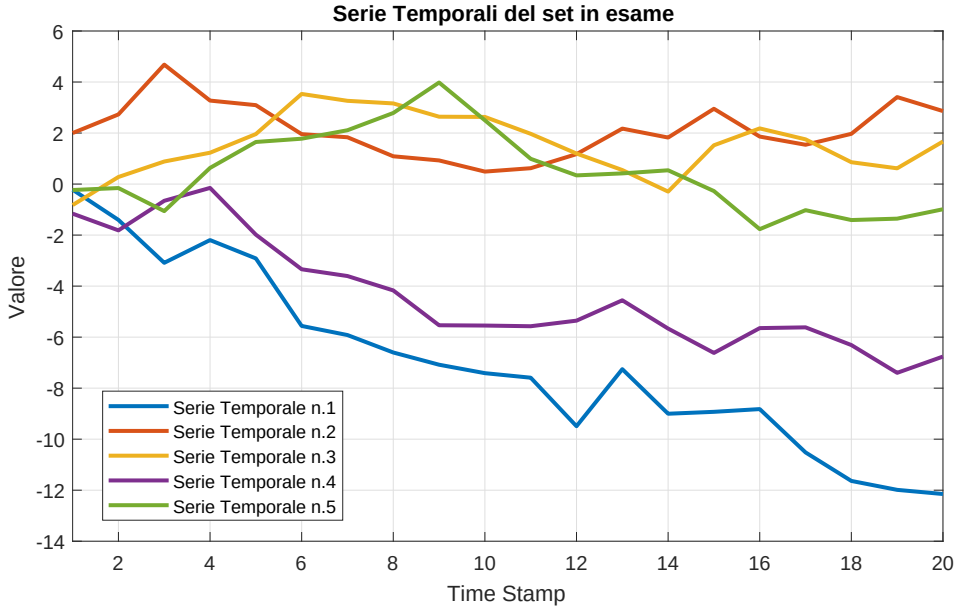

Figura 8: Set di serie temporali preso in esame.

TSkmeans. Il set delle serie temporali può essere rappresentato tramite una matrice con un numero di righe pari al numero di serie presenti nel set e un numero di colonne pari al numero totale di istanti di tempo che caratterizzano tali serie. Nel nostro caso dunque dovremo costruire una matrice di dimensioni 5×20 contenente tutti i valori che le varie serie temporali assumono in corrispondenza di ogni istante di tempo. Di seguito viene riportata la matrice così creata (solo alcuni valori) basandosi sul set mostrato in Figura 8 dove ogni riga fa riferimento ad una serie temporale diversa.

$$
\begin{aligned}
X &= \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,19} & x_{1,20} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,19} & x_{2,20} \\ x_{3,1} & x_{3,2} & \cdots & x_{3,19} & x_{3,20} \\ x_{4,1} & x_{4,2} & \cdots & x_{4,19} & x_{4,20} \\ x_{5,1} & x_{5,2} & \cdots & x_{5,19} & x_{5,20} \end{bmatrix} = \\
&= \begin{bmatrix} -0,2208 & -1,4051 & \cdots & -11,9867 & -12,1495 \\ 1,9997 & 2,7304 & \cdots & 3,4095 & 2,8615 \\ -0,8231 & 0,2774 & \cdots & 0,6173 & 1,6674 \\ -1,1598 & -1,8150 & \cdots & -7,3994 & -6,7616 \\ -0,2331 & -0,1563 & \cdots & -1,3557 & -0,9879 \end{bmatrix}
\end{aligned}
\tag{3.6}
$$

Passiamo ora a descrivere come si comporta l'algoritmo TSkmeans se fatto partire con i seguenti input: la matrice X descritta precedentemente basata sul set di esempio mostrato in Figura 8, il numero di cluster da ricercare, ovvero due ($k = 2$), e il valore per il parametro α, che abbiamo deciso di porre pari alla dispersione totale del set 4.12 ($\alpha = gs$).

Una volta avviato il TSkmeans entra come prima cosa nella sua fase di inizializzazione (il passo 1). Questa prima fase, come descritto nelle precedenti sezioni e come mostra la sua implementazione nel Listato 3.2, ha sostanzialmente tre compiti: sceglie in modo casuale i centroidi iniziali per ogni cluster, genera in modo casuale i pesi iniziali da associare ad ogni istante di tempo relativo ad ogni cluster e infine genera le variabili principali che verranno utilizzate in ogni iterazione del risolutore per problemi di programmazione quadratica "*quadprog*", utilizzato per aggiornare la matrice dei pesi W durante le iterazioni principali del TSkmeans. Nel nostro particolare caso, in questa prima fase, vengono dunque scelte in modo casuale due serie temporali che fungeranno da centroidi per i due cluster iniziali e che verranno utilizzate per inizializzare la matrice dei centroidi Z, utilizzata dal TSkmeans per salvarsi ad ogni iterazioni i centroidi di ogni cluster e di dimensioni 2×20 nel nostro caso. Assumendo che nella nostra esecuzione di esempio dell'algoritmo le serie scelte siano la prima e la terza del set fornito, la matrice Z sarà inizializzata nel seguente modo (i valori sono ripresi rispettivamente dalla prima e dalla terza serie temporale).

$$\begin{aligned} Z &= \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,19} & x_{1,20} \\ x_{3,1} & x_{3,2} & \cdots & x_{3,19} & x_{3,20} \end{bmatrix} = \\ &= \begin{bmatrix} -0,2208 & -1,4051 & \cdots & -11,9867 & -12,1495 \\ -0,8231 & 0,2774 & \cdots & 0,6173 & 1,6674 \end{bmatrix} \end{aligned} \tag{3.7}$$

Successivamente verrà inizializzata la matrice W dei pesi che servirà al nostro algoritmo per tracciare ogni singolo peso associato agli istanti di tempo di ogni cluster e dunque nel nostro caso necessiteremo di una matrice di dimensioni 2×20. I pesi in questa fase sono assegnati in

modo casuale rispettando i vincoli descritti in 3.2. In generale, nel nostro esempio la matrice W potrà assumere una forma simile alla seguente.

$$W = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,19} & w_{1,20} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,19} & w_{2,20} \end{bmatrix} = \\ = \begin{bmatrix} 0.0338 & 0.0232 & \cdots & 0.0405 & 0.1437 \\ 0.1357 & 0.0564 & \cdots & 0.0042 & 0.0346 \end{bmatrix} \tag{3.8}$$

Dopodichè il TSkmeans andrà a generare le variabili necessarie all'esecuzione del risolutore "*quadprog*" per risolvere il problema di programmazione quadratica che dovrà affrontare al passo 5. In particolare, come già spiegato in 3.3 durante questo passaggio il TSkmeans dovrà risolvere il problema di programmazione quadratica basato sulla ricerca di un vettore w (corrispondente ai pesi da ricercare) che rispetta i vincoli lineari imposti in 3.2 e minimizza la seguente funzione in forma matriciale : $f' * w + \frac{1}{2} * w' * H * w$. Per l'intera esecuzione del nostro algoritmo sia la matrice H sia le matrici e i vettori che rappresentano i vincoli lineari rimangono costanti e dunque l'algoritmo genererà queste variabili un sola volta durante questa fase di inizializzazione, mentre il vettore f verrà ricalcolato in ogni iterazione del TSkmeans durante il passo 5. Le variabili generate in questa fase di inizializzazione sono dunque le seguenti: la matrice Hessiana H che rappresenta la parte quadratica della funzione obiettivo del nostro problema, una matrice Aeq e un vettore beq utilizzato per imporre al risolutore "*quadprog*" di considerare il vincolo già visto in 3.2, ovvero

$$\text{Primo Vincolo} : \sum_{j=1}^{m} w_{pj} = 1 \qquad \forall p,\ 1 \leqslant p \leqslant k \tag{3.9}$$

e infine due vettori lb e ub per comunicare al risolutore "*quadprog*" di imporre il vincolo anch'esso già descritto in 3.2, ovvero

$$\text{Secondo Vincolo} :\ 0 \leqslant w_{pj} \leqslant 1 \qquad \forall w_{pj} \in W \tag{3.10}$$

Nello specifico, la matrice Hessiana H risulta una matrice simmetrica e sparsa, in generale inoltre la matrice ha un numero di righe e di colonne pari a $k * m$, ovvero il numero di cluster da ricercare moltiplicato per il numero totale di istanti di tempo che caratterizzano le serie temporali del set di dati in esame. Nel nostro caso avrà dunque una dimensione pari a 40×40. Per calcolarla possiamo basarci sulla tecnica proposta in [14]

e implementata nel Listato 3.2. La matrice Aeq ha un numero di righe pari al numero di cluster e un numero di colonne pari al prodotto fra il numero di cluster e il numero totale di istanti di tempo, nel nostro caso avrà dunque una dimensione pari a 2×40. Il vettore beq invece ha un dimensione pari al numero di cluster e nel nostro caso avrà dunque 2 elementi. La matrice Aeq e il vettore beq, utilizzati insieme, verranno interpretati dal risolutore "*quadprog*" per imporre il vincolo 3.9 attraverso la forma matriciale $Aeq * w = beq$, dove w rappresenta il vettore dei pesi che il risolutore "*quadprog*" dovrà calcolare, come spiegato in [6]. Nel nostro particolare esempio dunque Aeq e beq verranno generati nella seguente forma.

$$\begin{aligned} Aeq &= \begin{bmatrix} Aeq_{1,1} & \cdots & Aeq_{1,20} & Aeq_{1,21} & \cdots & Aeq_{1,40} \\ Aeq_{2,1} & \cdots & Aeq_{2,20} & Aeq_{2,21} & \cdots & Aeq_{2,40} \end{bmatrix} = \\ &= \begin{bmatrix} 1 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 1 & \cdots & 1 \end{bmatrix} \end{aligned} \tag{3.11}$$

$$beq = \begin{bmatrix} beq_1 \\ beq_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{3.12}$$

Infine, i vettori lb e ub avranno una dimensione pari a $k * m$ e nel nostro esempio avranno dunque un numero di elementi pari a 40. Utilizzati insieme verranno interpretati dal risolutore "*quadprog*" per imporre il vincolo 3.10 per ogni peso all'interno del vettore w e dunque fungeranno rispettivamente da limite inferiore e superiore per i valori dei pesi da calcolare per il vettore w. Nel nostro particolare esempio dunque lb e ub verranno generati nella seguente forma.

$$lb = \begin{bmatrix} lb_1 \\ \vdots \\ lb_{40} \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}, \qquad ub = \begin{bmatrix} ub_1 \\ \vdots \\ ub_{40} \end{bmatrix} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \tag{3.13}$$

Una volta che il TSkmeans ha terminato la fase di inizializzazione procederà ad entrare nel suo ciclo principale, come descritto nell'Algoritmo 2, continuando la sua esecuzione e aggiornando di volta in volta la matrice di appartenenza U, la matrice dei centroidi Z e quella dei pesi W fino a che non viene raggiunta la condizione di convergenza, che ricordiamo, verrà raggiunta quando gli assegnamenti ai cluster cessano

di cambiare, ovvero nel momento in cui la matrice di appartenenza U non cambia da un iterazione all'altra dell'algoritmo. Andremo ora dunque a descrivere un pò più nel dettaglio come si comporterà l'algoritmo nel nostro esempio di analisi durante il ciclo principale del TSkmeans.

Durante il passo 3 l'algoritmo TSkmeans si calcola la matrice di appartenenza U, inizialmente inizializzata a zero, che indicherà a quale cluster è stata assegnata ogni serie temporale del set. Per fare ciò il TSkmeans, usando l'equazione 3.3, va a calcolare per ogni serie temporale la sua distanza pesata 3.4 da ciascuno dei centroidi dei cluster. Successivamente ogni serie temporale viene assegnata al cluster rappresentato dal centroide più vicino ad essa. Supponendo dunque che la serie temporale i venga assegnata al cluster p la matrice U viene aggiornata inserendo il valore 1 nella cella u_{ip}. Questo procedimento viene così ripetuto fino a che ogni serie temporale del set è stata esaminata, a questo punto avremo ottenuto la nostra matrice U finale correttamente aggiornata. Inoltre, durante questo passo, se un cluster risulterà vuoto a quest'ultimo verrà assegnata la serie temporale più lontana dal uso cluster, modificando opportunamente la matrice U. L'implementazione di questo passo viene mostrata nel Listato 3.3. Nel nostro esempio, considerando le matrici iniziali definite precedentemente, la matrice U calcolata alla prima iterazione del TSkmeans avrà una dimensione pari a 5×2 e assumerà la seguente forma.

$$U = \begin{bmatrix} u_{1,1} & u_{1,2} \\ u_{2,1} & u_{2,2} \\ u_{3,1} & u_{3,2} \\ u_{4,1} & u_{4,2} \\ u_{5,1} & u_{5,2} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{3.14}$$

Successivamente il TSkmeans eseguirà il passo 4 dove procederà ad aggiornare la matrice dei centroidi Z usando l'equazione 3.5. In pratica, per ottenere l'elemento w_{pj} della nuova matrice Z il TSkmeans va a calcolarsi la media dei valori che le serie temporali appartenenti al p-esimo cluster assumono nell'istante di tempo j-esimo. Questo procedimento viene quindi ripetuto per ogni cluster fino a che ogni centroide viene ricalcolato, dopodichè avremo ottenuto la nostra matrice Z finale correttamente aggiornata. L'implementazione di questo passo viene mostrata nel Listato 3.4. Nel nostro esempio, considerando le matrici iniziali definite precedentemente, la matrice Z aggiornata alla prima iterazione del TSk-

means avrà una dimensione pari a 2×20 e assumerà la seguente forma.

$$\begin{aligned} Z &= \begin{bmatrix} z_{1,1} & z_{1,2} & \cdots & z_{1,19} & z_{1,20} \\ z_{2,1} & z_{2,2} & \cdots & z_{2,19} & z_{2,20} \end{bmatrix} = \\ &= \begin{bmatrix} -0.6903 & -1.6100 & \cdots & -9.6930 & -9.4555 \\ 0.3145 & 0.9505 & \cdots & 0.8904 & 1.1804 \end{bmatrix} \end{aligned} \tag{3.15}$$

Infine il TSkmeans eseguirà il passo 5 dove procederà ad aggiornare la matrice dei pesi W usando il risolutore per problemi di programmazione quadratica "*quadprog*". Tale risolutore per risolvere il nostro problema che si basa sulla minimizzazione della funzione $f' * w + \frac{1}{2} * w' * H * w$ e dunque essere in grado di calcolarsi i nuovi pesi da associare alla matrice W rispettando i vincoli lineari in 3.2 necessita di sei input, ovvero: il vettore f che comprende i coefficienti dei pesi incogniti w della funzione descritta poco prima e inoltre le matrici H e Aeq e i vettore beq, lb e ub che abbiamo descritto precedentemente. L'unico input che deve essere ricalcolato in questo passo risulta il vettore f, esso infatti rappresentando i coefficienti dei pesi incogniti w nella parte lineare della funzione obiettivo 3.1, ottenuta fissando le matrici U e Z, dovrà essere aggiornato ad ogni iterazione dato che anche le matrici U e Z cambieranno ad ogni iterazione dell'algoritmo. Il vettore f in generale avrà un numero di elementi pari a $k * m$. Una volta ricavato il vettore f il TSkmeans procede dunque a calcolare la nuova matrice aggiornata dei pesi W utilizzando il risolutore "*quadprog*". Il Listato 3.5 mostra l'implementazione in MATLAB di questo passo, nello specifico viene mostrato anche una possibile procedura per il calcolo del vettore f. Nel nostro esempio, considerando le matrici iniziali definite precedentemente, il vettore f calcolato alla prima iterazione del TSkmeans avrà un numero di elementi pari a 40 mentre la matrice W aggiornata alla prima iterazione del TSkmeans avrà una dimensione pari a 2×20 e assumerà una forma simile alla seguente.

$$\begin{aligned} W &= \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,19} & w_{1,20} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,19} & w_{2,20} \end{bmatrix} = \\ &= \begin{bmatrix} 0.1080 & 0.1062 & \cdots & 0 & 0 \\ 0.0503 & 0.0501 & \cdots & 0.0087 & 0.0068 \end{bmatrix} \end{aligned} \tag{3.16}$$

Arrivati a questo punto il TSkmeans avrà calcolato correttamente le nuove matrici U, Z e W e dunque procederà a controllare se è stato

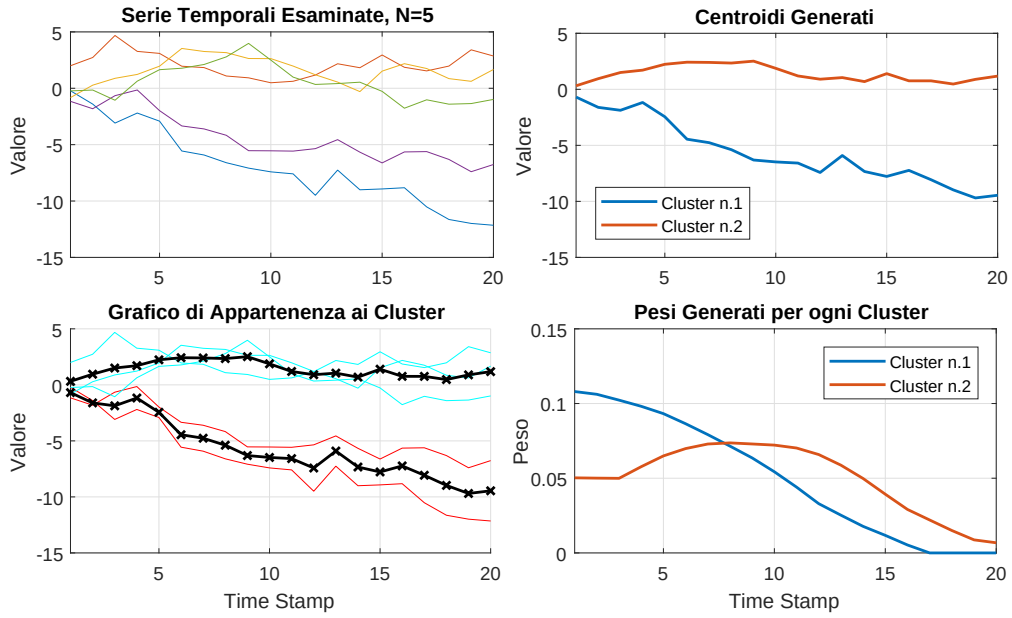

Figura 9: Risultati del clustering del TSkmeans sul set di esempio.

raggiunto il criterio di convergenza, confrontando la matrice U con la sua versione calcolata nell'iterazione precedente (U0). Se le due matrici sono uguali, il TSkmeans ha raggiunto la convergenza e dunque l'algoritmo termina uscendo dal suo ciclo principale assegnando il valore "*false*" alla variabile "*convergence*". In questo caso l'algoritmo restituisce come output le ultime versioni aggiornate delle matrici U, Z e *W*. Altrimenti, se U e U0 sono diverse, la matrice U viene salvata in U0 e il TSkmeans continua il suo ciclo riaggiornando tutte le matrici con i procedimenti spiegati precedentemente. Nel caso specifico del nostro set di esempio una volta completata la prima iterazione e dunque gli aggiornamenti delle matrici, il TSkmeans procederà con una altra iterazione, dato che alla fine della prima le matrici U e U0 saranno per forza diverse essendo U0 inizializzata a zero. Nella successiva iterazione però nessuna delle matrici principali cambierà il suo contenuto e dunque il TSkmeans raggiungerà la convergenza in sole due iterazioni e successivamente terminerà la sua esecuzione.

Concludiamo questo capitolo mostrando i risultati dall'algoritmo TSkmeans ottenuti con il set di esempio. Nella Figura 9 vengono mostrati i risultati finali del processo di clustering per il set di esempio, in particolare tale figura mostra le serie temporali presenti nel set originale, i

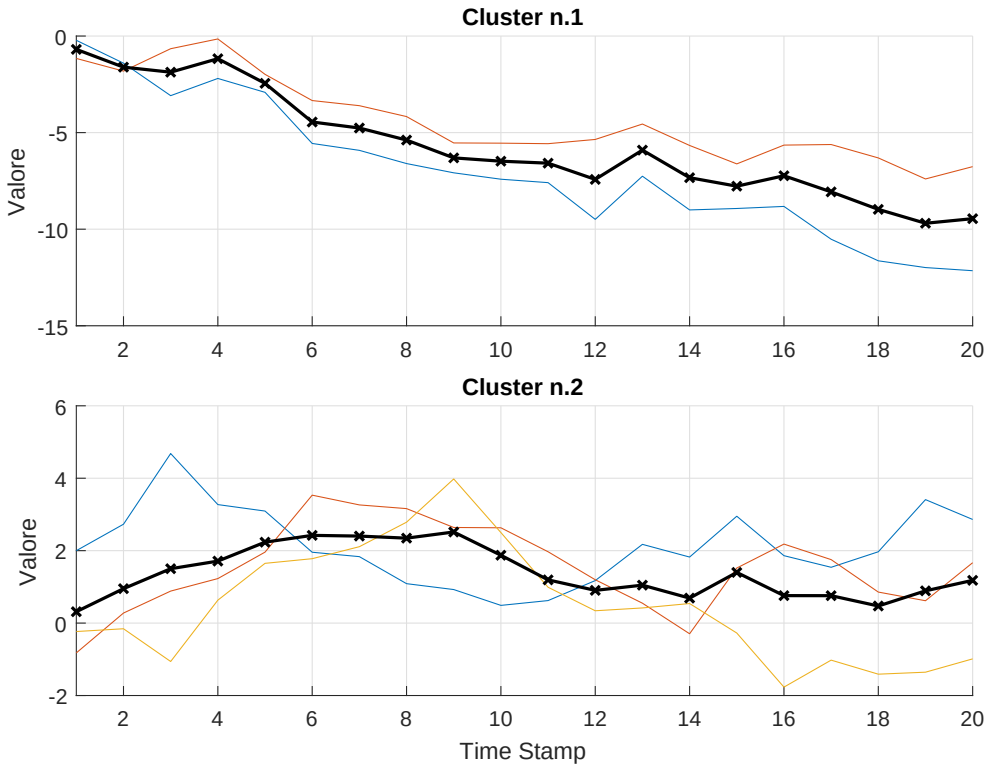

Figura 10: Cluster finali e centroidi restituiti dal TSkmeans per il set di esempio.

centroidi finali generati dall'algoritmo, un grafico che mostra a quali cluster sono state assegnate le varie serie e infine un grafico dei pesi associati ai cluster. La Figura 10 invece mostra in modo più approfondito quali cluster sono stati trovati e quali serie vi sono state inserite con inoltre i centroidi che caratterizzano tali cluster. Dando una rapida occhiata alle due figure possiamo notare facilmente come il TSkmeans sia stato in grado di trovare correttamente i due cluster nascosti nel set di esempio e le relative serie appartenenti ad essi, in più anche i pesi associati agli istanti di tempo dei cluster rispettano gli obiettivi che ci eravamo prefissati nella descrizione dell'algoritmo TSkmeans. Data la natura semplice del set di esempio un analisi approfondita dei pesi risulterebbe poco indicativa, rimandiamo dunque tale analisi al capitolo successivo, dove andremo ad effettuare uno studio più approfondito sulla qualità dei pesi restituiti dal TSkmeans su set di dati molto più rilevanti.

4

VERIFICHE SPERIMENTALI

In questo capitolo, andremo ad effettuare alcuni test sull'algoritmo TSkmeans presentato nel Capitolo 3, verificando il suo comportamento su diversi set di dati basati su serie temporali e misurando le sue prestazioni rispetto ad alcune fra le metriche più utilizzate nella valutazione delle tecniche di clustering. In particolare, inizialmente, parleremo degli approcci più utilizzati nella valutazione del clustering e vedremo in cosa consistono gli indici interni ed esterni per l'analisi dei cluster, presentando le metriche che verranno utilizzate per l'analisi del TSkmeans. Successivamente testeremo l'algoritmo TSkmeans sia su un set di dati sintetico, creato ad hoc, per mostrare le potenzialità dell'algoritmo nel caso in cui riesca a sfruttare i suoi punti di forza, sia su set di dati reali ricavati da cinque problemi legati ad applicazioni pratiche. L'algoritmo TSkmeans verrà inoltre confrontato con altre tecniche di clustering per serie temporali messe a disposizione dal software MATLAB, valutando le loro prestazioni con diverse misure di prossimità e mostrando i risultati ottenuti con grafici e tabelle riassuntive per comprendere meglio le prestazioni dei metodi di clustering confrontati.

4.1 VALUTAZIONE DEL CLUSTERING

La valutazione del clustering (*cluster validation*) rappresenta la parte più difficile e frustante della *cluster analysis* [13]. Eseguire una valutazione complessiva di una tecnica di clustering richiederebbe una misura indipendente e affidabile per la comparazione dei risultati ottenuti dal processo di clustering, tuttavia, la vasta tipologia di dati utilizzabili, le diverse rappresentazioni che possono assumere, l'ampio numero di algoritmi di clustering esistenti e le difficoltà nel definire il concetto stesso di cluster, rendono la ricerca di queste misure molto difficile. Difatti, non esiste uno metodo unico per la valutazione del clustering, ma una grande varietà di misure e indici, presi spesso da altre aree di studio, come ad

esempio la statistica teorica o l'*information retrieval*. Ciò nonostante, la valutazione del clustering è un processo molto importante che deve essere presente in ogni studio di algoritmi di clustering. Infatti, in genere ogni algoritmo basato su queste tecniche troverà dei cluster in un set di dati, anche se tale set non presenta nessun tipo di struttura o pattern naturale. La valutazione del clustering risulta dunque essenziale per capire se nel set di dati che stiamo esaminando sia presente una reale struttura nascosta non casuale.

Valutare un processo di clustering però, non consiste solo nel cercare di distinguere se il set di dati in esame ha delle strutture reali oppure casuali, ma esistono diversi importanti aspetti da considerare nella valutazione del clustering. Alcuni di questi aspetti sono riportati di seguito.

- Determinare la tendenza del cluster (*cluster tendency*) di un set di dati, ovvero, scoprire se esistono davvero delle reali strutture non casuali nei dati.
- Determinare il numero corretto di cluster presenti.
- Valutare quanto sono adeguati i risultati del clustering rispetto ai dati senza fare riferimento ad informazioni esterne.
- Confrontare i risultati di una cluster analysis con delle informazioni esterne conosciute, come ad esempio delle etichette di classe (*class labels*), fornite esternamente, degli oggetti su cui si esegue il clustering.
- Confrontare due set di cluster per determinare qual è il migliore.

Le misure di valutazione, chiamate anche indici, che vengono applicate per giudicare vari aspetti della valutazione del clustering, sono generalmente classificate in due tipologie [3].

Misure Non Supervisionate

Le misure non supervisionate comprendono gli indici che valutano la qualità del clustering senza riferirsi ad informazioni esterne. Inoltre, le misure non supervisionate sono spesso suddivise ulteriormente in due classi: misure di coesione del cluster (*cluster cohesion*), che determinano quanto gli oggetti contenuti nei cluster sono correlati fra loro, e le misure di separazione del cluster (*cluster separation*), che determinano quanto un cluster sia distinto o separato da altri cluster. Infine, le misure non supervisionate sono spesso chiamate indici interni (*internal indices*) per il

fatto che utilizzano solo informazioni presenti nel set di dati. Un esempio di indice interno, appartenente alle misure di coesione, è l'SSE 1.6.

Misure Supervisionate

Le misure supervisionate comprendono invece gli indici che valutano quanto corrisponde la struttura o un pattern trovato da un algoritmo di clustering rispetto ad una struttura fornita esternamente. Le misure supervisionate sono spesso chiamate indici esterni (*external indices*) per il fatto che utilizzano delle informazioni che non sono presenti nel set di dati. Un esempio di indice esterno è rappresentato dalla misura *purity*, che descriveremo nelle sezioni successive.

Nella prossime sezioni parleremo in modo specifico degli indici interni ed esterni, discutendo dei casi in cui vengono applicati e descrivendo le specifiche metriche che andremo ad utilizzare nella valutazione dell'algoritmo TSkmeans e di altre tecniche di clustering per dati basati su serie temporali.

4.2 INDICI INTERNI

La maggior parte degli indici interni per la valutazione del clustering si basano sui concetti di coesione (*cohesion*) e separazione (*separation*) e più in generale sul concetto di validità. Possiamo esprimere la validità (*validity*) di un set di k cluster come la somma pesata delle validità dei singoli cluster, ovvero,

$$\text{Validità Totale} = \sum_{i=1}^{K} w_i * \text{validità}(C_i) \tag{4.1}$$

dove C_i rappresenta il cluster i, mentre w_i il peso associato al cluster i per il calcolo della validità totale. La funzione "*validità*" può essere sostituita da una funzione di coesione, separazione o una combinazione di esse. I pesi inoltre, possono essere definiti in modo diverso a seconda della misura di valutazione usata, ad esempio alcune volte possono avere un valore pari ad 1 o pari alla dimensione del cluster, mentre in altri casi rispecchiano una proprietà più sofisticata. Se la funzione di validità corrisponde alla coesione, allora risultano migliori i valori più alti, se invece corrisponde alla separazione, i risultati migliori sono dati dai valori più bassi.

Per cluster basati su prototipi, come quelli che andremo a valutare nelle sezioni successive, la coesione di un cluster può essere definita come la somma delle prossimità rispetto al prototipo (centroide o medoide) del cluster. La separazione tra due cluster invece, può essere misurata dalla prossimità dei due loro prototipi [3]. La coesione di un cluster è dunque definita da

$$\mathrm{Coesione}(C_i) = \sum_{x \in C_i} \mathrm{prossimità}(x, c_i) \tag{4.2}$$

dove c_i rappresenta il prototipo del cluster C_i. La separazione di un cluster invece, come dimostrato in [3], è correlata alla separazione tra i prototipi dei cluster e un prototipo complessivo $\mathcal{C}$ calcolato rispetto ad ogni oggetto del set di dati, ovvero,

$$\mathit{Separazione}(C_i) = \mathrm{prossimità}(c_i, \mathcal{C}). \tag{4.3}$$

Per la misura di prossimità possiamo scegliere, ad esempio, una di quelle definite nella Sezione 1.4.2, possiamo dunque osservare che per la coesione del cluster, se scegliamo come prossimità la distanza euclidea quadratica 1.2, otteniamo il suo SSE 1.6. Quando la prossimità è misurata dalla distanza euclidea standard 1.1, la separazione viene normalmente calcolata dall'SSB (*sum of squares between clusters*), che corrisponde alla somma delle distanza al quadrato tra il centroide di un cluster c_i e il prototipo complessivo $\mathcal{C}$. L'SSB totale sarà invece ottenuto sommando gli SSB dei singoli cluster. Più l'SSB è alto e più i cluster risulteranno separati fra di loro. Di seguito la sua definizione, dove m_i rappresenta la dimenione dell'i-esimo cluster.

$$\mathit{SSB\ Totale} = \sum_{i=1}^{K} m_i * \mathrm{dist}(c_i, \mathcal{C})^2. \tag{4.4}$$

Per ottenere una indice interno globale per la valutazione del clustering, possiamo quindi utilizzare una combinazione delle precedenti definizioni di coesione e separazione per calcolarci la validità totale del cluster 4.1 come somma pesata di queste misure. Esiste inoltre una forte relazione fra la coesione e la separazione, infatti in [3] viene dimostrato che la somma fra l'SSE 1.6 e l'SSB 4.4 è costante e coincide con la somma totale dei quadrati TSS (*total sum of squares*), che rappresenta la somma delle distanze quadratiche tra ogni oggetto del set di dati e il prototipo complessivo $\mathcal{C}$. Minimizzare l'SSE (coesione) comporta la massimizzazione dell'SSB (separazione) e viceversa [3], dunque una buona qualità dei cluster si otterrà quando avremo un basso TSS.

Vogliamo mostrare infine un ultimo indice interno per la valutazione del clustering, ovvero il coefficiente di *silhouette*.

Il coefficiente di *silhouette* combina le misure di coesione e separazione. Di seguito spiegheremo i tre passaggi necessari per calcolare il coefficiente di *silhouette* per l'i-esimo oggetto di un set di dati. Per prima cosa, calcoliamo la sua distanza media da tutti gli altri oggetti appartenenti allo stesso cluster e chiamiamo questo valore a_i. Successivamente, per l'i-esimo oggetto e qualsiasi cluster che non lo contiene verrà calcolata la distanza media di esso da tutti gli altri oggetti contenuti in questi cluster, e di queste distanze prenderemo quella minima, chiamando il suo valore b_i. Infine, calcoleremo il coefficiente di *silhouette* per l'i-esimo oggetto attraverso la seguente formula.

$$\text{Silhouette}_i = \frac{b_i - a_i}{\max(a_i, b_i)} \tag{4.5}$$

Il valore del coefficiente di *silhouette* può variare da -1 a 1. Un valore negativo non è desiderabile perchè avremo che la distanza media dagli oggetti del proprio cluster risulti maggiore della minima distanza media dagli oggetti in un differente cluster. Vogliamo dunque che il coefficiente di *silhouette* sia positivo, ovvero $a_i < b_i$, e vogliamo che a_i sia il più vicino possibile a 0, dato che in questo caso otterremo il valore massimo 1 del coefficiente. Per calcolare il coefficiente di *silhouette* di un cluster basterà calcolare la media del coefficiente degli oggetti appartenenti a tale cluster, mentre se vogliamo una misura media globale basterà eseguire la media del coefficiente di *silhouette* di tutti gli oggetti del set di dati [3].

Gli indici interni sono quindi utilizzati per la valutazione del clustering quando come informazioni abbiamo a disposizione solo i cluster finali generati dall'algoritmo che vogliamo valutare, proprio per questo dunque, disponendo solo di informazioni legate strettamente alla tecnica utilizzata, possono essere utilizzati solo per comparare algoritmi di clustering che utilizzano gli stessi modelli o metriche e le stesse misure di prossimità.

4.3 INDICI ESTERNI

Gli indici esterni sono delle misure per la valutazione del clustering che sono tipicamente utilizzate quando abbiamo delle informazioni esterne riguardanti il set di dati da esaminare, in genere, delle etichette di classe associate ad ogni oggetto del set fornite esternamente da persone esperte che conoscono la struttura e la natura dei dati appartenenti a tale set.

Tali etichette, esprimono semplicemente la classe di appartenenza di ogni oggetto del set di dati. In questi casi, la procedura che si segue consiste nel misurare il grado di corrispondenza tra le etichette dei cluster finali ottenuti da un algoritmo di clustering e le etichette di classe fornite. Questi tipi di indici sono quindi molto utili per confrontare e valutare le prestazioni degli algoritmi di clustering che utilizzano anche differenti approcci o metriche. Di seguito andremo a presentare gli indici esterni utilizzati per la valutazione dell'algoritmo TSkmeans.

Gli indici che presenteremo seguono due approcci. Alcune tecniche usano misure che derivano dagli algoritmi di classificazione come ad esempio la metrica *Purity* o l'*F-Score* (*F-measure*). Queste ultime valutano in quale misura un cluster contiene oggetti di una singola classe. Altre tecniche invece, come ad esempio la metrica *Rand Index* (*Rand measure*), valutano in quale misura due oggetti che appartengono alla stessa classe si trovano nello stesso cluster e vice versa. Questi due tipi di approcci vengono chiamati, rispettivamente, orientati alla classificazione (*classification-oriented*) e orientati alla similarità (*similarity-oriented*). Inoltre, nelle misure orientate alla classificazione, in genere viene misurato il grado di corrispondenza tra delle etichette di classe pronosticate da un algoritmo di classificazione (predette prima del clustering) e quelle reali (fornite esternamente), ma per le misure che presenteremo, i risultati non cambieranno se verranno utilizzate le etichette dei cluster finali, ottenuti dalle tecniche di clustering, invece che pronostici di etichette di classe [3].

Tuttavia, non esiste un compromesso od una tecnica accettata universalmente per la valutazione del clustering con indici esterni, perché la differente natura dei dati e il concetto stesso e non ben definito di cluster non hanno consentito ai ricercatori di trovare una misura ottimale e universale da applicare in questi casi. Ogni misura dunque avrà i suoi punti deboli e i suoi punti forti ed a seconda del tipo di set di dati a cui viene applicata, potrebbe dare risultati non del tutto soddisfacenti [2].

Purity

L'indice esterno *Purity* rappresenta una semplice e trasparente misura di valutazione. Supponiamo che $C = \{C_1, C_2, \ldots, C_K\}$ sia il set dei cluster restituiti dall'algoritmo di clustering che vogliamo valutare e $C' = \{C'_1, C'_2, \ldots, C'_K\}$ sia il set delle classi dei dati, ovvero, il set ricavato dalle etichette di classe per il set di dati in esame, dove C_i rappresenta l'insieme degli oggetti appartenenti al cluster i, mentre C'_j rappresenta l'insieme degli oggetti appartenenti alla classe j. Per calcolare la *purity*,

ogni cluster è assegnato alla classe più frequente presente in esso e successivamente l'accuratezza di tale assegnamento viene misurata contando il numero di oggetti correttamente assegnati e dividendolo per il numero totale degli oggetti del set di dati, che chiameremo N. Formalmente dunque la *purity* viene calcolata nel modo seguente [15].

$$\text{Purity} = \frac{1}{N} \sum_{i=1}^{K} \max_{1 \leqslant j \leqslant K} |C_i \cap C'_j| \tag{4.6}$$

Un clustering di bassa qualità avrà una *purity* con un valoro vicino a 0, mentre un clustering di alta qualità avrà una valore di *purity* vicino a 1. Tuttavia, nel caso in cui il numero di cluster sia molto alto, è molto facile ottenere una valore dell'indice di *purity* ingannevolmente alto, in particolare, se ogni oggetto viene assegnato ad un cluster diverso avremo una purity pari a 1. Perciò non è consigliabile fare affidamento solo sull'indice di *purity* per la valutazione del clustering [2].

F-Score

L'indice esterno *F-score* combina le informazioni della metriche *Precision* e *Recall*, molto utilizzate nell'ambito dell'*Information Retrieval*, per valutare in che misura un cluster contiene solo oggetti di una particolare classe e tutti gli oggetti di una classe. Data una particolare classe C'_j di dimensione n_j e un determinato cluster C_i di dimensione n_i, supponendo che $n_{i,j}$ rappresenti il numero di oggetti nel cluster C_i che appartengono alla classe C'_j, allora l'*F-Score* di questa coppia classe-cluster è definito da

$$\text{F-Score}(C_i, C'_j) = \frac{2 * \text{Recall}(C_i, C'_j) * \text{Precision}(C_i, C'_j)}{\text{Recall}(C_i, C'_j) + \text{Precision}(C_i, C'_j)} \tag{4.7}$$

dove $\text{Recall}(C_i, C'_j)$ calcola la misura di *Recall* definita da $n_{i,j}/n_j$, ovvero, il numero di veri positivi diviso il numero di tutti i test che sarebbero dovuti risultare positivi (ovvero veri positivi più falsi negativi), mentre $\text{Precision}(C_i, C'_j)$ calcola la misura di *Precision* definita da $n_{i,j}/n_i$, ovvero, il numero di veri positivi diviso il numero di tutti i risultati positivi. L'*F-Score* per un classe generica C_r è il massimo valore dell'*F-Score* tra C_j e uno dei cluster finali restituiti, di seguito la formula.

$$\text{F-Score}(C'_j) = \max_{C_i \in C} \text{F-Score}(C_i, C'_j) \tag{4.8}$$

Mentre l'*F-Score* dell'intero clustering è definito come la somma degli *F-Score* di ogni classe, pesati a seconda delle dimensioni di queste ultime. Formalmente definito da

$$\text{F-Score} = \sum_{j=1}^{k} \frac{n_j}{N} \max_{1 \leqslant i \leqslant k} \frac{2 * (n_{i,j}/n_j) * (n_{i,j}/n_i)}{n_{i,j}/n_j + n_{i,j}/n_i} \tag{4.9}$$

dove k rappresenta il numero dei cluster e conseguentemente delle classi disponibili, N è il numero totale di oggetti del set di dati, mentre le altre variabili sono state prese dalle definizioni precedenti [16]. Anche l'*F-Score* è un indice che può variare da 0 a 1, dove 1 risulta il valore ottimale.

Rand Index

Un altro indice esterno molto utilizzato e orientato alla similarità è la misura *Rand Index*. Supponiamo di avere un set di cluster $C = \{C_1, C_2, \ldots, C_K\}$, un set di classi $C' = \{C'_1, C'_2, \ldots, C'_K\}$ e un set di dati $X = \{X_1, X_2, \ldots, X_N\}$ formato da N oggetti. Definiamo le seguenti variabili, utilizzando i set precedenti.

- a, il numero delle coppie di oggetti in X che appartengono alla stessa classe in C' e allo stesso cluster in C,
- b, il numero delle coppie di oggetti in X che appartengono alla stessa classe in C' ed a due cluster diversi in C,
- c, il numero delle coppie di oggetti in X che appartengono a due classi diverse in C' e allo stesso cluster in C,
- d, il numero delle coppie di oggetti in X che appartengono a due classi diverse in C' e a due cluster diversi in C.

Allora $a + b + c + d = M$, dove $M = N(N-1)/2$ è il numero totale di coppie possibili di oggetti del set X e N è il numero totale degli oggetti [17]. Il *Rand Index* viene dunque calcolato da

$$\text{Rand Index} = \frac{a + d}{M}. \tag{4.10}$$

Il *Rand Index* ha un valore che va da 0 a 1, con 0 che esprime la totale discordanza tra C e C' mentre 1 la loro completa uguaglianza.

Normalized Mutual Information

Il *Normalized Mutual Information* (NMI) è un indice esterno molto utilizzato per la valutazione del clustering per serie temporali che, come i precedenti indici, assume un valore che varia da 0 a 1, dove 1 rappresenta la perfetta correlazione tra un set di cluster e un set di classi. Supponendo di avere un set di cluster $C = \{C_1, C_2, \ldots, C_K\}$ e un set di classi $C' = \{C'_1, C'_2, \ldots, C'_K\}$, l'*NMI* viene definito formalmente da

$$\mathrm{NMI} = \frac{\sum_{i=1}^{K}\sum_{j=1}^{K} n_{i,j}\log_2(\frac{n*n_{i,j}}{n_i*n_j})}{\sqrt{(\sum_{i=1}^{K} n_i \log_2 \frac{n_i}{n})(\sum_{j=1}^{K}\log_2\frac{n_j}{n})}}, \tag{4.11}$$

dove, n rappresenta il numero totale di oggetti del set di dati, n_p è il numero di oggetti nel cluster C_i, n_q è il numero di oggetti nella classe C'_j, mentre $n_{i,j}$ è il numero degli oggetti che appartengono sia al cluster C_i sia alla classe C'_j [18].

4.4 TEST SU SET DI DATI SINTETICO

Avendo introdotto gli indici per la valutazione del clustering, passeremo ora ad analizzare ed a valutare le prestazioni dell'algoritmo TSkmeans rispetto agli indici presentati e confrontandolo con alcuni degli algoritmi di clustering partizionali più usati in letteratura. In questa sezione, valuteremo l'algoritmo quando ha a che fare con un set di serie temporali pensato per sfruttarne i suoi punti di forza, ovvero un set di serie che presentano una forte correlazione in alcune loro sotto sequenze. A questo scopo creeremo un set di serie temporali sintetico, creato dunque ad hoc con caratteristiche precise, su cui valutare le prestazioni dell'algoritmo. Inoltre, l'algoritmo TSkmeans, come abbiamo mostrato nel Capitolo 1, necessita in input un valore α per decidere in che misura bilanciare la dispersione degli oggetti all'interno dei cluster e l'uniformità dei pesi associati a istanti di tempo adiacenti, per questo dunque mostreremo una procedura per ottenere il valore ottimale per il parametro α da utilizzare come input per le successive valutazioni.

Essendo il TSkmeans un algoritmo partizionale basato sul K-means, confronteremo le sue prestazioni con i seguenti algoritmi: K-means basato sulla distanza Euclidea quadratica 1.2, K-means basato sul coefficiente di Pearson 1.4, K-means basato sulla distanza di Manhattan 1.3, K-means basato sulla similarità del coseno, K-means basato sulla distanza DTW 2.2, K-medoids basato sul DTW e K-medoids basato sulla distanza Eucli-

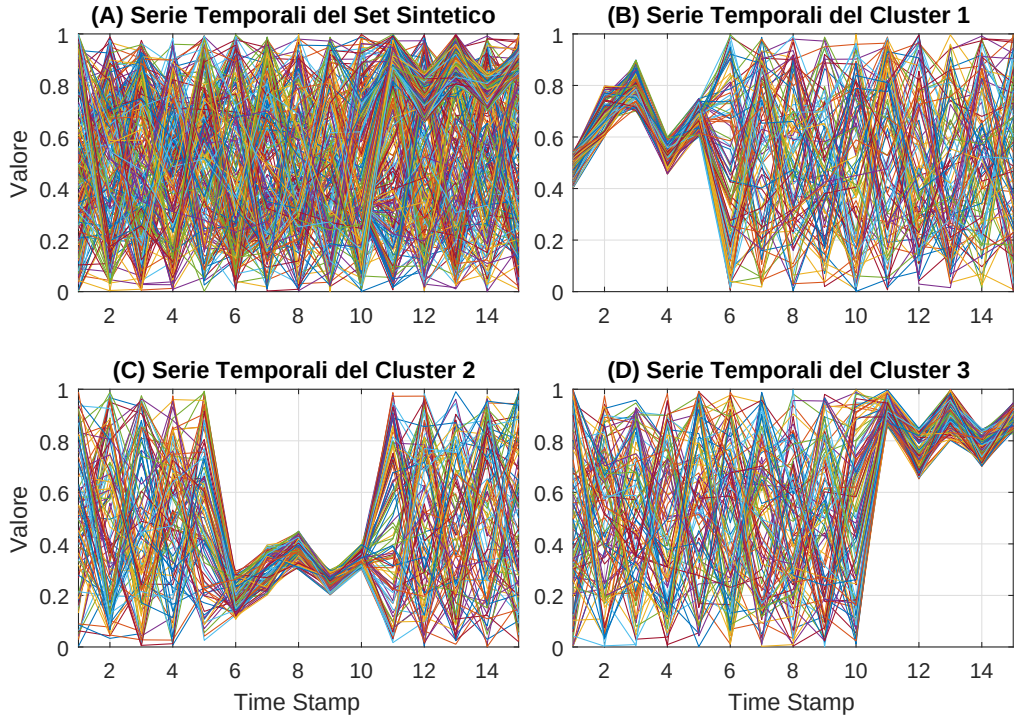

Figura 11: Serie temporali nel set di dati sintetico.

dea quadratica. L'algoritmo K-means basato sulla distanza DTW è stato implementato da zero riutilizzando solo la misura per la distanza DTW fornita dal software MATLAB, per tutti gli altri algoritmi invece, abbiamo utilizzate le versioni messe a disposizione da tale software [19, 20].

4.4.1 *Generazione del Set di Dati Sintetico*

Al fine di verificare che TSkmeans sia in grado di estrarre dei sottospazi (sotto sequenze) omogenei e corretti dal processo di clustering di serie temporali, progetteremo un set di dati sintetico che presenti in ogni suo cluster una sotto sequenza dove le serie temporali del set presentano una forte correlazione. Di seguito descriveremo le caratteristiche del set.

Il set di dati sintetico proposto è formato in totale da 300 serie temporali basate su 15 istanti di tempo. Il set è caratterizzato da 3 cluster, ognuno composto da 100 serie temporali. Il set sintetico completo viene mostrato nella Figura 11 (A), mentre la Figura 11 (B) mostra il Cluster 1, la Figura 11 (C) il Cluster 2 e la Figura 11 (D) il Cluster 3, dove ogni curva mostra una serie temporale diversa. In ogni cluster è stata generata una sequenza

di 5 istanti di tempo consecutivi in cui le serie temporali presentano pattern similari, ovvero dei valori simili rispetto agli istanti di tempo, mentre per tutti gli altri istanti si è scelto un valore casuale. Nel Cluster 1, la sequenza che presenta pattern simili va dall'istante di tempo 1 all'istante 5 compresi, nel Cluster 2, i pattern simili vanno dall'istante 6 all'istante 10 compresi mentre il Cluster 3 la sequenza con pattern simili va dall'istante 11 all'istante 15 compresi. Oltre a generare il set sintetico viene generato anche un vettore di etichette di classe $C = \{c_1, c_2, \ldots, c_N\}$, dove N è il numero totale di serie temporali del set sintetico ($N = 300$) e c_i è la classe, o in questo caso il cluster reale, a cui appartiene l'i-esima serie temporale. Il vettore di etichette di classe C verrà poi utilizzato dagli indici esterni durante i test per valutare la qualità dei cluster ottenuti dai vari algoritmi di clustering sul set sintetico. Di seguito viene riportato lo script MATLAB utilizzato per la creazione del set sintetico di serie temporali, per la generazione dei suoi cluster e del vettore delle etichette di classe.

```
% inizializzo i cluster e il data set
C1 = zeros(100, 15);
C2 = zeros(100, 15);
C3 = zeros(100, 15);
X = zeros(300, 15);
membership = zeros(300, 1);

% genero le serie temporali per ogni cluster
for i = 1:100
   C1(i, 1) = (0.55-0.4).*rand + 0.4;
   C1(i, 2) = (0.8-0.6).*rand + 0.6;
   C1(i, 3) = (0.9-0.7).*rand + 0.7;
   C1(i, 4) = (0.6-0.45).*rand + 0.45;
   C1(i, 5) = (0.75-0.6).*rand + 0.6;
   C1(i, 6:15) = rand(1, 10);
   C2(i, 1:5) = rand(1, 5);
   C2(i, 6) = (0.3-0.1).*rand + 0.1;
   C2(i, 7) = (0.4-0.2).*rand + 0.2;
   C2(i, 8) = (0.45-0.3).*rand + 0.3;
   C2(i, 9) = (0.3-0.2).*rand + 0.2;
   C2(i, 10) = (0.4-0.3).*rand + 0.3;
   C2(i, 11:15) = rand(1, 5);
   C3(i, 1:10) = rand(1, 10);
   C3(i, 11) = (1-0.85).*rand + 0.85;
   C3(i, 12) = (0.85-0.65).*rand + 0.65;
```

```
    C3(i, 13) = (1-0.80).*rand + 0.80;
    C3(i, 14) = (0.85-0.7).*rand + 0.7;
    C3(i, 15) = (0.95-0.825).*rand + 0.825;
end

% creo il data set ordinato per cluster
X(1:100, 1:15) = C1;
X(101:200, 1:15) = C2;
X(201:300, 1:15) = C3;

% vettore di etichette di classe
membership(1:100) = 1;
membership(101:200) = 2;
membership(201:300) = 3;

% riordino casualmente le righe e ottengo il set sintetico finale X
X = [membership X];
X = X(randperm(end), :);
save('synthetic_data_set', 'X', '-ascii');
```

4.4.2 *Ricerca del Valore Ottimale di α*

Il parametro α in input all'algoritmo TSkmeans ha l'importante funzione di controllare l'uniformità dei pesi per gli istanti di tempo, risulta dunque essenziale studiare l'impatto del parametro α rispetto ai risultati del processo di clustering per poter sceglierne il valore che consenta al TSkmeans di raggiungere le prestazioni migliori. Osservando la funzione obiettivo del TSkmeans 3.1 possiamo notare come uno stesso valore di α possa fornire diversi livelli di uniformità ai pesi associati agli istanti di tempo a causa delle differenti dimensioni dei set di dati a [1]. Dunque, per cercare di minimizzare l'impatto dell'estensione del set di dati sull'uniformità dei pesi possiamo assegnare un diverso valore ad α a seconda della dispersione globale del set di dati in esame, in questo modo il valore di α verrà calcolato tenendo conto anche delle dimensioni del set. La dispersione globale (*global scatter*) di un set di dati viene stimata nel seguente modo:

$$gs = \sum_{i=1}^{n} \sum_{j=1}^{m} (x_{ij} - z_{oj})^2 \tag{4.12}$$

dove $z_{oj} = (\sum_{i=1}^{n} x_{ij})/n$. Dovendo scegliere per i nostri obbiettivi, come spiegato nella Sezione 3.4, un $\alpha > 0$ seguiremo la scelta fatta nell'articolo originale del TSkmeans [1], mostrando come variano le valutazioni dei quattro indici esterni, discussi nella Sezione 4.3, al variare dei valori di $\ln(\alpha/gs)$ da -2.9957 ($\alpha/gs = 0.05$) a 6.9078 ($\alpha/gs = 10^3$). I valori di α/gs verranno presi usando passi sempre più grandi, in particolare prenderemo i seguenti valori: 0.05, con un passo pari a 0.2 i valori da 0.2 a 1, con un passo pari a 2 i valori da 2 a 10, con un passo pari a 50 i valori da 50 a 500, e 10^3, per un totale di 22 valori. Una volta ricavati i valori di α da testare abbiamo dunque valutato le prestazioni dell'algoritmo TSkmeans calcolandoci la media dei risultati degli indici esterni su 100 iterazioni di esso rispetto ad ogni valore di α in esame (22 valori).

Nella Figura 12 viene mostrato il cambiamento delle prestazioni del TSkmeans al variare del parametro α per il set sintetico, ovvero, le tendenze degli indici esterni al variare del parametro α. Osservando i dati sperimentali riportati in Figura 12, possiamo notare come l'algoritmo TSkmeans raggiunge delle ottime prestazioni quando $\ln(\alpha/gs) < 1$, ovvero quando il parametro $\mathtt{alpha}$ varia tra $0 < \alpha < e * gs$, dove gs rappresenta la dispersione globale del set sintetico mentre e rappresenta il numero di Eulero. Per misurare le prestazioni dell'algoritmo TSkmeans sul set di dati sintetico proposto abbiamo deciso di valutare l'algoritmo su $\ln(\alpha/gs) = 0$, ovvero $\alpha/gs = 1$, e dunque utilizzando un valore di α pari alla dispersione globale del set sintetico, in modo tale da ottenere dei pesi abbastanza uniformi senza intaccare troppo sulle prestazioni dell'algoritmo.

4.4.3 *Analisi delle Prestazioni e dei Risultati su Set Sintetico*

Mostreremo adesso come si comporta l'algoritmo TSkmeans se applicato al set di dati sintetico proposto precedentemente, confrontandolo con gli algoritmi di clustering partizionale più usati per l'analisi di serie temporali, citati precedentemente. Gli algoritmi basati sul K-means o sul K-medoids tendono a produrre una soluzione locale ottimale a seconda della posizione dei prototipi iniziali scelti. Per comparare le prestazioni del TSkmeans con gli altri algoritmi di clustering andremo dunque ad eseguire un test basato su 100 iterazioni, dove in ogni iterazione eseguiremo ciascun algoritmo e ne valuteremo le prestazioni. All'inizio di ogni iterazione del test, per cercare di rendere i risultati più comparabili possibili, genereremo un set di prototipi iniziali da applicare in input a ciascuna esecuzione dei vari algoritmi. Per effettuare la valutazione di

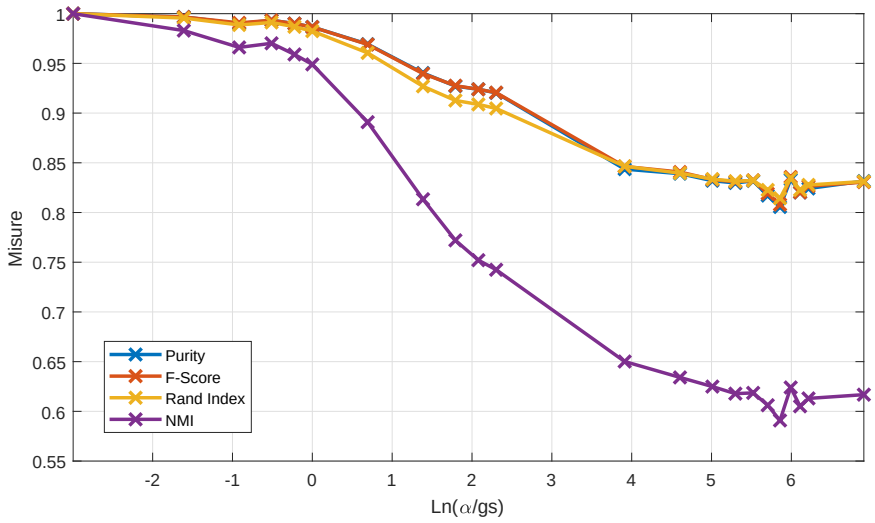

Figura 12: Prestazioni del TSkmeans sul set sintetico al variare del parametro α.

tutti gli algoritmi abbiamo dunque calcolato la media degli indici interni ed esterni e dei tempi di esecuzione ottenuti dopo aver eseguito tali algoritmi per ogni iterazione del test. In particolare, per il TSkmeans abbiamo scelto un valore di α pari alla dispersione globale del set sintetico, per i motivi spiegati nelle Sezioni 4.4.2 e 3.4.

I risultati ottenuti per gli indici esterni sono mostrati nella Tabella 2. Come possiamo notare l'algoritmo TSkmeans riesce ad ottenere delle prestazioni significativamente migliori, superiori allo 0.95% per ogni metrica, rispetto a tutti gli altri algoritmi e questo conferma la capacità del TSkmeans nel trovare cluster di qualità per set di serie temporali che presentano una forte correlazione in alcune loro sotto sequenze. In particolare, osserviamo che l'algoritmo K-means basato sulla distanza DTW è quello che si avvicina di più alle prestazione del TSkmeans, questo per via della capacità della distanza DTW di trovare corrispondenze ottime tra serie temporali. L'algoritmo che invece ottiene i risultati meno soddisfacenti è il K-medoids.

Per quanto riguarda gli indici interni, non potendo direttamente confrontare le misure di coesione e di separazioni dei diversi algoritmi, per via delle loro diverse misure di prossimità, abbiamo scelto di valutare la qualità dei loro cluster finali confrontando i valori medi ottenuti per le metriche basate sulla distanza Euclidea quadratica 1.2 ovvero, l'SSE 1.6, l'SSB 4.4 e il TSS in aggiunta al coefficiente di *silhouette* 4.5 sempre basato su tale distanza. Con queste condizione ci aspettiamo che l'al-

Algoritmo	Purity	F-Score	Rand Index	NMI
Time Series K-Means	**0,986333333**	**0,986326628**	**0,982112821**	**0,948174643**
Euclide K-Means	0,791033333	0,793730143	0,805091193	0,575133902
Pearson K-Means	0,6607	0,641179437	0,698556745	0,365802562
Manhattan K-Means	0,7026	0,704396466	0,749279822	0,497389988
Coseno K-Means	0,692133333	0,669130401	0,717218952	0,404929828
DTW K-Means	0,876166667	0,87455308	0,850332219	0,655931307
DTW K-Medoids	0,5956	0,587754689	0,654052174	0,26367622
Euclide K-Medoids	0,566	0,584421519	0,665084504	0,333768924

Tabella 2: Indici esterni medi ottenuti sul set sintetico

Algoritmo	SSE	SSB	TSS	Silhouette
Time Series K-Means	248,1616183	70,05222872	318,213847	0,193038274
Euclide K-Means	**245,8126212**	**72,40122581**	**318,213847**	**0,211873875**
Pearson K-Means	1545,630515	1334,612523	2880,243038	0,171888983
Manhattan K-Means	262,6800132	114,0652551	376,7452683	0,189018879
Coseno K-Means	703,3291256	433,3445472	1136,673673	0,178642311
DTW K-Means	247,995268	70,21857902	318,213847	0,196352386
DTW K-Medoids	392,2371303	184,3479062	576,5850366	0,103708787
Euclide K-Medoids	311,7709106	155,125631	466,8965415	0,159175225

Tabella 3: Indici interni medi ottenuti sul set sintetico

goritmo K-means basato sulla distanza Euclidea ottenga le prestazioni migliori. I risultati ottenuti, sono mostrati dalla Tabella 3. Effettivamente, l'algoritmo K-means basato sulla distanza Euclidea riesce ad ottenere le prestazioni migliori in particolare per quanto riguarda l'SSE e coefficiente di *silhouette*, superando di poco anche il TSkmeans e confermando le nostre supposizioni. Il TSkmeans però riesce comunque ad ottenere una valutazione leggermente migliore per l'SSB e dunque ottiene per l'indice TSS la medesima valutazione dell'algoritmo K-means basato sulla distanza Euclidea.

Analizzando invece i risultati ottenuti nel calcolo dei tempi medi di esecuzione, mostrati nella Tabella 4, possiamo notare come l'algoritmo K-means basato sulla distanza Euclidea risulti il più rapido, ma il TSkmeans, nonostante la sua relativa complessità, riesce comunque ad ottenere ottime prestazioni in termini di tempi di esecuzione. Gli algoritmi più lenti risultano invece quelli sulla distanza DTW, essendo

quest'ultima, come spiegato nella Sezione 2.4.2, molto costosa dal punto di vista computazionale.

Algoritmo	Tempi di Esecuzione (secondi)
Time Series K-Means	0,018647306
Euclide K-Means	**0,002060774**
Pearson K-Means	0,00217509
Manhattan K-Means	0,002352556
Coseno K-Means	0,002056027
DTW K-Means	0,264020586
DTW K-Medoids	0,730440912
Euclide K-Medoids	0,006601104

Tabella 4: Tempi di esecuzione medi degli algoritmi sul set sintetico

Infine, vogliamo mostrare come sono stati costruiti dall'algoritmo TSkmeans i cluster finali per il set sintetico e inoltre vogliamo verificare che i pesi finali, calcolati durante il processo di clustering e associati agli istanti di tempo dei cluster finali, rispecchino la reale importanza delle sotto sequenze nel clustering delle serie temporali presenti nel set. Nella Figura 13 viene mostrato il risultato del processo di clustering dell'algoritmo TSkmeans. La Figura 13 (A) mostra il set sintetico completo formato da tutte le sue serie temporali, la Figura 13 (B) mostra i centroidi finali generati dal TSkmeans, la Figura 13 (C) mostra in un unico grafico tutti i cluster generati dall'algoritmo, dove ad ogni serie è assegnato uno specifico colore a seconda del cluster a cui appartiene, mentre la Figura 13 (D) mostra i pesi degli istanti di tempo per ogni cluster.

Confrontando la Figure 14 e 11 notiamo subito che il cluster 1 trovato dal TSkmeans combacia con il cluster reale 2 del set sintetico, il cluster 2 combacia con il cluster reale 3 e il cluster 3 combacia con il cluster reale 1 del set sintetico. Dunque, a parte per una differenza di ordine delle etichette, i cluster trovati dal TSkmeans combaciano quasi perfettamente con i reali cluster del set sintetico definito nella Sezione 4.4.1. L'unica differenza è data da alcune serie temporali che sono state erroneamente assegnate al cluster 3 della Figura 14. Questo dimostra che il TSkmeans è in grado di scovare i reali cluster o pattern all'interno di un set di serie temporali.

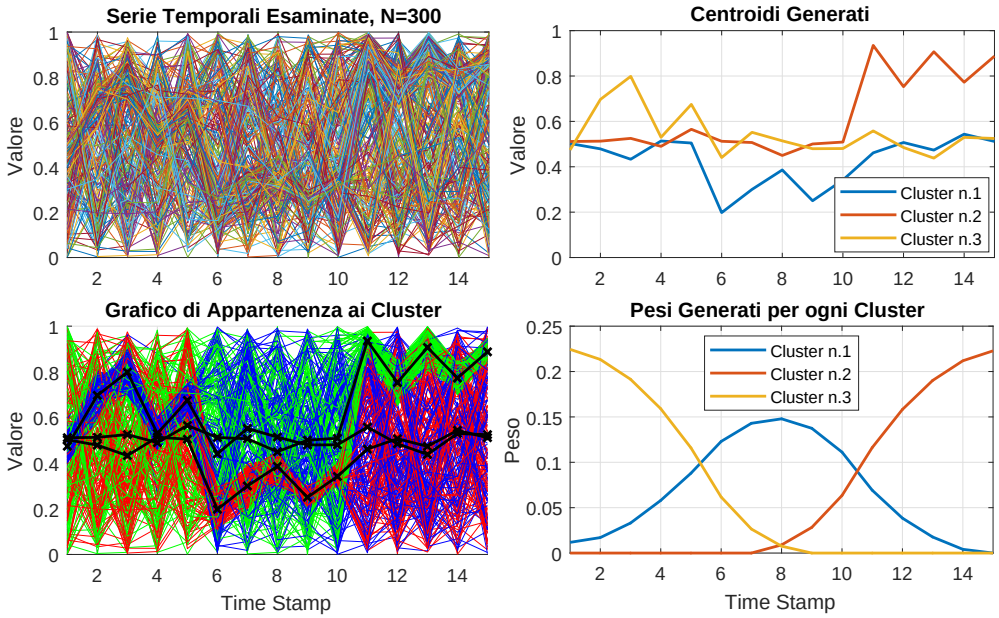

Figura 13: Risultati del clustering del TSkmeans sul set sintetico.

Riguardo ai pesi, se confrontiamo le Figure 13 (D) e 11 possiamo notare che l'algoritmo TSkmeans riesce correttamente ad individuare quali sono le sotto sequenze di ogni cluster che hanno un forte valore discriminativo per il processo di clustering. Infatti, se osserviamo la Figura 13 (D) notiamo che il TSkmeans ha assegnato per il cluster 1 un forte valore discriminativo, e di conseguenza un peso maggiore, agli istanti di tempo compresi tra 6 e 10, e questo combacia perfettamente con il fatto che gli istanti di tempo più rilevanti del cluster reale 2, come spiegato nella Sezione 4.4.1, siano proprio quelli appartenenti a tale intervallo, dove le serie temporali presentano una forte correlazione. Lo stesso ragionamento vale per i restanti cluster finali 2 e 3 trovati dall'algoritmo TSkmeans. Di conseguenza, tutte le serie temporali che presentano forti similarità tra gli istanti di tempo 6 e 10 verranno assegnate al cluster 1 generato dal TSkmeans, come dimostrato dalla Figura 13, e lo stesso discorso vale per i restanti cluster finali.

Le considerazioni fatte in questa sezione provano dunque che l'algoritmo TSkmeans è in grado di rilevare sottospazi (sotto sequenze) omogenei in set di dati sintetici basati su serie temporali.

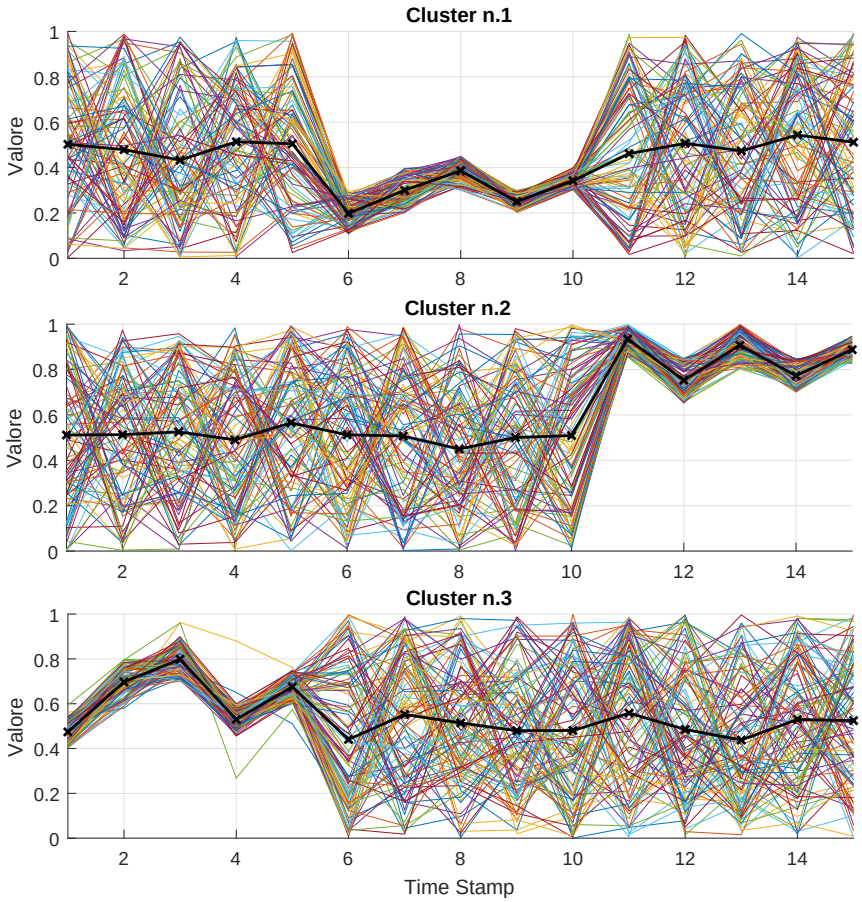

Figura 14: Cluster finali, con i relativi centroidi, restituiti dal TSkmeans per il set sintetico.

4.5 TEST SU SET DI DATI REALI

In questa sezione, valuteremo l'algoritmo TSkmeans e le altre tecniche di clustering utilizzando cinque set di dati reali legati a problemi pratici che sono stati presentati e discussi in [21, 22]. Come nel caso del set sintetico, faremo prima uno studio per cercare il valore ottimale del parametro α per ogni set di dati, mentre successivamente valuteremo e mostreremo i risultati ottenuti testando su di essi i vari algoritmi di clustering.

Il primo set di dati su cui valuteremo gli algoritmi di clustering sarà il set "FaceFour". Questo set, descritto in [23], contiene un insieme di serie temporali che rappresentano i profili di diverse persone, ottenute in modo simile all'esempio fatto nella Sezione 2.2.1. Il set è stato costruito prendendo molteplici immagini di quattro profili appartenenti a persone diverse, utilizzando anche espressioni differenti del volto. I profili so-

Set di Dati	Serie Temporali	Istanti di Tempo	Cluster
FaceFour	350	112	4
ECGFiveDays	884	136	2
ItalianPowerDemand	1096	24	2
SynteticControl	600	60	6
ArrowHead	211	175	3

Tabella 5: Proprietà dei set di dati reali

no stati poi convertiti in serie temporali utilizzando un elaboratore di immagini. I cluster nascosti in questo set sono dunque quattro.

Il secondo set che utilizzeremo è chiamato "ECGFiveDays" e consiste in un set di serie temporali che rappresentano l'elettrocardiogramma di un uomo di 67 anni, registrato in due date differenti. I cluster nascosti in questo caso saranno dunque due.

Il terzo set è chiamato "ItalianPowerDemand" e consiste in un set di serie temporali che rappresentano l'andamento della domanda di energia elettrica dell'Italia nell'anno 1997. Questo set è stato creato per distinguere l'andamento nei mesi tra Ottobre e Marzo rispetto ai mesi fra Aprile e Settembre. I cluster da trovare sono quindi pari a due.

Il quarto set è chiamato "SynteticControl" e consiste in un set di serie temporali che rappresentano l'andamento delle carte di controllo, ovvero strumenti utilizzati nell'ambito della statistica per mantenere sotto osservazione i vari parametri di un processo, generate in modo sintetico e basate sul processo descritto in [22]. I cluster da trovare sono pari a sei.

Infine il quinto e ultimo set è chiamato "ArrowHead" e consiste in un set di serie temporali che rappresentano la forma di un gruppo di punte di freccia, ricavate da immagini come nel caso del primo set, e utilizzate per la classificazione di diversi tipi di frecce per uno studio antropologico. In questo caso, i cluster nascosti sono tre.

Nella Tabella 5 sono riportati le proprietà principali di questi set di dati, utili per comprendere la natura dei set e le valutazioni che mostreremo in seguito. Passeremo ora a cercare i valori ottimali del parametro α per ottenere le migliori prestazioni per l'algoritmo TSkmeans.

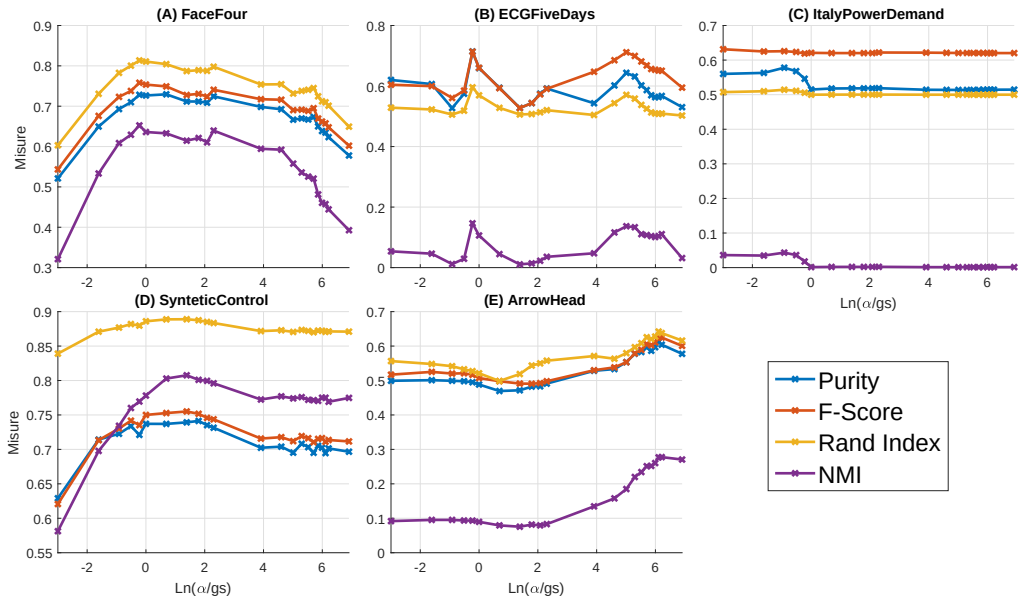

Figura 15: Prestazioni del TSkmeans sui set di dati reali al variare del parametro α.

4.5.1 *Ricerca del Valore Ottimale di α*

Per la ricerca del valore ottimale di α, come per il caso del set sintetico, sceglieremo i valori di α valutando come variano le medie degli indici esterni su 100 iterazioni dell'algoritmo TSkmeans rispetto all'intervallo di valori per $\ln(\alpha/gs)$ da -2.9957 ($\alpha/gs = 0.05$) a 6.9078 ($\alpha/gs = 10^3$). In particolare, per α/gs considereremo lo stesso intervallo di valori visto nella Sezione 4.4.2 per il caso del set sintetico (22 valori totali).

Nella Figura 15 viene mostrato il cambiamento delle prestazioni del TSkmeans al variare del parametro α per ogni set reale descritto precedentemente. In generale, possiamo osservare che all'aumentare del valore di α le prestazioni del TSkmeans su tali set aumentano di conseguenza, fino ad arrivare ad un punto in cui l'aumento ulteriore del valore di α fa degradare le prestazioni sulla maggior parte dei set. Per il set "FaceFour" (Figura 15 (A)), le prestazioni massime vengono raggiunte quando scegliamo un valore vicino a $\ln(\alpha/gs) = 0$ e dunque per questo set sceglieremo un valore di α pari a gs, la dispersione totale 4.12 del set. Per il set "ECGFiveDays" (Figura 15 (B)), le prestazioni massime vengono raggiunte, anche in questo caso, per valori di $\ln(\alpha/gs)$ vicini a 0 e dunque per questo set sceglieremo un $\alpha = gs$. Per il set "ItalianPowerDemand" invece (Figura 15 (C)), le prestazioni massime vengono raggiunte per

Algoritmo	Purity	F-Score	Rand Index	NMI
Time Series K-Means	**0,711428571**	**0,740902869**	**0,800263835**	**0,62046674**
Euclide K-Means	0,619732143	0,643951615	0,73789897	0,434269448
Pearson K-Means	0,619285714	0,645743903	0,737889318	0,440086309
Manhattan K-Means	0,659375	0,683934856	0,761015122	0,494492705
Coseno K-Means	0,619285714	0,645743903	0,737889318	0,440086309
DTW K-Means	0,574196429	0,617547639	0,660217181	0,402113367
DTW K-Medoids	0,696428571	0,748823005	0,787323037	0,630183363
Euclide K-Medoids	0,623392857	0,650082077	0,732833012	0,424862644

Tabella 6: Indici esterni medi ottenuti sul set "FaceFour"

valori di $\ln(\alpha/gs)$ vicini a -1 e dunque per questo set sceglieremo un $\alpha = gs/e$. Per il set "SynteticControl" (Figura 15 (D)), otteniamo le massime prestazioni per valori di $\ln(\alpha/gs)$ vicini a 1 e dunque per questo set sceglieremo un $\alpha = gs * e$. Infine, per il set "ArrowHead" (Figura 15 (E)), abbiamo ottenuto le massime prestazioni per valori di $\ln(\alpha/gs)$ vicini a 6 e dunque per questo set sceglieremo un $\alpha = gs * e^6$.

4.5.2 *Analisi delle Prestazioni e dei Risultati su Set Reali*

Mostreremo adesso come si comporta l'algoritmo TSkmeans con i set di dati reali presentati precedentemente, confrontandolo con gli algoritmi di clustering partizionale già citati. Come nel caso del set sintetico, per confrontare le prestazioni del TSkmeans con gli altri algoritmi di clustering abbiamo eseguito un test basato su 100 iterazioni, dove in ogni iterazione ogni algoritmo è stato eseguito e valutato, sul set di dati reale esaminato, attraverso una media degli indici esterni ed interni per la valutazione del clustering e rispetto ai tempi di esecuzione ottenuti. Inoltre, come spiegato nella Sezione 4.4.3 nel caso del set sintetico, ad ogni iterazione del test abbiamo generato casualmente un nuovo set di prototipi da passare in input ai vari algoritmi di clustering per ottenere dei risultati più rilevanti e comparabili. Di seguito analizzeremo i risultati ottenuti nei test riferendoci solo alle valutazioni ottenute con gli indici esterni e ai tempi di esecuzione degli algoritmi, essendo quest'ultimi i dati più rilevanti ed interessanti.

I risultati ottenuti dai vari algoritmi rispetto agli indici esterni sono mostrati nelle Tabelle 6, 7, 8, 9 e 10.

Come possiamo notare dalle tabelle, l'algoritmo TSkmeans riesce ad

Algoritmo	Purity	F-Score	Rand Index	NMI
Time Series K-Means	**0,651414027**	**0,650444788**	**0,564724151**	**0,099148309**
Euclide K-Means	0,516606335	0,516587819	0,500001691	0,000818773
Pearson K-Means	0,515395928	0,515382646	0,499914037	0,000692331
Manhattan K-Means	0,505045249	0,505015172	0,499499905	9,53892E-05
Coseno K-Means	0,515395928	0,515382646	0,499914037	0,000692331
DTW K-Means	0,57459276	0,587454768	0,512643292	0,024396051
DTW K-Medoids	0,624174208	0,618802245	0,532767381	0,060236087
Euclide K-Medoids	0,516968326	0,516967708	0,500010249	0,000830939

Tabella 7: Indici esterni medi ottenuti sul set "ECGFiveDays"

Algoritmo	Purity	F-Score	Rand Index	NMI
Time Series K-Means	0,57129562	0,624861489	0,51239556	0,040740059
Euclide K-Means	0,51459854	0,620223369	0,499970003	0,001339382
Pearson K-Means	0,51459854	0,622317201	0,499970003	0,001401493
Manhattan K-Means	**0,678439781**	**0,72152331**	**0,637524798**	**0,248742946**
Coseno K-Means	0,51459854	0,622317201	0,499970003	0,001401493
DTW K-Means	0,510072993	0,593938404	0,499746592	0,0004417
DTW K-Medoids	0,510948905	0,603176849	0,499783355	0,000579065
Euclide K-Medoids	0,51459854	0,622317201	0,499970003	0,001401493

Tabella 8: Indici esterni medi ottenuti sul set "ItalianPowerDemand"

Algoritmo	Purity	F-Score	Rand Index	NMI
Time Series K-Means	0,745733333	0,761457953	0,892002393	0,811788441
Euclide K-Means	0,69685	0,710479019	0,870913634	0,774455804
Pearson K-Means	0,592183333	0,581727431	0,817860824	0,603801536
Manhattan K-Means	0,631616667	0,64583197	0,845612577	0,656017973
Coseno K-Means	0,592183333	0,581727431	0,817860824	0,603801536
DTW K-Means	0,726683333	0,751426388	0,874652031	0,728802501
DTW K-Medoids	**0,956666667**	**0,956607807**	**0,972365053**	**0,909724987**
Euclide K-Medoids	0,5517	0,546711348	0,806646689	0,58065094

Tabella 9: Indici esterni medi ottenuti sul set "SynteticControl"

Algoritmo	Purity	F-Score	Rand Index	NMI
Time Series K-Means	**0,607535545**	**0,620895227**	**0,638603024**	0,271419402
Euclide K-Means	0,565924171	0,587779848	0,607749041	0,258257701
Pearson K-Means	0,565308057	0,585057386	0,605330174	0,262193164
Manhattan K-Means	0,57943128	0,607766194	0,630229294	**0,276596756**
Coseno K-Means	0,565308057	0,585057386	0,605330174	0,262193164
DTW K-Means	0,589810427	0,599034165	0,63150079	0,24094861
DTW K-Medoids	0,588056872	0,588809678	0,627227262	0,260555797
Euclide K-Medoids	0,570236967	0,59685999	0,614284812	0,271528738

Tabella 10: Indici esterni medi ottenuti sul set "ArrowHead"

ottenere le migliori prestazioni rispetto a vari indici esterni nella maggioranza dei casi e questo conferma le potenzialità del TSkmeans nel trovare cluster di alta qualità per set di serie temporali. In particolare, vediamo che le prestazioni migliori vengono ottenute con i set di dati "FaceFour", "ECGFiveDays", "ArrowHead" e "SynteticControl", riuscendo ad ottenere degli indici esterni quasi sempre con un valore superiore allo 0.6%. Nei set di dati "FaceFour", "ECGFiveDays" e "ArrowHead" ottiene le valutazioni massime rispetto ad ogni altro algoritmo di clustering. Per quanto riguarda invece il set "SynteticControl", l'algoritmo che ottiene le migliori prestazioni risulta il K-medoids basato sulla distanza DTW, che si distingue particolarmente dagli altri algoritmi con dei valori sempre sopra lo 0.9%, ma il TSkmeans riesce ad ottenere le prestazioni migliori rispetto a tutti gli algoritmi basati sul K-means e ottiene comunque dei risultati che lo collocano al secondo posto rispetto a tutti gli altri algoritmi. L'algoritmo che invece riesce abbastanza sorprendentemente, dato i risultati in altri set di dati, ad ottenere le prestazioni migliori per il set "ItalianPowerDemand" è il K-means basato sulla distanza Manhattan. Questo suo buon risultato è dovuto probabilmente alla forte resistenza della distanza Manhattan ai dati anomali e ai rumori, che nel set "ItalianPowerDemand" sono presenti in larga quantità.

Per quanto riguarda invece i tempi di esecuzione, mostrati nella Tabella 11 l'algoritmo che ottiene le migliori prestazioni in ogni set di dati è il K-means basato sulla distanza Euclidea.

Anche in questo caso il risultato non deve sorprendere date le ottime prestazioni rispetto al tempo dell'algoritmo K-means, come spiegato nella Sezione 1.4, e per la semplicità della distanza Euclidea, che nella sua forma quadratica 1.2 consente un ulteriore rapididà di esecuzione per

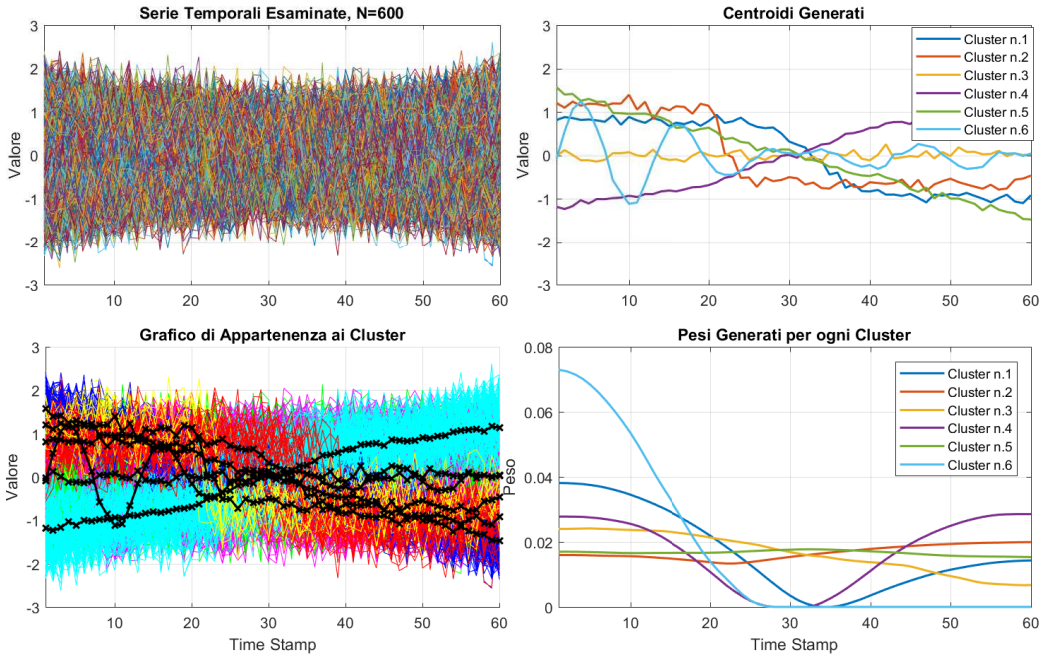

Figura 16: Risultati del clustering del TSkmeans sul "SynteticControl".

l'assenza del calcolo della radice. Il TSkmeans, considerando la complessità della sua funzione obiettivo 3.1, ottiene comunque delle buone prestazioni in termini di tempi di esecuzione e nella maggioranza dei casi risulta molto più rapido degli algoritmi che sfruttano la distanza DTW 2.2, nota per la sua pesantezza. Nel caso del set "ECGFiveDays" infatti, la misura DTW, rallenta l'algoritmo K-medoids per un tempo medio di addirittura 25 secondi.

Infine, vogliamo mostrare come sono stati costruiti, dall'algoritmo TSkmeans, i cluster finali per il set "SynteticControl", per dimostrare che, anche per set di dati reali ricavati da problemi pratici, il TSkmeans sia

Tempi di Esecuzione (secondi)

Algoritmo	FaceFour	ECGFiveDays	ItalianPowerDemand	SynteticControl	ArrowHead
Time Series K-Means	3,657559638	0,371639306	0,05533087	0,321638485	0,576603267
Euclide K-Means	**0,002325891**	**0,005394802**	**0,00251172**	**0,002999766**	**0,003515345**
Pearson K-Means	0,002754315	0,006040031	0,002926271	0,003325851	0,003809942
Manhattan K-Means	0,006108559	0,014857353	0,004154094	0,007399809	0,00807227
Coseno K-Means	0,002631143	0,005952164	0,002884968	0,003275857	0,003706915
DTW K-Means	0,529018369	1,741742405	0,63453599	1,809497805	0,612554328
DTW K-Medoids	2,079258219	25,23698814	10,34541187	5,673538444	3,415672039
Euclide K-Medoids	0,00564299	0,053022563	0,056147643	0,047775712	0,007509423

Tabella 11: Tempi di esecuzione medi degli algoritmi sui set di dati reali

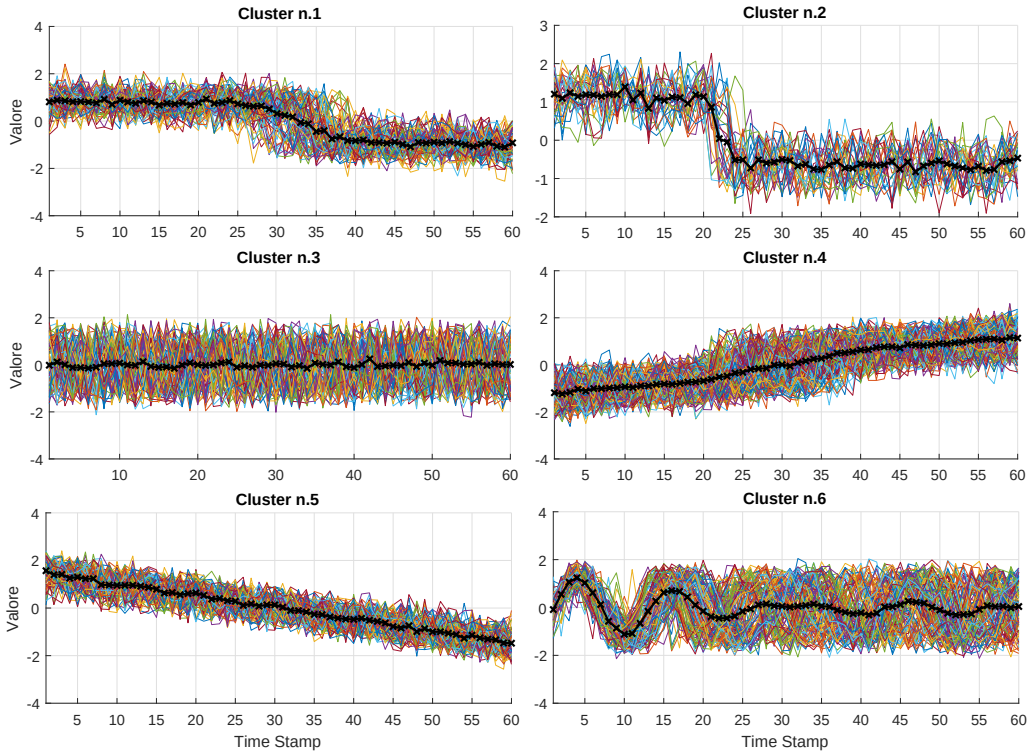

Figura 17: Cluster finali, con i relativi centroidi, restituiti dal TSkmeans per il set "SynteticControl".

effettivamente in grado di scovare sotto sequenze omogenee di serie temporali a cui assegna un peso maggiore a seconda della loro importanza durante il processo di clustering.

Se osserviamo la Figura 16 possiamo notare come, nel caso del set "SynteticControl", il TSkmeans abbia correttamente trovato 6 cluster. In particolare, confrontando tale immagine con la Figura 17, possiamo notare come il cluster 6 presenti un sottospazio interessante che va dall'istante di tempo 1 all'istante 15. Questo particolare sottospazio viene correttamente identificato dal TSkmeans, in quanto i pesi associati agli istanti di tempo del cluster 6 che vanno dall'istante 1 a 15, mostrati in Figura 16, sono sensibilmente più grandi rispetto ai pesi associati agli altri istanti di tempo dello stesso cluster. Questo da una parte indica che gli istanti tra 1 e 15 del cluster 6 hanno un forte valore discriminante all'interno del processo di clustering del TSkmeans, e dall'altra indica che in quello stesso intervallo di tempo sono presenti serie temporali che presentano una forte similarità. Lo stesso ragionamento e le stesse osservazioni possono essere fatte per gli altri cluster del set "SynteticControl" che il

TSkmeans ha scoperto.

Essendo questo risultato valido anche per gli altri set di dati reali possiamo affermare che l'algoritmo TSkmeans è effettivamente in grado di trovare sotto sequenze omogenee di serie temporali in cui si manifestano forti correlazioni fra i dati, ed è inoltre in grado di associare un peso maggiore a tali sequenze durante il processo di clustering. L'algoritmo TSkmeans si dimostra dunque essere un buon algoritmo di clustering partizionale per dati in forma di serie temporale.

5

CONCLUSIONI

Il primo scopo di questa tesi è stato quello di presentare una panoramica sugli algoritmi di clustering per comprenderne le potenzialità e la loro utilità nella ricerca di pattern e strutture nascoste nei più svariati set di dati, presentando le classificazioni di tali algoritmi ed i vari approcci che utilizzano. Abbiamo successivamente parlato delle tecniche di clustering applicate a dati in forma di serie temporale, parlando anche in questo caso della loro classificazione, delle loro caratteristiche principali, delle possibili applicazioni reali di queste tecniche e dei principali algoritmi. Abbiamo poi presentato un nuovo algoritmo di clustering per serie temporali proposto per la prima volta da un gruppo di ricercatori in [1] chiamato "Time Series K-means" (TSkmeans) basato sulla ricerca di sottospazi omogenei nascosti nei set di dati. Abbiamo parlato delle sue caratteristiche, dei suoi punti di forza, della sua complessità e abbiamo fornito una sua implementazione in MATLAB. Infine, abbiamo valutato le prestazioni dell'algoritmo sia su un set di dati sintetico, creato ad hoc per sfruttarne i punti di forza, sia su cinque set di dati reali ricavati da problemi pratici. In entrambi i casi abbiamo valutato l'algoritmo rispetto ad alcuni tra i più utilizzati indici esterni per la valutazione dei processi di clustering e i risultati che abbiamo ottenuto confermano che per la maggior parte dei set di dati testati l'algoritmo TSkmeans ottiene dei risultati migliori in termini di accuratezza e qualità dei cluster finali scoperti rispetto ad altri algoritmi classici per serie temporali.

Possono però esserci particolari set di dati, sopratutto quelli legati a problemi pratici che, per le loro enormi dimensioni e per la presenza in essi di molti dati anomali o rumori, possono mettere in difficoltà l'algoritmo TSkmeans. Abbiamo però osservato che anche nei casi peggiori dei nostri test il TSkmeans riesce a mantenere delle prestazioni solo di poco inferiori rispetto agli algoritmi classici per serie temporali e che dunque risulta essere un ottimo algoritmo di clustering per serie temporali, so-

pratutto per serie che mostrano una forte corrispondenza in varie loro sotto sequenze.

Uno spunto per un eventuale miglioramento dell'algoritmo TSkmeans, come accennato anche in [1], potrebbe essere la ricerca di un metodo automatico per il calcolo del valore ottimale per il parametro α, necessario al funzionamento dell'algoritmo. L'idea e' quella di trovare una tecnica che, a seconda del set di dati a cui viene applicato l'algoritmo, vada a calcolare in modo automatico e senza intaccare troppo sulle prestazioni del TSkmeans un valore di α che consenta di ottenere risultati ottimali per quel determinato set di dati.

BIBLIOGRAFIA

[1] Xiaohui Huang, Yunming Ye, Liyan Xiong, Raymond Y.K. Lau, Nan Jiang, Shaokai Wang - *Time series k-means: A new k-means type smooth subspace clustering for time series data* - Information Sciences, Vol. 367-368, 1 November 2016, Pages 1-13 - http://www.sciencedirect.com/science/article/pii/S0020025516303796 (Cited on pages 7, 8, 39, 44, 46, 70, 71, 85, and 86.)

[2] Saeed Aghabozorgi, Ali Seyed Shirkhorshidi, Teh Ying Wah - *Time-series clustering, A decade review* - Information Systems, Vol. 53, October-November 2015, Pages 16-38 - http://www.sciencedirect.com/science/article/pii/S0306437915000733 (Cited on pages 7, 27, 29, 31, 33, 64, and 65.)

[3] Pang-Ning Tan, Michael Steinbach, Vipin Kumar - *Introduction to Data Mining* - Pearson, 2006 (Cited on pages 11, 22, 23, 60, 62, 63, and 64.)

[4] Matteo Golfarelli - *Introduzione al Data Mining* - Università di Bologna, Settembre 2016 - http://bias.csr.unibo.it/golfarelli//DataMining/MaterialeDidattico/DMISI-Introduzione.pdf (Cited on page 7.)

[5] Stefano Rovetta - *Il problema del clustering* - Università di Genova, Aprile 2013 - ftp://ftp.disi.unige.it/person/RovettaS/lab-inf-II/2002-03/li-ii-3.pdf (Cited on page 15.)

[6] MathWorks, MATLAB - *quadprog* - https://it.mathworks.com/help/optim/ug/quadprog.html (Cited on pages 43, 46, and 53.)

[7] Sakoe, H., Chiba, S. - *Dynamic Programming Algorithm Optimization for Spoken Word Recognition* - IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. 26, No. 1, Pages 43-49, February 1978 - http://ieeexplore.ieee.org/document/1163055/ (Cited on page 34.)

[8] T.Warren Liao - *Clustering of time series data, a survey* - Pattern Recognition, Vol. 38, November 2005, Pages 1857-1874 -

http://www.sciencedirect.com/science/article/pii/S0031320305001305 (Cited on pages 20, 21, and 36.)

[9] J. Z. Huang, M. K. Ng, Hongqiang Rong, Zichen Li - *Automated Variable Weighting in k-Means Type Clustering* - IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 27, No. 5, Pages 657-668, May 2005 - http://ieeexplore.ieee.org/document/1407871 (Cited on page 40.)

[10] V. Niennattrakul, C. A. Ratanamahatana - *On Clustering Multimedia Time Series Data Using K-Means and Dynamic Time Warping* - 2007 International Conference on Multimedia and Ubiquitous Engineering (MUE'07), Seoul, 2007, Pages 733-738 - http://ieeexplore.ieee.org/document/4197360 (Cited on page 30.)

[11] *Sito Ufficiale di MATLAB* - https://it.mathworks.com (Cited on pages 9 and 39.)

[12] Marco Trubian - *Dispensa del Corso di Complementi di Ricerca Operativa* - Università degli Studi di Milano, Marzo 2008 - https://homes.di.unimi.it/~trubian/OttNonLineare.pdf (Cited on page 46.)

[13] Anil K. Jain, Richard C. Dubes - *Algorithms for Clustering Data* - Prentice Hall, 1988 - http://homepages.inf.ed.ac.uk/rbf/BOOKS/JAIN/Clustering_Jain_Dubes.pdf (Cited on page 59.)

[14] C.O.S. Sorzano - *Symmetric matrices and quadratic forms*, Dispense del corso di Algebra - Universidad CEU San Pablo, Dicembre 2013 - http://biocomp.cnb.csic.es/~coss/Docencia/algebra/tema8.pdf (Cited on pages 47 and 52.)

[15] Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze - *An Introduction to Information Retrieval* - Cambridge University Press, 2008 - https://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf (Cited on page 65.)

[16] Y. Zhao and G. Karypis - *Comparison of Agglomerative and Partitional Document Clustering Algorithms* - Technical Report 02-014, University of Minnesota, 2002 - http://handle.dtic.mil/100.2/ADA439503 (Cited on page 66.)

[17] J. Z. Huang, M. K. Ng, Hongqiang Rong, Zichen Li - *Automated variable weighting in k-means type clustering* - IEEE Transactions on

Pattern Analysis and Machine Intelligence, Vol. 27, No. 5, Pages 657-668, May 2005 - http://ieeexplore.ieee.org/document/1407871/ (Cited on page 66.)

[18] Xiaohui Huang, Yunming Ye, Huifeng Guo, Yi Cai, Haijun Zhang, Yan Li - *DSKmeans: A new kmeans-type approach to discriminative subspace clustering* - Knowledge-Based Systems, Vol. 70, Pages 293-300, ISSN 0950-7051, November 2014 - http://www.sciencedirect.com/science/article/pii/S0950705114002664 (Cited on page 67.)

[19] MathWorks, MATLAB - *kmeans* - https://it.mathworks.com/help/stats/kmeans.html (Cited on pages 20, 21, and 68.)

[20] MathWorks, MATLAB - *kmedoids* - https://it.mathworks.com/help/stats/kmedoids.html (Cited on page 68.)

[21] Yanping Chen, Eamonn Keogh, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, Gustavo Batista - *The UCR Time Series Classification Archive* - 2015 - http://www.cs.ucr.edu/~eamonn/time_series_data (Cited on page 76.)

[22] Anthony Bagnall, Jason Lines, William Vickers, Eamonn Keogh - *The UEA & UCR Time Series Classification Repository* - http://www.timeseriesclassification.com (Cited on pages 76 and 77.)

[23] Chotirat Ann Ratanamahatana, Eamonn Keogh - *Everything you know about Dynamic Time Warping is Wrong* - Department of Computer Science and Engineering, University of California, Riverside, 2004 - http://wearables.cc.gatech.edu/paper_of_week/DTW_myths.pdf (Cited on page 76.)