UNIVERSITÀ DI PISA

Dipartimento di Informatica

Corso di Laurea in Informatica

Classificazione Interpretabile di Serie Temporali utilizzando Motifs e Discords Estratti con Matrix Profile

Relatori:

Riccardo Guidotti

Candidato:

Matteo D'Onofrio

Anno Accademico 2019/2020

Sommario

La classificazione delle Serie Temporali è un problema attuale, diffuso e trasversale; troviamo serie temporali in campi come economia (variazione dei prezzi), statistica (indici demografici), medicina (elettrocardiogrammi) ecc. Lavorare su di esse consente di ottenere molte informazioni nascoste dietro la sola apparente sequenza di valori. Purtroppo però, abbiamo a che fare con una quantità di dati sempre superiore, con diversi formati, lunghezze e spesso anche errori, che non consentono sempre di effettuare delle analisi facilmente su di essi. Di conseguenza sono necessari metodi per la loro classificazione, che siano efficienti, efficaci e interpretabili dall'uomo. L'interpretabilità è un fattore decisivo quando si vuole capire il perchè dei risultati ottenuti e avere una visione completa del problema. Nella tesi viene proposto un metodo per la classificazione delle serie temporali basato sulle informazioni fornite da Motifs e Discords, estratti dalla struttura dati denominata Matrix Profile. Gli esperimenti mostrano che, oltre a fornire spiegazioni interpretabili locali e globali sulle ragioni della classificazione, il metodo risulta altamente qualitativo ed efficiente in confronto allo stato dell'arte.

Time series classification is current problem, widespread and transversal: we can find them in many fields such as economy (prices variation), statistics (demographic indices), medicine (electrocardiogram) and so on. Working on them allow us to retrieve many information hidden behind a simple sequence of values. Unfortunately nowadays we deal with an always growing amount of data, with different data type, lenght and sometimes with errors, that don't let us to make analysis in an easy way. So we need efficient, effective and interpretable technique for their classification. The interpretability is a crucial factor when we want to understand the reason below and let people have a clear and complete point of view. In this Thesis we propose a time series classification technique, based on the information given by Motifs and Discords, which are extracted from the Matrix Profile. Experiments show that as well as we get local and global interpretable explanations about classification, we also have high qualitative and efficient results compared to the state of the art.

Indice

Capitolo 1

Introduzione

1.1 Introduzione

Al giorno d'oggi si ritiene molto importante sviluppare modelli di apprendimento automatico, capaci di ottenere in tempi rapidi soluzioni per problemi complessi. Nello specifico, è risultato di particolare interesse l'individuazione di modelli di apprendimento supervisionato per effettuare la classificazione di serie temporali. Tale interesse è suscitato dal fatto che i tradizionali modelli di apprendimento supervisionato, su questo specifico tipo di classificazione, mostrano empiricamente una scarsa capacità di predizione e/o una bassa efficienza.

Alcuni dei campi in cui sono prevalentemente utilizzate le serie temporali sono: l'economia, la medicina, la climatologia, l'epidemiologia, statistica, informatica ecc. Di seguito vengono mostrati degli esempi di utilizzo delle serie temporali. Con riferimento alla Figura 1.1, vengono riportate serie temporali che descrivono: indagine statistica in un aeroporto (in alto a sinistra), anomalie nelle temperatura (in alto a destra), numero di morti per Covid-19 in Italia (in basso a sinistra), confronto della spesa pubblica tra Italia e Germania (in basso a destra).

Tutti questi campi hanno in comune una necessità di metodi accurati, efficienti ed interpretabili per effettuare analisi e predizioni sulle serie temporali.

Di conseguenza è stata effettuata una ricerca finalizzata alla realizzazione di modelli di apprendimento migliori (più accurati ed efficienti), per il problema di classificazioni di serie temporali. Una svolta decisiva è stata introdotta da due risultati distinti descritti di seguito, inerenti l'analisi delle proprietà delle serie temporali e la loro classificazione.

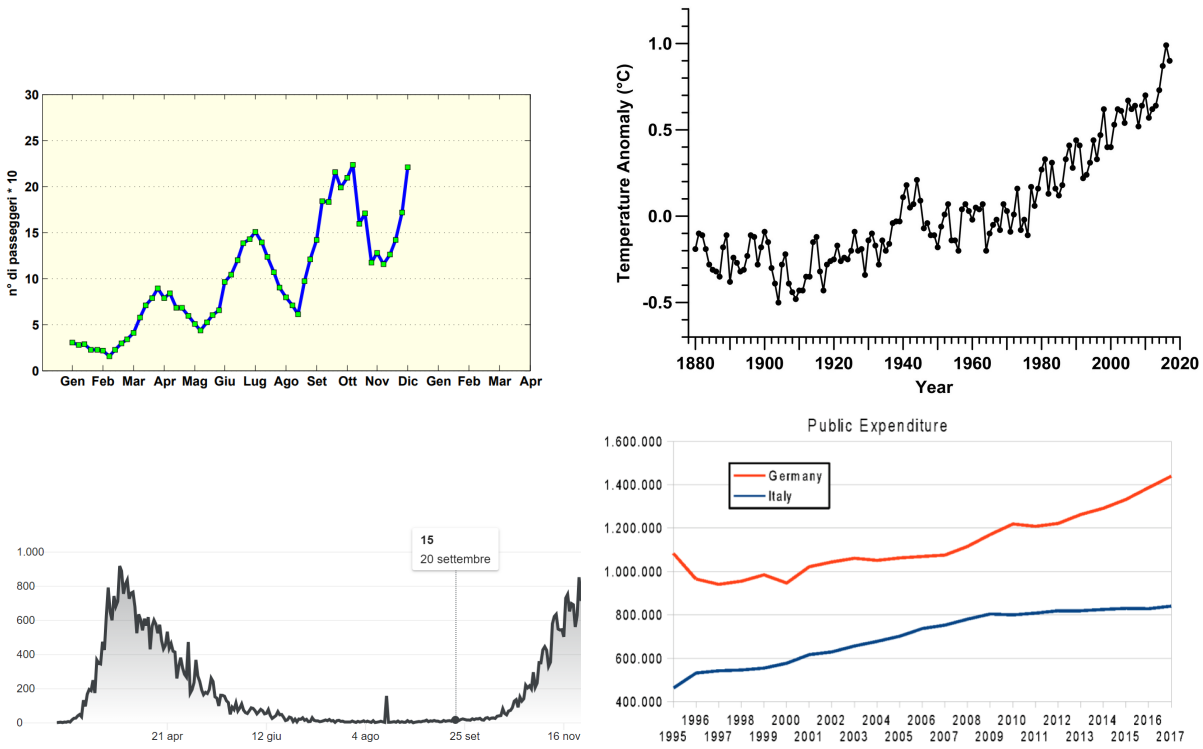

Figura 1.1: Serie temporali appartenenti a diversi settori

- Il primo risultato presenta dei metodi estremamente efficienti, capaci di estrarre da una serie temporale sia le sue sottosequenze più comuni e frequenti, sia le sue sottosequenze più rare e anomale. Le strutture dati cardine su cui si fonda tale risultato, sono il *Distance Profile [7] e il Matrix Profile [3].*

- Il secondo risultato introduce, per un problema di classificazione di serie temporali, le *Shapelets* [6] e i relativi metodi per individuarle. Una *shapelet* è una sottosequenza appartenente a una serie temporale con proprietà capaci di distinguere ed identificare la classe di appartenenza della serie da cui è estratta.

Questi metodi e risultati ottenuti, hanno permesso di delineare meglio gli strumenti di cui si necessitava al fine di definire un modello di apprendimento supervisionato, finalizzato a classificare le serie temporali in maniera accurata, efficiente e interpretabile. La realizzazione di un tale modello, è l'obiettivo della tesi.

Il primo passo effettuato è stato quello di individuare un modello di apprendimento supervisionato, che rendesse facilmente interpretabile le regole di classificazione generate. Il

modello che meglio risponde a questa esigenza è il *Decision Tree Classifier*, che di conseguenza si è scelto di utilizzare. Il *Decision Tree Classifier* però mostra una scarsa capacità di generalizzazione e una scarsa accuratezza per problemi di classificazione di serie temporali. Questo problema è stato risolto con l'introduzione delle *shapelet* nella fase di apprendimento del *Decision Tree*. Dopo aver individutato le shapelet estratte dalle serie temporali su cui si effettua la fase di apprendimento, *la feature discriminante di ogni serie temporale diventa la distanza tra le serie temporali e le shapelet*. Questa nuova feature delle serie temporali viene sfruttata in ogni nodo durante la generazione delle regole di classificazione del *Decision Tree*. L'introduzione e l'uso di questa nuovo tipo di sottosequenze ha permesso una forte aumento dei valori di accuratezza e della capacità di generalizzazione del modello che si vuole realizzare.

Rimane ancora un ultimo problema, quello dell'*efficienza*. Infatti lo spazio di ricerca, dato dalla dimensione dell'insieme delle sottosequenze generate (candidati shapelet) in cui bisogna cercare quelle più significative (*shapelet*), è molto vasto, e una ricerca esaustiva per problemi di grandi dimensioni risulta estremamente lenta. Questo secondo problema è stato risolto tramite l'uso di *Matrix Profile e Distance Profile*. Queste strutture dati, generate tramite metodi estremamente efficienti, permettono di ottenere da ogni serie temporale le sue sottosequenze più significative, ovvero le sottosequenze più comuni contenute in una serie temporale (*Motifs*), e le sottosequenze più rare contenute in una serie temporale (*Discords*). La riduzione dello spazio di ricerca è permessa grazie ad una ricerca delle shapelet da utilizzare nell'apprendimento del modello, concentrata solo su queste specifice sottosequenze più significative delle serie temporali, *Motifs e Discords*. Nonostante la prima riduzione dello spazio di ricerca, l'insieme dei candidati in cui cercare le shapelet si mostrava estremamente vasto. Questo ha spinto verso la ricerca di un'ulteriore riduzione possibile, che non causasse una perdita di accuratezza nella classificazione. La seconda riduzione dello spazio di ricerca è stata realizzata tramite l'algoritmo *K-Medoids* [5], applicato al sottoinsieme generato dalla prima riduzione dello spazio di ricerca.

L'insieme di questi metodi e strategie di ricerca, ha permesso la realizzazione di un modello di apprendimento supervisionato, capace di classificare le serie temporali, in maniera accurata, efficiente ed interpretabile. Una sperimentazione su sei dataset di serie temporali dimostra che il metodo proposto migliora quelli esistenti in tempo e accuratezza garantendo

l'interpretabilità della classificazione.

Il resto della tesi è organizzato come segue. In Sezione 2 viene presentato il background, con le informazioni e i concetti di base su cui si fonda la tesi. In Sezione 3 viene presentato il metodo proposto, con una descrizione iniziale astratta, seguita dallo pseudocodice del modello implementato e la relativa analisi della complessità spaziale e temporale. In Sezione 4 viene effettuata una valutazione qualitativa e quantitativa del modello proposto, seguita dagli esperimenti effettuati. Inoltre nella Sezione 4 vengono mostrati dei confronti tra il modello proposto e dei modelli dello stato dell'arte. Infine in Sezione 5 si riassumono i risultati ottenuti e si propongono direzioni future di ricerca.

Capitolo 2

Background

In questo capitolo vengono introdotti i concetti pricipali su cui si fonda la tesi. Dal momento che l'obiettivo è la classificazione di serie temporali, iniziamo dal concetto di classificazione in Machine Learining, seguito da un'implementazione tramite il Decision Tree. Si introducono poi i metodi necessari all'estrazione di informazioni implicite dalle serie temporali, e infine un'algoritmo di clustering.

2.1 Classificazione delle Serie Temporali

Il Problema della Classificazione

Nel campo del Machine Learning, abbiamo tre tipi di apprendimento che possiamo effettuare, per risolvere un dato problema: *apprendimento supervisionato, apprendimento Non supervisionato, apprendimento per rinforzo* Ai fini della tesi, ci concentreremo solo sul primo problema. L'apprendimento supervisionato ha come obiettivo quello di costruire un modello di classificazione partendo da dei dati di addestramento etichettati al fine di fare previsioni su dati non etichettati. Con il termine *supervisione* si intende quindi che nel dataset i valori di output (dette anche classi o etichette) sono noti per una precedente etichettatura.

Più formalmente, nella tesi siamo interessati ad effettuare una classificazione supervisionata [5], che consiste nell'avere dei dati di input, strutturati come una collezione di record, in cui ogni istanza è caratterizzata da una tupla $\langle x, y \rangle$, dove x è l'insieme degli attributi, di

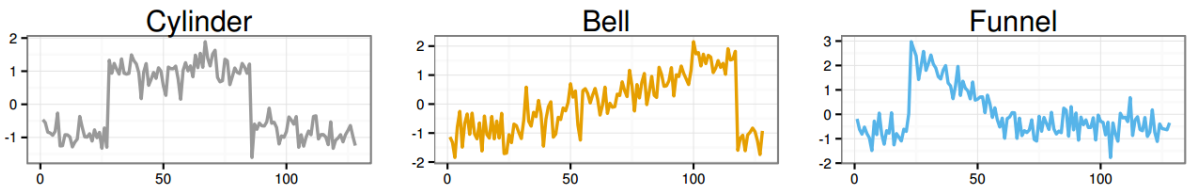

Figura 2.1: Serie temporali di classi diverse.

qualunque tipo (valori continui, nominali, ordinali, binari), che descrivono l'istanza, mentre y, necessariamente nominale, è l'etichetta che definisce la classe di appartenenza.

L'obiettivo dell'apprendimento supervisionato è quello di definire un modello f di classificazione che viene allenato su un training set X, Y dove X è l'insieme delle istanze di training e Y sono le etichette (classi) ad esse associate. La funzione f derivante dall'allenamento sul training prende in input un'istanza x per cui l'etichetta è sconosciuta e le assegna il valore previsto dell'etichetta y. Avendo a disposizione un test set X', Y' per cui le etichette Y' associate a X' sono conosciute, ma tale per cui i record in X' non sono utilizzati per imparare f, è possibile studiare il livello di accuratezza con cui f etichetta i record in X' confrontando le predizioni di $f(X')$ con Y'. f classifica correttamente un'istanza se $f(x) = y$. Un buon modello di classificazione è in grado di classificare correttamente istanze del problema che non ha mai visto precedentemente.

Classificazione di Serie Temporali

In particolare, nella tesi viene affrontato il problema di classificazione di serie temporali, ovvero, le istanze consistono in serie temporali definite come segue [6]:

Definizione 1 (Serie Temporale)**.** *Una serie temporale $T = \langle t_1, \ldots, t_m \rangle$ è un insieme di m valori reali ordinati rispetto al tempo che esprimono la dinamica di un certo fenomeno al variare di esso.*

Tipicamente viene dato in input un dataset composto da una collezione di serie temporali, appartenenti a classi diverse, su cui si vuole imparare a riconoscere le caratteristiche principali utili a classificarle. Tale dataset D è comopsto da un insieme X contenente n istanze (serie temporali), e da un vettore Y della stessa lunghezza, contenente le rispettive etichette di ogni

serie temporale. Formalmente, $X = \{T_1, \ldots, T_n\}$, e $Y = \{c_1, \ldots, c_n\}$. L'obiettivo è quello di imparare una funzione f che mappi lo spazio delle possibili serie temporali nello spazio dei possibili valori della classe. Un algoritmo di apprendimento automatico, dato in input un dataset $D = \langle X, Y \rangle$ di serie temporali annotate con etichetta, cerca di imparare una funzione f che, data una serie temporale T_i è in grado di calcolare il valore c_i. In Figura 2.1, sono presenti 3 serie temporali, appartenenti allo stesso dataset, etichettate con classi differenti.

2.2 Matrix Profile, Motifs e Discords

In questa sezione vengono descritti gli elementi cardine del metodo propsoto nella tesi: *Matrix Profile*, *Motifs* e *Discords*. Prima di introdurle nel dettaglio vengono definite le strutture dati necessarie a descriverle.

Dal momento che nel metodo proposto l'interesse è posto sulle proprietà *locali* di una serie temporale, e non su proprietà globali come media, standadrd deviation, mediana, etc., viene definita una regione locale come una *sottosequqnza* $T_{i,l}$ di una serie temporale T come segue:

Definizione 2 (Sottosequenza)**.** *Una sottosequenza $T_{i,l}$ di una serie temporale T consiste in un sottoinsieme di l valori contigui di T, che ha inizio in posizione i: $T_{i,l} = \langle t_i, t_{i+1}, \ldots, t_{i+l-1} \rangle$, con $1 \leq i \leq m - l + 1$.*

Vengono defninite tutte le possibili sottosequenze di T, tramite l'uso di una *sliding window* di dimensione l, partendo dalla prima sottosequenza individuata e muovendosi verso destra di una posizione alla volta:

Definizione 3 (Insieme di Sottosequenze)**.** *Definiamo con $A = \{T_{1,l}, T_{2,l}, \ldots, T_{m-l+1,l}\}$, con $|A| = m - l + 1$, l'insieme di tutte le possibili sottosequenze di T, di lunghezza l, ottenute tramite lo scorrimento di una sliding window di lunghezza l.*

Si introduce ora un indice di similarità tra due sottosequenze (lo stesso vale per la similarità tra due serie temporali), che permette di definire quanto due sottosequenze sono distanti (o vicine) tra loro.

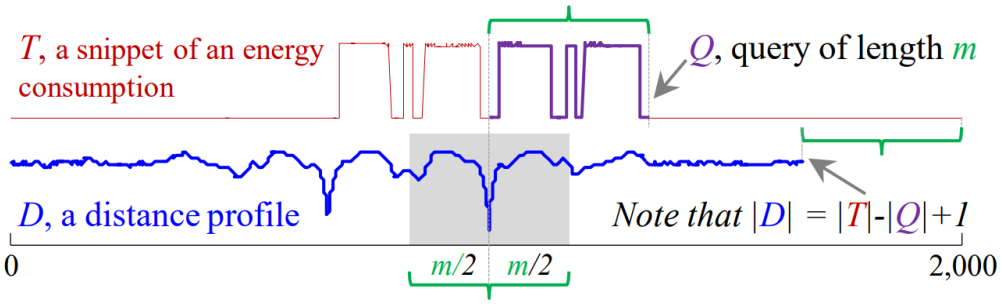

Figura 2.2: Serie temporale (sopra), Distance Profile (sotto) e Trivial Matches

Definizione 4 (Distanza tra Sottosequenze). *Date due sottosequenze* $Q_{i,l} = \langle q_i, q_{i+1}, \ldots, q_{i+l-1} \rangle$ *e* $R_{j,l} = \langle r_j, r_{j+1}, \ldots, r_{j+l-1} \rangle$ *di lunghezza* l*. La distanza tra di esse è definita dalla loro distanza Euclidea:* $dist(Q_{i,l}, R_{j,l}) = \sqrt{\sum_{k=1}^{l}(q_k - r_k)^2}$

Data una generica sottosequenza di *query* Q è possibile calcolare la sua distanza con tutte le sottosequenze A di una serie temporale T, ovvero con tutte le $T_{i,l} \in A$. Il vettore di distanze risultatnte prende il nome di *distance profile*:

Definizione 5 (Distance Profile). *Vettore di distanze tra una sottosequenza* Q *e tutte le sottosequenze* $T_{i,l} \in A$*:* $DP = \{d_{q,1}, \ldots, d_{q,m-l+1}\}$ *con* $d_{q,i} = dist(Q, T_{i,l})$.

Al fine di definire il *Matrix Profile*, l'obiettivo è quello di trovare per ogni sottosequenza di T, la sottosequenza più simile, appartenente a T, diversa da se stessa; ovvero trovare il *Nearest-Neighbor* NN di ciascuna sottosequenza. Per ottenerlo, inizialmente viene calcolata la distanza tra una sottosequenza e tutte le altre. Successivamente viene presa la distanza di valore minimo, la quale implica massima vicinanza con la sottosequenza con cui è stata calcolata. Formalmente, viene presa ogni sottosequenza $T_{i,l} \in A$ e calcolato il relativo *distance profile* DP_i rispetto a tutte le sottosequenze di T. Questo equivale ad applicare la definizione vista sopra, nel quale invece di utilizzare una *query* Q generica, viene considerata una sottosequenza $T_{i,l} \in A$ e calcolata la sua distanza con tutte le sottosequenze $T_{j,l} \in A$, con $1 \leq j \leq m - l + 1$. Come risultato, si ottiene un insieme di distance profile $\{DP_1, DP_2, \ldots, DP_{m-l+1}\}$, uno per ogni differente sottosequenza di T.

Prima di estrarre la distanza minima, bisogna gestire un problema. Essendo ogni DP_i calcolato tra una sottosequenza $T_i \in T$ e tutte le $T_j \in T$, necessariamente ci saranno indici in

DP_i in cui i valori saranno uguali a 0. In posizione $DP_i[i]$ (confronto di $T_{i,l}$ con se stessa), e vicini a 0 appena prima e dopo l'indice i (avendo in comune molti valori con essa). Tali distanze venogno dette *trivial matches* e vengono gestite ignorando i valori delle distanze calcolate tra T_i e le sottosequenze T_j con indice di partenza j compreso tra: $l/2$ prima e dopo i (la zona grigia nella Figura 2.2).

Infine il Nearest-Neighbor viene definito come segue. Supponendo che il $min(DP_1)$ sia un valore che si trova in posizione j nel distance profile, ciò indica che il Nearest-Neighbor della sottosequenza $T_{1,l}$ è la sottosequenza $T_{j,l}$, e la loro distanza è proprio il valore $min(DP_1)$.

In pratica, viene effettuata l'estrazione del valore minimo di ogni *distance profile*, $min(DP_i)$, e del suo relativo indice I_i all'interno del vettore. Tali valori vengono inseriti in due meta-serie temporali denominate *Matrix Profile* e *Matrix Profile Index*, nelle stesse rispettive posizioni i, al fine di permettere di accedere rapidamente al Nearest-Neighbor di una desiderata sottosequenza. Formalmente, le meta-serie temporali sono definitme come:

Definizione 6 (Matrix Profile (MP))**.** *Il Matrix Profile (MP) consiste in un vettore in cui l'i-esimo valore rappresenta la distanza tra la sottosequenza $T_{i,l}$ e la sottosequenza piu vicina ad essa: $MP = \{min(DP_1), min(DP_2), \ldots, min(DP_{m-l+1})\}$.*

Definizione 7 (Matrix Profile Index (MPI))**.** *Il Matrix Profile Index (MPI) consiste in vettore in cui l'i-esimo valore rappresenta l'indice della sottosequenza più vicina a $T_{i,l}$: $MPI = \{I_1, I_2, \ldots, I_{m-l+1}\}$.*

Come mostrato nella Figura 2.3, tanto più una sottosequenza ha un NN simile (vicino), tanto più il relativo valore nel MP sarà basso. Viceversa, per sottosequenze che hanno un NN molto distante il relativo valore nel MP sarà alto.

Ne segue che grazie al Matrix Profile è molto semplice estrarre i *Motif* e *Discord*, ovvero le k sottosequenze più simili o più uniche, da una time series. Formalmente, scelto un valore k i top k Motif vengono definiti come:

Definizione 8 (Top K Motifs)**.** *Data una serie temporale T e il suo rispettivo Matrix Profile MP, i top k Motif sono l'insieme delle k sottosequenze di T che hanno il valore più basso nel MP.*

Definizione 9 (Top K Discords)**.** *Data una serie temporale T e il suo rispettivo Matrix Profile MP, i top k Motif sono l'insieme delle k sottosequenze di T che hanno il valore più alto nel MP.*

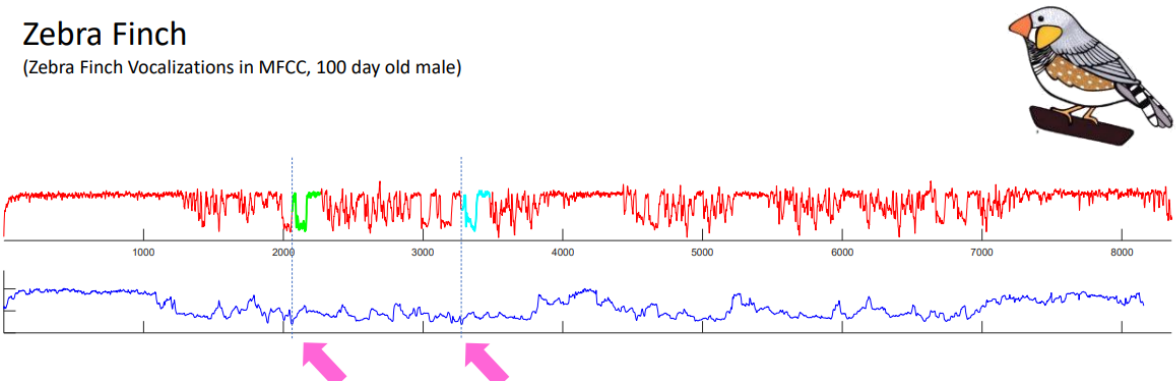

Figura 2.3: Serie temporale (rosso), MP (blu): valori bassi della MP ci indicano dove troviamo pattern simili (distanza piccola). Viceversa valori alti implicano sottosequenze che catturano un comportamento raro.

In pratica, i Motifs mostrano le sottosequenze più comuni, frequenti o ripetute in una serie temporale, mentre i Discords mostrano quelle più rare, e gli eventi anomali.

2.3 Decision Tree

Nella Sezione 2.1 è stato introdotto il problema della classificazione. In questa sezione viene definito uno dei modelli interpretabili di classificazione più diffusi. Viene introdotta un'idea astratta sul suo funzionamento, seguita da un'algoritmo che lo implementa.

Un *Decision Tree* è un albero composto da: nodo radice (non ha archi entranti), nodi interni (hanno 1 arco entrante e almeno 2 archi uscenti) e nodi foglie (1 arco entrante e 0 archi uscenti). La struttura dell'albero viene generata tramite un apprendimento su un insieme di training X, Y. X è un insieme di n serie temporali, ognuna composta da m attributi $\{att_1, \ldots, att_m\}$. Y è un vettore di n etichette, di tipo nominale. Il Decision Tree durante l'apprendimento definisce in ogni nodo interno un vincolo, sul valore di un attributo att_i scelto; ogni possibile valore posto dal vincolo è associato ad un nodo figlio. Ogni nodo foglia contiene il valore dell'etichetta di una classe. Con questo procedimento il Decision Tree genera delle regole composte da congiunzioni di vincoli rappresentati dal cammino: nodo radice-nodo foglia.

Fornita un istanza di test, la classificazione viene effettuata applicando ai suoi attributi le condizioni di test presenti nei nodi e seguendo il cammino in base al risultato del test. Una

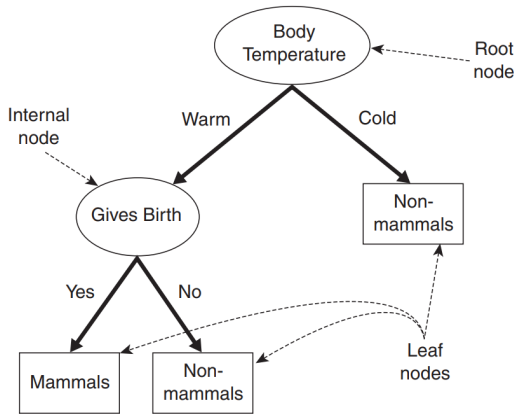

Figura 2.4: Rappresentazione di un Decision Tree per un problema di classificazione di mammiferi.

volta arrivata in un nodo foglia, l'istanza viene etichettatata con il valore di classe della foglia.

Nella Figura 2.3, nel nodo radice e nodo interno ci sono gli attributi scelti, mentre nei rami i valori dei vincoli posti. Nei nodi foglia invece c'è il valore dell'etichetta, con cui vengono classificate le istanze: *Se la temperatura corporea dell'animale è calda, ma non partorisce, non è un mammifero.*

Prima di introdurre uno degli algoritmi più diffusi per la realizzazione di un Decision tree, viene definita l'*Entropia*, la misura dell'omogeneità di una certa distribuzione:

Definizione 10 (Entropia E)**.** *Fornito un dataset D, composto da istanze appartenenti a C classi differenti, sia $p(C)$ la frequenza di ogni classe $p(C) = \frac{\#IstanzeDiClasseC}{|D|}$, l'Entropia si definisce come $E(D) = -\sum_{i=1}^{C}(p(i)log(p(i)))$, con $0 \leq E(D) \leq \log_2(C)$.*

L'etropia è un indice di quanto i valori delle etichette delle istanze y_i in un dataset, sono distribuiti omogeneamente:

- $E(D) = 0$ quando tutte le istanze di D, appartengono ad una sola classe (*massima omogeneità*)
- $E(D) = \log_2(K)$ quando le K classi differenti (almeno 2), sono perfettamente equidistribuite in D (*massima disomogeneità*)

Algoritmo di Hunt L'algoritmo riceve in input un insieme di training X, Y, che viene inserito nel nodo radice. Procede poi ricorsivamente come segue:

1. Misura l'entropia del dataset nel nodo attuale
2. Inizia la ricerca del *Best attribute*: scandisce ogni attributo nel dataset ricevuto, e per ciascuno, sceglie uno alla volta tutti i valori che tale attributo assume in tutte le istanze
3. Per ciascuno di questi valori, si effettua uno split del dataset in sotto-dataset tale che, ogni sotto-dataset ha solo le istanze che soddisfano il vincolo imposto sull'attributo scelto
4. Si misura l'entropia dei sotto-dataset ottenuti
5. Si procede iterativamente per tutti gli attributi e per tutti i valori
6. Viene scelto infine, lo split che più di tutti, ha diminuito il valore dell'entropia, rispetto a quella misurata inizialmente (o se vogliamo, lo split che massimizza la differenza tra l'entropia calcolata sul dataset iniziale e l'entropia ottenuta negli split).
7. Vengono generati tanti nodi figli, quanti i differenti vincoli, mutuamente esclusivi, imposti sull'attributo su cui è stato splittato il dataset. Ad ogni nodo figlio viene rimosso l'attributo su cui è stato effettuato lo split
8. Si procede ricorsivamente finche non si raggiungono criteri di stop

Tramite questo algoritmo viene effettuata una ricerca *Greedy* nello spazio dei Decision tree, perché l'algoritmo costruisce il modello che minimizza il numero di livelli e massimizza la diminuzione dell'entropia, nel cammino radice-foglia. Nel punto 6 è chiaro quanto sia fondamentale l'uso dell'entropia nella scelta del miglior split: esso viene determinato in base a quanto più riesce ad abbassare il valore dell'entropia, dopo aver effettuato lo split, rispetto al valore che aveva precedentemente. Questo concetto può essere sinteticamente ed equivalentemente racchiuso con il *Gain*:

Definizione 11 (Gain)**.** *Fornito un dataset D con entropia E_D, effettuato uno split (su un attributo Att con vincoli V) su di esso che produce n sotto-dataset distinti, ognuno con entropia $E_1, \ldots, E_n$, il Gain è la differenza di entropia ottenuta da tale split: Gain(Att, V)= $E_D - (\sum_{i=1}^{n} E_i)$*

Quindi, la miglior riduzione dell'entropia equivale alla massimizzazione del valore di Gain.

2.4 Shapelets

Viene introdotto di seguito in maniera astratta, uno dei concetti chiave della tesi, definendo il contesto in cui esso si rivela utile. Nella classificazione in generale, il modello durante l'apprendimento sceglie degli attributi e definisce vincoli su di essi. Nell'ambito della classficazione delle serie temporali, il modello durante l'apprendimento può scegliere delle sottosequenze di diverse serie temporali e definire vincoli su di essi. Ciò implica che invece di cercare le caratteristiche di singoli valori, un modello di calssificazione come il Decision Tree pone l'attenzione sulle sottosequenze delle serie temporali. Infatti, come dimostrato empiricamente in [6], tali sottosequenze permettono l'uso di algoritmi più veloci, accurati e interpretabili, rispetto a coloro che non ne fanno uso, in questo tipo di classificazione. Di seguito viene dettagliato un particolare tipo di sottosequenza denominata Shapelet.

Informalmente una *Shapelet* è una sottosequenza di una serie temporale, che più di ogni altra sottosequenza, distingue ed identifica la propria classe di appartenenza.

Viene definita di seguito una funzione necessaria per sfruttare le proprietà delle shapelet:

Definizione 12 (Subsequence Distance $SD(T, s)$)**.** *La Subsequence Distance $SD(T, s)$ è la distanza minima tra una sottosequenza s lunga l e una serie temporale T. La $SD(T, s)$ è ottenuta generando l'insieme A di tutte le sottosequenze di T di lunghezza l e poi calcolando la distanza minima tra s e ogni sottosequenza $T_{i,l} \in A$. La distanza minima viene restituita da $SD(T, s)$.*

Una definizione formale di *Shapelet* è la seguente [6]:

Definizione 13 (Shapelet)**.** *Fornito un dataset D, scelto un valore di soglia OSP su cui effettuare lo split, una $Shapelet$ sul dataset D è una sottosequenza s_i estratta da una serie temporale $T_i \in D$, tale che: $(Gain(s_i), OSP) \geq Gain(s_j, OSP')$, per ogni altra sottosequenza s_j e ongli altro possibile valore di soglia OSP' scelto.*

Una shapelet, o un insieme di shapelet, possono essere sfruttati per derivare delle features altamente discriminative da poter utilizzare in un problema di classificazione.

In pratica, data una shapelet s e una serie temporale T, la distanza $SD(T, s)$ della shapelet s da T diventa la feature che descrive la serie temporale T. Con riferimento alla Figura 2.5, se una data serie temporale T è "vicina" alla prima shapelet allora può essere classificata come "Clovis", altrimenti come "Avonlea".

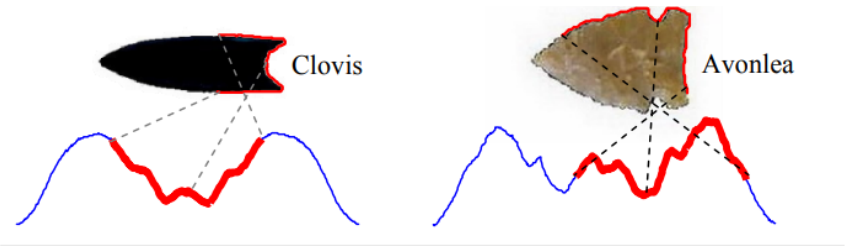

Figura 2.5: Punte di freccia appartenenti a classi diverse. Dopo aver convertito la loro forma in serie temporali [2], le Shapelet (in rosso) sono capaci di distinguere bene le differenti classi.

2.5 Clustering e K-Means

Di seguito viene introdotto l'*apprendimento non supervisionato* e un metodo appartenente a questa classe di apprendimento, che si rivelerà molto utile ai fini della tesi.

Nell'*apprendimento non supervisionato*, al contrario di quello supervisionato, sono presenti dati senza etichetta o classe. Con le tecniche non supervisionate è possibile scoprire la struttura dei dati ed estrapolare informazioni utili. In queste tecniche non si può però contare su una variabile nota relativa al risultato o su di una funzione di ricompensa.

Tra le tecniche di apprenidmento non supervisionato, si introduce quella del *Clustering* [5]. Tale tecnica consiste nel ricevere in input una collezione di istanze non etichettate X, e trovare al suo interno cluster (gruppi) di istanze accomunate dall'avere valori simili di alcuni o tutti gli attributi. In particolare, l'obiettivo di tale metodo è quello di: *individuare cluster le cui istanze minimizzano la distanza tra istanze appartenenti ad uno stesso cluster, e massimizzano la distanza tra cluster differenti.*

Tra gli algoritmi di Clustering, esiste una siddivisione basata sul loro approccio: *algoritmi con approccio partizionale*, in cui ogni istanza può appartenere ad un solo cluster; *algoritmi con approccio gerarchico*, in cui ogni istanza può appartenere a più cluster contemporaneamente. Ai fini della tesi, viene introdotto il seguente algoritmo di clustering con approccio partizionale:

K-Means

1. Si sceglie inizialmente un valore intero k
2. Si scelgono randomicamente k istanze dal dataset, dette *Centroidi*. Ogni centroide definisce il centro di un Cluster

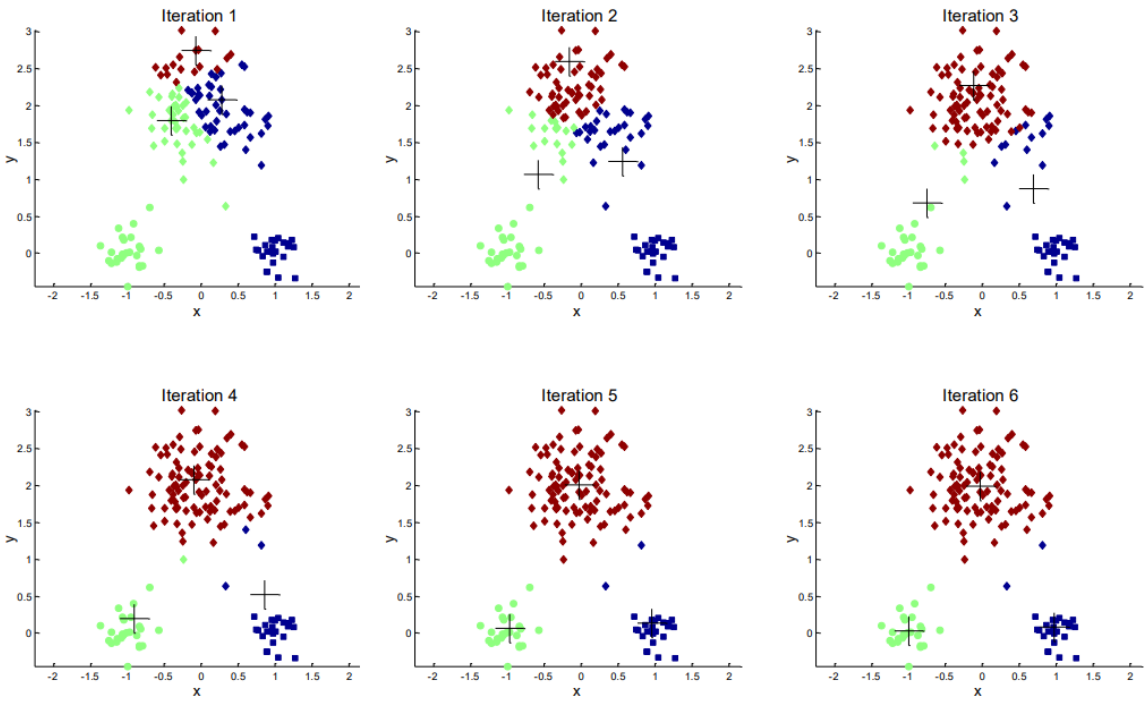

Figura 2.6: Iterazioni dell'algoritmo K-Means

3. Si calcola la distanza (Euclidea) tra ogni istanza nel dataset e tutti i k centroidi. Ogni istanza viene assegnata al cluster individuato dal centroide con distanza minore da essa.

4. In ogni cluster viene calcolato il nuovo centroide, definito come un punto nello spazio, che ha il valore di ogni attributo uguale alla media dei valori dello stesso attributo di tutti i punti nello stesso cluster.

5. Si reiterano i punti 3 e 4, finché il calcolo del nuovo centroide genera un centroide diverso da quello precedente (oppure si definiscono criteri di stop).

Nella Figura 2.6 vediamo come, partendo da 3 centroidi selezionati randomicamente, in 6 iterazioni, l'algoritmo converge verso dei centroidi significativi, rappresentanti le 3 classi di cui è composto il dataset in esame.

Il *K-Medoids* è una variante del K-Means. Tale tecnica calcola il centroide come *medoide* del cluster, ovvero il punto appartenente al clsuter che minimizza la somma delle distanze con tutti i punti nel cluster. In questo modo, invece di avere K punti nello spazio come centro dei

cluster (che potrebbero verosimilmente non esistere), si ottengono k istanze del dataset come centroidi. I cluster e i centroidi (medoidi) ottenuti, forniscono molte informazioni sul dataset:

- L'analisi del centroide(medoide), permette di capire quali sono i valori medi degli attribui delle istanze appartenente al cluster centrato in esso.
- L'analisi della distribuzione delle istanze del dataset nei vari cluster, evidenzia come sono ripartite le istanze nelle classi identificate.

Capitolo 3

Metodo Proposto

L'obiettivo della tesi è classificare serie temporali usando una tecnica di apprendimento supervisionata, implementata da un modello che sia *accurato*, *efficiente* ed *interpretabile*. Di seguito viene descritto il metodo proposto per la realizzazione di un modello che soddisfi queste esigenze, giustificando le scelte implementative fatte.

3.1 Scelte Implementative

Al fine di soddisfare i requisiti di accuratezza e interpretabilità, viene scelto come modello supervisionato il *Decision Tree*. Come illustrato in Sezione 2.3, la buona interpretabilità deriva dal fatto che un Decision Tree effettua l'apprendimento tramite la generazione di regole logiche. Tale struttura delle regole permette di analizzare il modello da un punto di vista globale (cammino nodo radice-nodo foglia) e locale (vincolo su un singolo nodo), rendendo chiaro come avviene la classificazione. Nonostante il Decision Tree sia un ottimo modello per la classificazione in generale, presenta molti problemi e bassa accuratezza, in un problema specifico di classificazione di serie temporali.

Per ovviare a questo problema, il primo approccio è stato quello di utilizzare come features di classificazione le *shapelet* per modellare le serie temporali. Infatti, essendo queste sottosequenze estremamente rappresentative rispetto alla classe di appartenenza permettono di migliorare le performance di ogni tipo di classificatore per le serie temporali. D'altro canto, anche l'apprendimento di un Decision Tree basato su shapelet ha mostrato empiricamente due

Algorithm 1: TSCMP

Input : D - training dataset $\langle X, Y \rangle$,
$candidatesGroup$ - define if consider only Motifs, only Discords, or both,
$maxDepth$ - maximum depth of the DT,
$minSL$ - min number of sample instances needed to define a leaf in the Decision Tree,
$removeCandidates$ - true if a shapelt cannot be reused in a subsequent split,
l - sliding window size,
k - number of motifs and/or discords extracted from a time series,
$numMedoids$ - number of chosen medoids for K-Medoids algorithm

Output: DT - trained Decision Tree

1 $S \leftarrow ExtractCandidates(X, l, k, candidatesGroup)$; // retrieve the shapelet candidate set
2 $DatasetDist \leftarrow ComputeDistances(X, S)$; // compute distances between time series and candidates
3 $DT \leftarrow DecisionTree(maxDepth, minSL, S, numMedoids, removeCandidates)$; // create tree
4 $DT \leftarrow BuildTree(DatasetDist)$ // train the Decision Tree
5 **return** DT;

problemi. Il primo è dato dal vasto spazio di ricerca,a causa della generazione e analisi di tutti i possibili candidati shapelet che comportano un lungo tempo di esecuzione. Il secondo problema è dato dalla accuratezza non troppo elevata riportata negli esperimenti, che mostrano la scarsa capacità di generalizzazione di questo modello su questo specifico problema.

A seguito di questa analisi, si è ritenuto necessario definire un nuovo modello, modificando il classico Decision Tree realizzato tramite l'algoritmo di Hunt, in modo tale da effettuare un apprendimento basato su shapelet "locali" che fanno riferimento solo alle serie temporali presenti nel nodo correntemente analizzato per lo split. L'algoritmo per il nuovo modello proposto prende il nome di *Time Series Classification based on Matrix Profile (TSCMP).*

3.2 Time Series Classification based on Matrix Profile

Di seguito viene presentato in dettaglio l'algoritmo *Time Series Classification based on Matrix Profile (TSCMP)* tramite pseudocidici che descrivono ogni funzione.

Nell'Algoritmo 1 viene presentato l'algoritmo TSCMP. Nella linea 1 viene calcolato l'insieme dei candidati shapelet S. Poiché la scelta del tipo di candidato estratto (Motifs, Discords o

Algorithm 2: ExtractCandidates

```
1  Function ExtractCandindates(X, l, k, CandidatesGroup):
2      S ← ∅
3      for x_i ∈ D do                                   // for each time series selected
4          selectedCandidates ← {}
5          MP_i ← MatrixProfile(x_i, l)                 // compute matrix profile on instance
6          if CandidatesGroup == 0 then selectedCandidates ← ExtractMotifs(MP_i, k);
7          if CandidatesGroup == 1 then selectedCandidates ← ExtractDiscords(MP_i, k);
8          if CandidatesGroup == 2 then selectedCandidates ← ExtractBoth(MP_i, k);
9          S ← S ∪ selectedCandidates                   // add candidates to main list
10 return S;
```

Algorithm 3: ComputeDistances

```
1  Function ComputeDistances(D, S):
2      DatasetDist ← ∅
3      for x_i, y_i ∈ D do                              // for each time series and relative label selected
4          Dist_i ← {}                                  // contains the min distances between x_i and each candidate c_j
5          for c_j ∈ S do                               // for each candidate
6              Dist_i[j] ← SD(x_i, c_j)
7          DatasetDist_i ← (Dist_i, y_i)
8  return DatasetDist;
```

entrambi), fissato il dataset su cui viene allenato il modello, genera risultati diversi, la scelta del tipo di candidato da estrarre è fornita dall'utente ed è definita dal parametro *candidatesGroup* ricevuto in ingresso.

Con riferimento all'Algoritmo 2: per ogni serie temporale $x_i \in X$ viene calcolata la *Matrix Profile* MP_i, da cui vengono estratte, le k sottosequenze (Motifs) con distanza minore dal proprio NN e/o le k sottosequenze (Discords) con distanza maggiore dal proprio NN. L'estrazione dei candidati dall'insieme dei Motifs e Discords, garantisce che in S ci siano solo i candidati più significativi.

Successivamente, nella linea 2 dell'Algoritmo 1, viene calcolato il dataset *DatasetDist*. In tale dataset, per ogni serie temporale $x_i \in X$ viene riportata la relativa etichetta y_i e la distanza

Algorithm 4: BuildTree

```
1  Function BuildTree(DT):
2      if DT.node.stoppingCriteria == True then return DT.node;
3      S' ← getSpecificCandidates(DT.S, DT.node);        // take only candidates respecting conditions
4      S'' ← KMedoids(S', DT.numMedoids);                // take only the K-Medoids candidates
5      s, OSP ← FindSplit(S'', DT.node, DT.removeCandidates);   // find values for split
6      DT.node.left ← DecisionTree.newNode(DT.node.instances[s] ≤ OSP);   // apply split
7      DT.node.right ← DecisionTree.newNode(DT.node.instances[s] > OSP);
8      BuildTree(DT.node.left);                          // recursive call
9      BuildTree(DT.node.right);
10 return DT;
```

Algorithm 5: FindSplit

```
1  Function FindSplit(S'', Dt.node, DT.removeCandidates):
2      bestGain ← 0, bestOSP ← 0, shapelet ← ∅
3      for c_i ∈ S'' do                                  // for each shapelet candidate
4          if DT.removeCandidatesandAlreadyUsed(c_i) then continue;
5          OSP_i, Gain_i ← bestOSP(DT.node, c_i)         // find OSP with best Gain
6          if Gain_i > bestGain then (bestGain, bestOSP, shapelet) = (Gain_i, OSP_i, c_i);
7  Return shapelet, bestOSP;
```

minima $SD(x_i, c_j)$ tra la serie temporale x_i e ogni candidato shapelet $c_j \in S$. Il calcolo del *DatasetDist* è mostrato nella funzione illustrata nell'Algoritmo 3.

Infine, nelle linee 3 e 4 dell'algoritmo 1, viene inizializzato il *Decision Tree* con i parametri forniti in ingresso; poi viene passato il *DatasetDist* e viene invocata la funzione di apprendimento *BuildTree*, che è ricorsiva e parte dal nodo radice del *Decision Tree* e si ripete per ogni split binario nel sottoalbero che viene creato fino a quando non viene incontrata la condizione di terminazione. La funzione di apprendimento descritta dall'Algoritmo 4 genera il *Decision Tree* in maniera ricorsiva iniziando dal nodo radice. Di seguito si descrivono le varie fasi:

- La shapelet scelta in ogni nodo per definire il vincolo, viene individuata all'interno di un sottoinsieme $S'' \subset S$. S'' viene individuato specificamente per il sottoinsieme di serie temporali contenuto nel nodo corrente; di seguito viene mostrato come ottenere tale

sottoinsieme. Inizialmente (linea 3 Algoritmo 4) viene definito $S' \subset S$, con S' composto dai candidati shapelet estratti dalle sole serie temporali presenti nel nodo corrente. Essendo S', generalmente, ancora molto grande, viene eseguito l'algoritmo di clustering *K-Medoids* su S' (linea 4 Algoritmo 4), estraendo solo i $numMedoids$ candidati più distintivi dal sottoinsieme dei candidati precedentemente selezionati. Viene ottenuto un nuovo sottoinsieme $S'' \in S'$, con S'' composto dai $numMedoids$ ***medoidi*** ottenuti. La generazione di un sottoinsieme S'', effettuata in ogni nodo, permette di generare vincoli specializzati sullo specifico sottoinsieme di serie temporali.

- Nella linea 5 dell'Algoritmo 4 vengono scelti i valori su cui definire il vincolo ed effettuare lo split. La shapelet e il valore di soglia scelti per definire il vincolo, vengono individuati all'interno del sottoinsieme S'' da *FindSplit*, funzione riportata nell'Algoritmo 5. L'obiettivo di *FindSplit* è quello di trovare la shapelet $c_i \in S''$ e il relativo valore di soglia OSP_i, da definire come condizione di test per effettuare lo split nel nodo corrente. La scelta di tali valori ha come obiettivo quello di a massimizzare il *Gain* ottenuto (vedi Sezione 2.3) dallo split.

- *FindSplit* per ottenere tale condizione di test, effettua una ricerca esaustiva nell'insieme dei candidati $c_i \in S''$. Per ogni candidato c_i determina il valore di soglia OSP_i, ed effettua uno "split di prova", finalizzato a misurarne il $Gain_i$ ottenuto (linea 7 nella funzione 5). Lo "split di prova" consiste nell'inserire tutte le serie temporali x_i con distanza minima $SD(x_i, c_i) < OSP_i$ nel nodo figlio sinistro, mentre quelle con distanza minima $SD(x_i, c_i) > OSP_i$ vengono messe nel nodo figlio destro.

- Dopo aver analizzato il *Gain* ottenuto dallo "split di prova" su ogni candidato, viene scelta la coppia c_i, OSP_i, il cui split nel nodo corrente, su tali valori, riporta il valore di *Gain* maggiore.

- Nelle linee 6-9 dell'Algoritmo 4 viene applicato lo split al sotto-dataset contenuto nel nodo corrente, generati i nodi figli destro e sinistro, ed effettuata la chiamata ricorsiva su di essi.

- Si procede ricorsivamente finché un nodo soddisfa i criteri di stop (linea 2), e viene definito come nodo foglia.

Come viene mostrato nella sezione sperimentale, l'uso di questo algoritmo permette di soddisfare l'obiettivo della tesi garantendo una buona accuratezza (data dalla specializzazione dei vincoli), efficienza (data dalla forte riduzione dello spazio di ricerca) e buona interpretabilità (data dall'uso di una variante del Decision Tree).

3.3 Analisi della Complessità in Spazio e Tempo

In questa sezione viene analizzata la complessità spaziale e temporale dall'algoritmo *TSCMP*.

Inizialmente viene analizzato lo spazio di ricerca dell'algoritmo *TSCMP*. L'analisi viene fatta sia da un punto di vista *globale*, in cui si considera il numero totale di candidati generati $|S|$, e sia da un punto di vista *locale* in cui si consdiera il numero di candidati da esplorare in ogni nodo $|S''|$. Dato un dataset D di dimensione $|D| = n$; scelto un valore k che definisce il numero di Motifs e Discords estratti da ogni serie temporale (viene considerato il caso pessimo, in cui per ogni serie temporale vengono estratti entrambi); scelto un valore $numMedoids$ per l'algoritmo *K-Medoids*:

- lo *spazio di ricerca globale* $|S|$ è $O(2kn)$, poiché per ognuna delle n serie temporali, estraggo al caso pessimo k *Motifs* e k *Discords*;

- lo *spazio di ricerca locale* $|S''|$ è $O(numMedoids)$, poiché come visto nell'algoritmo 5, in ogni nodo lo spazio di ricerca viene ridotto estraendo i $|numMedoids|$ candidati più distintivi da S'.

Di seguito viene effettuato uno studio della complessità temporale dell'algoritmo *TSCMP* attraverso un'analisi asintotica dei suoi tempi di esecuzione, considerando sempre il caso pessimo. Dato un dataset D contenente $|D| = n$ serie temporali; fissata la lunghezza di ogni serie temporale $T_i \in D, |T_i| = m$; fissata la dimensione dell'insieme dei candidati generati $|S| = O(2kn)$; fissata la dimensione del primo sottoinsieme di candidati individuato $|S'| = O(2kn)$; fissata la dimensione dei candidati selezionati $S'' \in S, |S''| = numMedoids$; scelta la dimensione dei candidati $s_i \in S, |s_i| = l$; scelto il numero k di Motifs e Discords

estratti da ogni serie temporale; scelto il numero $numMedoids$ di Medoidi estratti dall'algoritmo *K-Medoids*; scelto il numero t di iterazioni dall'algoritmo *K-Medoids*; si ottengono le seguenti complessità temporali:

- *Distance Profile* $= O(m \log(m))$. Distance profile calcolato tramite l'algoritmo *MASS (Mueen's Algorithm for Similarity Search)* [4], tra una sottosequenza e una serie temporale.
- *Matrix Profile* $= O(m^2)$. Matrix profile calcolato tramite l'algoritmo *STOMP (Scalable Time series Ordered Matrix Profile)* [1].
- *Extract Candidates* $= O(2m^2nk))$. Per ognuna delle n serie temporali viene calcolato il relativo Matrix Profile MP con costo $O(m^2)$; in ogni MP vengono estratti, al caso pessimo, k Motifs e k Discords.
- *Compute Distances* $= O(2kn^2m \log(m))$. Per ognuna delle n serie temporali viene calcolato il *Distance Profile* con ogni candidato $s_i \in S$, $|S| = O(2kn)$.
- *Find Split* $= O(n^2 \log(n) numMedoids)$. Per ogni candidato $s_i \in S''$, ordino la lista delle n distanze minime tra ogni serie temporale contenuta nel nodo corrente e il candidato s_i scelto, con costo $\Theta(n \log(n))$. Su questa lista delle distanze minime cerco l'OSP migliore, effettuando al caso pessimo, n split con costo costante.
- *KMedoids* $= O(2knI numMedoids)$. Sia I il numero di iterazioni, viene calcolata la distanza *Euclidea* tra ogni candidato $c_i \in S'$, con $|S'| = O(2kn)'$ e ogni medoide $|numMedoids|$.
- *Build Tree* $= O(n^2 \log(n) numMedoids)$. Il costo di questa funzione, analizzata asintoticamente, è determinato solamente dal costo di *Find Split*, essendo i costi delle altre funzioni richiamate inferiori. *getSpecificCandidates* effettua una scansione delle serie temporali contenute nel nodo corrente con costo $O(n)$. *K-Medoids* esegue l'algoritmo con costo $O(nI numMedoids)$.

Sulla base di questi risultati, possiamo stimare il *costo totale in tempo* come la somma tra i costi delle funzioni: *Extract Candidates*, *Compute Distances* e *BuildTree*. Essendo generalmente

il numero di serie temporali n molto maggiore della lunghezza m di ogni serie temporale $n >> m$, si può intuire come il costo della funzione *Build Tree* domini tutte le altre: *costo totale in tempo di TSCMP* è $O(n^2 log(n) numMedoids)$.

Capitolo 4

Esperimenti

In questo capitolo sono riportati gli esperimenti, effettuati al fine di valutare le prestazioni del metodo *TSCMP*. Viene definito prima l'ambiente nel quale sono stati effettuati gli esperimenti, seguito dalla loro esposizione e valutazione qualitativa; in seguito viene effettuata una valutazione dei risultati ottenuti, al variare dei parametri di TSCMP. Infine, vengono effettuati confronti con altri modelli allo stato dell'arte valutando le differenze attraverso una valutazione quantitativa. *Il codice sorgente è pubblicato su Git-Hub*[1]

4.1 Experimental Setting

Per quanto riguarda il setting sperimentale degli esperimenti, si descrivono di seguito: *(i)* le librerie utilizzate per la realizzazione dell'algoritmo TSCMP, *(ii)* i dataset su cui sono stati effettuati gli esperimenti, *(iii)* le misure di valutazione delle prestazioni, e *(iv)* i modelli dello stato dell'arte utilizzati per effettuare un confronto. Gli esperimenti sono stati effettuati su una macchina con le seguenti specifiche hardware: processore *Intel i5-3570K da 3.4GHz*, memoria RAM da *8 GB DDR3 da 1800 MHz* e sistema operativo *Windows 10*.

[1] `https://github.com/matteo-Donofrio-unipi/D-Onofrio-thesis`

Librerie

L'algoritmo *TSCMP* che implementa il modello proposto, è scritto in Python 3.8.3. Le librerie utilizzate per la realizzazione dell'algoritmo sono descritte di seguito.

- *scikit-learn*[2] per implementare i modelli *Decision Tree Classifier e K-Means*
- *pyts*[3]: per implementare il modello *ShapeletTransformation*
- *pandas*[4]: per implementare le strutture dati tabellari i dataset
- *numpy*[5]: per implementare vettori, matrici e operazioni su di essi
- *matplotlib*[6]: per effettuare stampe e plot di grafici
- *matrixprofile*[7]: per calcolare *Distance vector, Matrix profile, Motifs e Discords*
- *binarytree-pypi*[8]: per implementare l'albero binario
- *scipy*[9]: per il calcolo della distanza euclidea tra sottosequenze

Dataset

I sei dataset utilizzati per effettuare gli esperimenti sono stati presi dal sito "UEA and UCR Time Series Classification Repository" e sono descritti di seguito. Informazioni riassuntive dei dataset sono riportati in Tabella 4.1.

Italy Power Demand: le serie temporali riportano il consumo energetico in Italia nell'arco di 12 mesi. L'obiettivo è distinguere le osservazioni fatte tra Ottobre-Marzo e Aprile-Settembre.

Gun Point: le serie temporali riportano il tracciamento del movimento di due attori entrambi aventi la pistola nella fondina. Il primo estrae la pistola e la punta verso un obiettivo, mentre

[2] https://scikit-learn.org/stable/modules/classes.html
[3] https://pyts.readthedocs.io/en/stable/api.html
[4] https://pandas.pydata.org/docs/reference/index.html
[5] https://numpy.org/doc/stable/
[6] https://matplotlib.org/api/index.html
[7] https://github.com/target/matrixprofile-ts
[8] https://pypi.org/project/binarytree/
[9] https://docs.scipy.org/doc/scipy/reference/api.html

Dataset	n	m	classi	training	test
Arrow Head	211	251	3	17,00%	83,00%
ECG200	200	96	2	50,00%	50,00%
ECG5000	5000	140	5	10,00%	90,00%
GunPoint	200	150	2	25,00%	75,00%
ItalyPowerDemand	1096	24	2	6,00%	94,00%
PhalangesOutlinesCorrect	2568	80	2	70,00%	30,00%

Tabella 4.1: Descrizione dataset utilizzati per gli esperimenti

il secondo punta semplicemente il dito verso l'obiettivo. Le misurazioni sono fatte tracciando le coordinate della mano *sugli assi x e y*. L'obiettivo è riconoscere l'attore che estrae la pistola.

Arrow Head: le serie temporali rappresentano punte di freccia, le cui forme, provenienti da immagini, sono state convertite in serie temporali tramite un metodo *angle-based* [2]. Sono presenti 3 classi, 3 tipi distinti di punte di freccie. L'obiettivo è riconoscere la classe di appartenenza di ogni serie temporale.

ECG-200: le serie temporali tracciano la frequenza cardiaca registrata tramite un elettrocardiogramma. L'obiettivo è distinguere 2 classi: battito normale e infarto miocardico.

ECG-5000: le serie temporali tracciano la frequenza cardiaca registrata tramite un elettrocardiogramma. L'obiettivo è distinguere 5 classi in base a differenti attività cardiache.

Phalanges Outlines Correct: le serie temporali rappresentano i risultati di test di un modello allenato nella segmentazione delle falangi di una mano nelle radiografie effettuate sulle mani. L'obiettivo è classificare le immagini correttamente segmentate.

La possibilità di avere un classificatore interpretabile è importante in particolare per casi come quello di *ECG-200* e *ECG-5000* dove il medico vuole essere a conoscenza dei motivi per cui il sistema di supporto alle decisioni usate restituisce una classe piuttosto che un'altra.

Durante gli esperimenti, ogni dataset D viene suddiviso in 2 sotto-dataset: $\langle X, Y \rangle$ e $\langle X', Y' \rangle$. I modelli effettuano la fase di apprendimento sul sotto-dataset $\langle X, Y$ e successivamente la fase di test sul sotto-dataset $\langle X', Y' \rangle$.

Misure di Valutazione

Le metriche utilizzate per misurare le prestazioni dei modelli valutano: *(i)* il *tempo di esecuzione* impiegato durante la fase di apprendimento, e *(ii) l'accuratezza della classificazione* ottenuta nella fase di test. Nello specifico, il *tempo di esecuzione* impiegato nella fase di apprendimento è definito come il tempo, in secondi, necessario alla generazione del Decision Tree e di tutte le regole logiche, oppure nel caso dei metodi utilizzati per il confronto, per il training del modello. L'*accuratezza della classificazione* è definita come il numero di istanze nel test set classificate correttamente rispetto al numero di istanze totali nel test set:

$$Accuracy = \frac{\#\text{istanze classificate correttamente}}{\#\text{istanze totali}}$$

Modelli A Confronto

Di seguito vengono descritti due modelli dello stato dell'arte con cui è stato confrontato il modello proposto nella tesi. Il confronto è basato sulla comparazione delle misurazioni di tempo e accuratezza, ottenute dall'esecuzione dei differenti modelli sugli stessi dataset. Il vincolo è che anche i modelli a confronto siano interpretabili. Di conseguenza vengono scartati come possibili competitor tutti quei metodi che non forniscono una classificazione interpretabile e per i quali non è possibile capire la logica globale del modello di classificazione. Al fine di una comparazione più veritiera, il confronto è limitato a modelli che riescono ad esprimere la classificazione in forma di Decision Tree sfruttando shapelet definite su serie temporali. In questo modo si forza il modello di classificazione a restituire la stessa tipologia di modello e di interpretabilità garantendo così un confronto totalmente comparabile rispetto ad accuratezza e tempo di esecuzione.

Il primo competitor è lo *ShapeletTransformation (ST)*[10], un modello di apprendimento supervisionato il cui apprendimento è basato sul contenuto informativo fornito dalle shapelet. Nello specifico, lo ShapeletTransformation effettua l'apprendimento generando prima l'insieme dei candidati shapelet S estraendo ogni possibile sottosequenza di dimensione l da ognuna delle n serie temporali nel dataset. Poi estraendo da S le $\frac{n}{2}$ shapelet più significative. Infine definisce un Decision Tree, in cui in ogni nodo viene definito un vincolo basato sulla distanza

[10] `https://pyts.readthedocs.io/en/stable/api.html`

tra le serie contenute nel nodo corrente e le $\frac{n}{2}$ shapelet scelte. E' importante sottolinear che. a differenza del metodo proposto, la scelta delle shapelet avviene a priori globalmente e non localmente come per TSCMP. Inoltre, TSCMP usa come shapelet dei pattern derivati da Motif e Discord, mentre ShapeletTransformation ricerca le sottosequenze che definiscono le shapelet secondo la definizione riportata in [6]

Come secondo competitor viene scelto il *Decision Tree Classifier (DTC)* descritto in Sezione 2.3 equipaggiato con features come descritto di seguito. L'apprendimento del Decision Tree Classifier avviene infatti sul *DatasetDist* (vedi Algoritmo 1), lo stesso dataset generato dall'algoritmo TSCMP, contenete le distanze tra ogni serie temporale $T_i \in D$ e ogni candidato $s_i \in S$ trovate grazie al Matrix Porfile. Successivamente il DatasetDist viene passato al DTC per effettuare l'apprendimento, generando regole logiche su di esso. Si sottolinea che in questo caso le shapelet vengono considerate una volta sola e non raffinate ad ogni step come in Sezione 3.2

4.2 Valutazione Qualitativa

In questa sezione viene mostrata l'esecuzione dell'algoritmo *TSCMP* sul dataset *ECG200*. Nel seguente esperimento viene posta l'attenzione sull'*interpretabilità* del modello generato. Infatti dal modello TSCMP è possibile osservare sia le regole di classificazione da un punto di vista *globale* attraverso l'insieme delle regole generate dal modello definite dai cammini nodo radice-nodo foglia; sia le regole di classificazione da un punto di vista *locale* attraverso una esemplificazione che mostra come avviene la classificazione di ogni serie temporale tramite l'applicazione delle regole definite in ogni nodo.

In questo esperimento, il modello proposto nella tesi viene testato sul dataset *ECG200*. L'algoritmo *TSCMP* viene eseguito inizializzando il Decision Tree con i seguenti parametri individuati per il dataset[11] *ECG200*: *candidatesGroup* = Discords, *maxDepth* = 4, *minSL* = 20, *removeCandidates* = True, l = 30, k = 3, *numMedoids* = 100.

La fase di apprendimento del modello genera il Decision Tree in Figura 4.1 che permette di osservare le regole di classificazione da un punto di vista *globale*. Nello specifico, con riferimento alla Figura 4.1, ogni nodo interno del Decision Tree è identificato dall'*identificativo*

[11] I dettagli relativi allo studio dei parametri sono presentati nella prossima sezione.

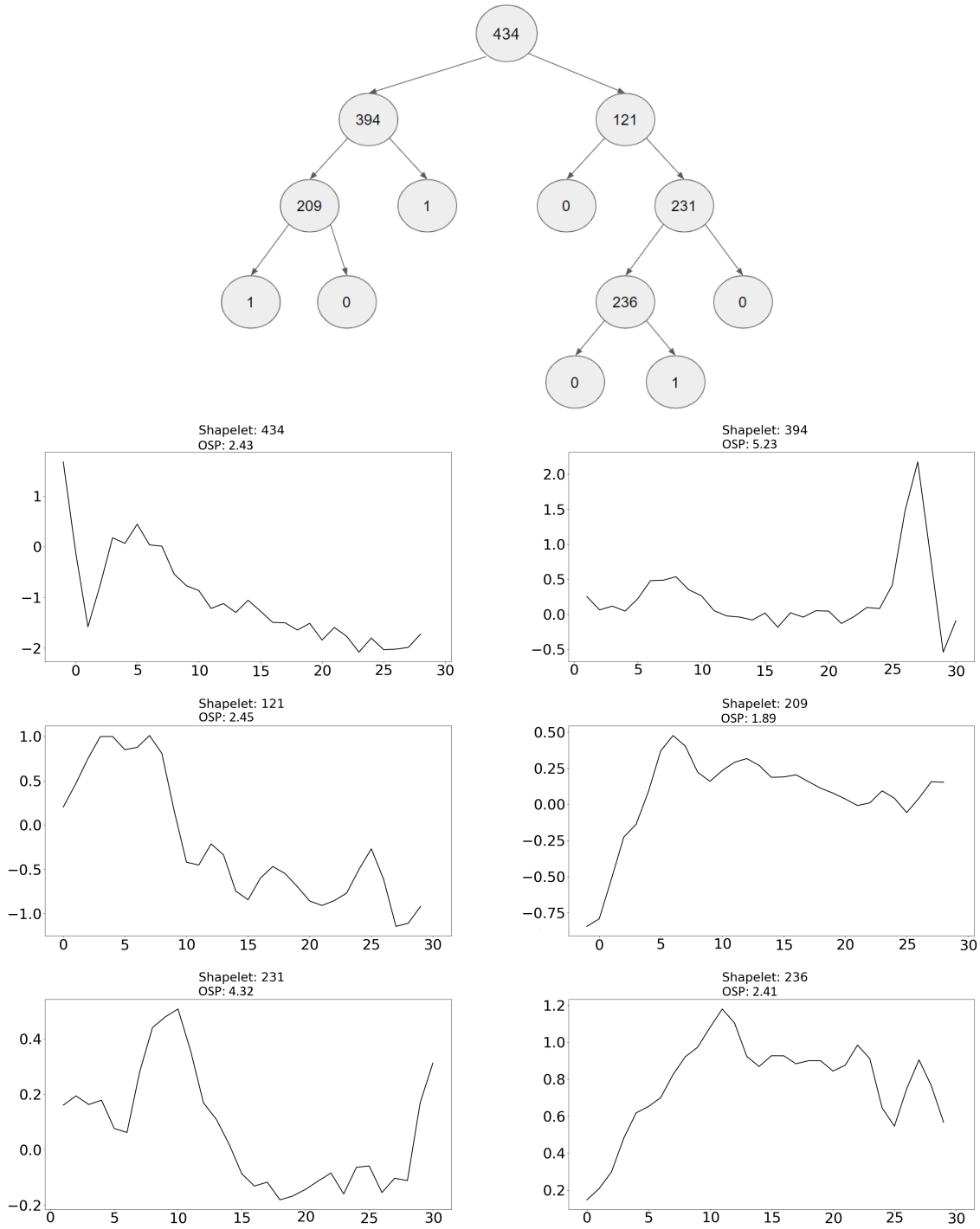

Figura 4.1: Decision Tree generato dalla fase di apprendimento con le corrispondenti shapelet.

della shapelet scelta come condizione di test al suo interno. Ogni nodo foglia è identificato dal valore dell'etichetta prevista: 1: *infarto miocardico*, 0: *battito regolare*. Nella Figura 4.1, vengono riportate in basso le shapelet utilizzate dal Decision Tree.

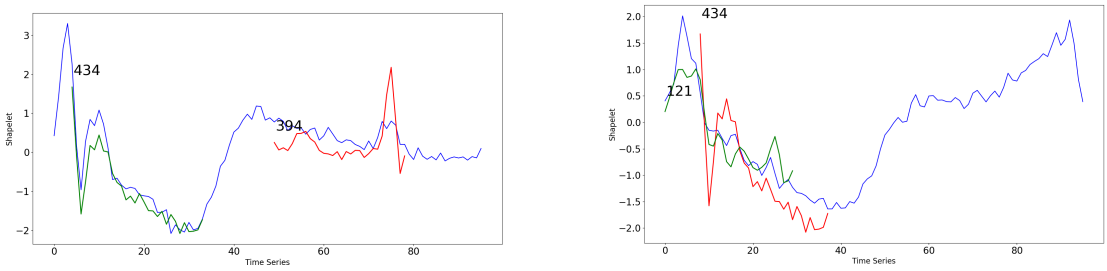

Figura 4.2: Classificazione di due istanze di test.

In Figura 4.2 vengono riportate le classificazioni di due istanze in cui si evidenzia come vengono applicate le condizioni di test definite in ogni nodo. Tramite questa visualizzazione è possibile osservare le regole di classificazione da un punto di vista *locale*. Con riferimento alla Figura 4.2 sinistra, la serie temporale blu x da classificare, nel decision tree incontra due shapelet: la 434 e la 394. In verde viene evidenziata la 434 poiché la distanza tra x e la 434 è minore dell'OSP (come se fosse contenuta in x). Invece la 394 viene evidenziata in rosso poiché la distanza tra x e la 394 è maggiore dell'OSP (ovvero non risulta contenuta in x). A seguito di tali controlli, x viene classificata con il valore 1, ovvero il paziente è a rischio di *infarto miocardico*. Dall'altro lato invece, con riferimento alla Figura 4.2 destra, x viene classficata con il valore 0, ovvero il paziente mostra un battito regolare, infatti è possibile osservare un diverso percorso effettuato e quindi diversi controlli: viene evidenziata in rosso la shapelet 434 poiché la distanza tra x e la 434 è maggiore dell'OSP. La shapelet 121 invece viene colorata in verde, essendo la distanza tra x e la 121, minore dell'OSP.

4.3 Valutazione Parametri

In questa sezione viene effettuata una valutazione dei risultati ottenuti, eseguendo esperimenti sul modello proposto al variare dei parametri di ingresso di TSCMP. Vengono riportati gli esperimenti effettuati sui dataset ECG200 e GunPoint. E' stata scelta inizialmente la seguente *configurazione base* dei parametri: *candidatesGroup* = Discords, *maxDepth* = 3, *minSL* = 20, *removeCandidates* = True, l = 20, k = 2, *numMedoids* = 20. Successivamente, sono stati effettuati gli esperimenti, variando il valore di un singolo parametro alla volta. Con riferimento alla

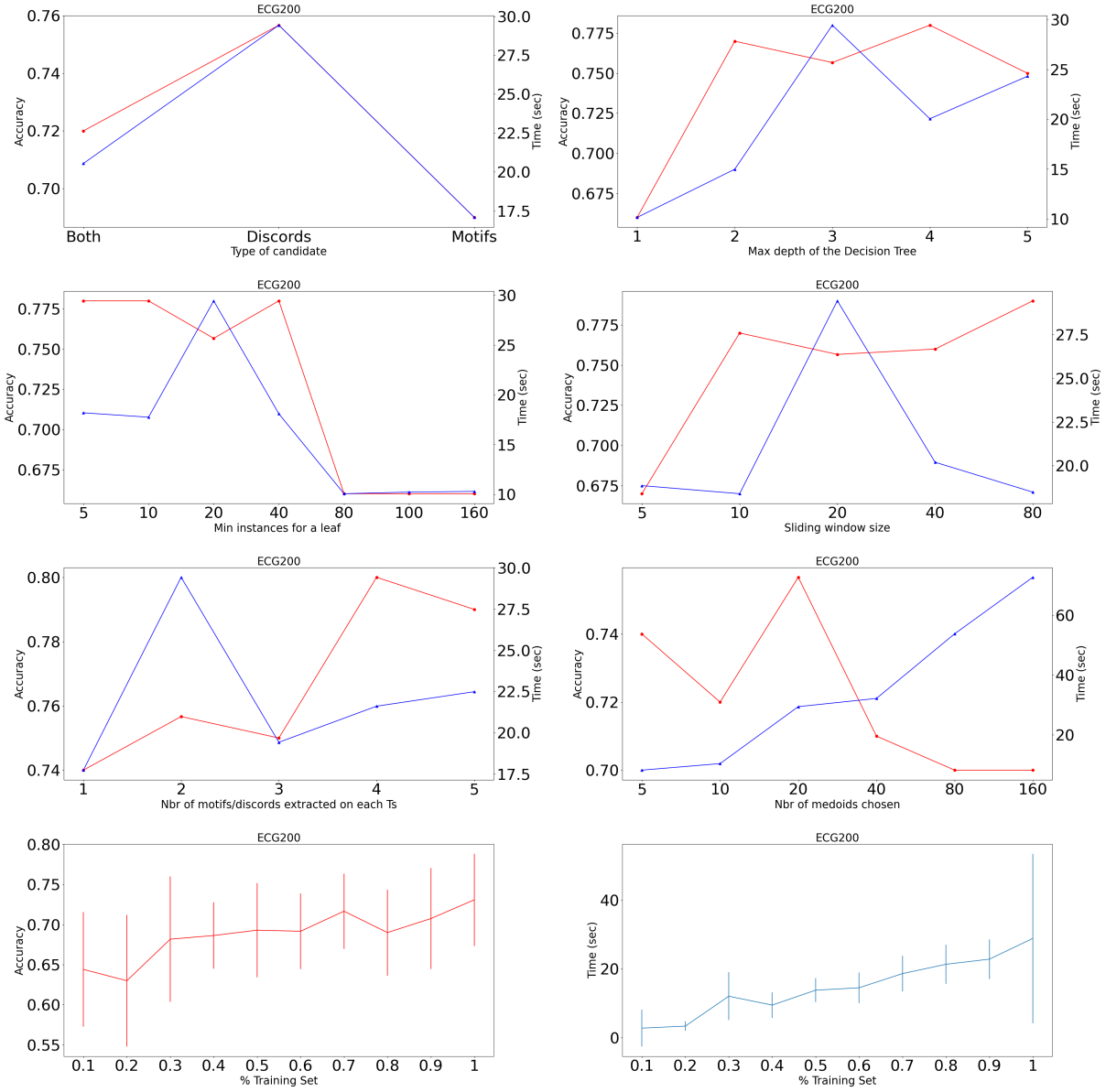

Figura 4.3: Accuratezza (rosso) e tempo di esecuzione (blu), ottenute variando i parametri in ingresso di *TSCMP* eseguito sul dataset *ECG200*.

Figura 4.3, ogni grafico, mostra i risultati medi di valutazione, ottenuti facendo variare il parametro riportato, e mantenendo tutti gli altri parametri con gli stessi valori della *configurazione base*.

Dai valori di valutazione visibili nella Figura 4.3 è possibile dedurre che quando il *numero di medoidi scelti* supera il valore $20 - 30$, il modello tende ad avere un valore di accuratezza decrescente e un tempo di esecuzione, ovviamente, crescente. Tale osservazione è di fonda-

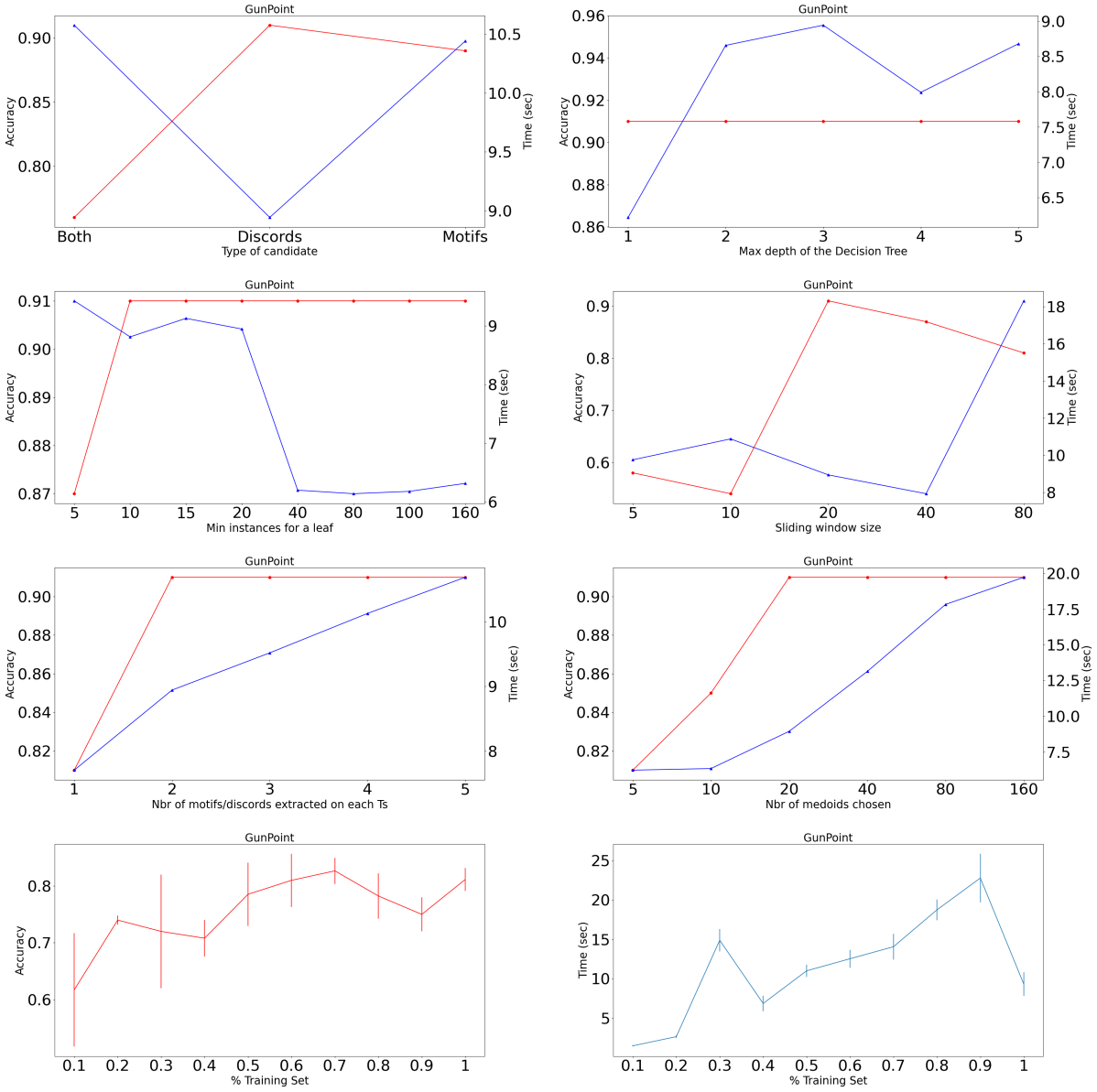

Figura 4.4: Accuratezza (rosso) e tempo di esecuzione (blu), ottenute variando i parametri in ingresso di *TSCMP* eseguito sul dataset *GunPoint*.

mentale importanza, poiché il *numero di medoidi* definisce la dimensione dell'insieme S'' in cui in ogni nodo, durante l'apprendimento, viene effettuata la ricerca della shapelet.

Per quanto riguarda il tipo di candidati, i *Discords* si sono rivelati i migliori in termine di accuratezza, dimostrando che *le shapelet più discriminative di una classe, sono quelle più rare.*

Infine, sempre con riferimento alla Figura 4.3, viene mostrata la variazione della media in accuratezza la e relativa deviazione stardard ottenute dell'esecuzione del modello al variare

della percentuale dell'insieme di apprendimento scelta. Dal grafico si nota un discreto valore di accuratezza ottenuto anche negli esperimenti in cui vengono scelte basse percentuali dell'insieme su cui viene effettuato l'apprendimento. Ovviamente, la crescita della percentuale dell'insieme di apprendimento è direttamente proporzionale alla crescita dell'accuratezza e alla decrescita della deviazione standard. Tali risultati suggeriscono che è possibile considerare solo un sottoinsieme $X_{\%} \subset X$, su cui effettuare l'apprendimento, senza avere un drastica perdita del valore di accuratezza.

Per quanto riguarda la Figura 4.4, è interessante osservare, il grafico che mostra la variazione del parametro k, ovvero del *numero di candidati estratti da ogni serie temporale*. Tale grafico mostra che con valori di $k \geq 2$, i valori di accuratezza medi convergono sul valore 0.92, dimostrando la *capacità di scelta specializzata dei candidati, effettuata dal modello*.

L'unico parametro di cui non viene riportata la variazione, è *RemoveCandidates*, ovvero il parametro booleano con cui si sceglie se rimuovere un candidato s dall'insieme dei candidati S, dopo che tale candidato è stato scelto come shapelet dal *Decision Tree*. La variazione di *RemoveCandidates* non viene riportata poiché, per quanto riguarda la valutazione quantitativa: mostra risultati migliori, in ogni dataset, quando settato a *True*, ovvero quado si rimuovono i candidati scelti. Per quanto riguarda la valutazione qualitativa, effettuare più di una volta, il confronto tra un serie temporale e lo stesso candidato, genera un confronto *ovvio* o addirittura inutile, poiché qualsiasi risultato ottenuto nel primo confronto sarà replicato successivamente.

4.4 Confronto con Modelli Interpretabili Esistenti

Di seguito vengono riportati i risultati ottenuti dal confronto di *TSCMP* con due modelli dello stato dell'arte *Shapelet Transformation* e *Decision Tree Classifier* descritti in precedenza.

Per il *TSCMP* è stata scelta una configurazione di parametri[12] specifica. Tale configurazione è stata determinata sulla base di un confronto dei risultati ottenuti da numerose esecuzioni di *TSCMP*, al variare dei parametri di ingresso, su tutti i dataset citati in tabella 4.1. I parametri che mediamente, hanno riportato valori migliori di *accuratezza nella classificazione*, sono stati selezionati. Per lo *ShapeletTransformation*, l'unico parametro di ingresso fornito è la dimen-

[12] *candidatesGroup*=Discords, *maxDepth*=3, *minSL*=20, *removeCandidates*=True, *l*=20, *k*=2, *numMedoids*=20

Dataset	Accuracy			Tempo Esecuzione		
	TSCMP	ST	DTC	TSCMP	ST	DTC
ArrowHead	0.58	**0.64**	0.54	9.56	38.12	**8.00**
ECG200	**0.84**	0.80	0.58	18.19	25.17	**5.43**
ECG5000	**0.90**	0.89	0.36	**124.43**	1002.84	169.99
GunPoint	**0.90**	0.84	0.72	8.67	28.34	**7.23**
ItalyPowerDemand	**0.92**	0.91	0.53	9.84	**5.64**	27.06
PhalangesOutlinesCorrect	**0.70**	0.68	0.39	699.39	3777.52	**274.51**

Tabella 4.2: Accuratezza e Tempo di Esecuzione per i vari dataset e metodi testati. Il migliore è evidenziato in grassetto.

sione della *Sliding Window*. Al fine di ottenere un confronto valido, la dimensione della *Sliding Window* fornita è 20, la stessa dimensione fornita al *TSCMP* Infine per il *Decision Tree Classifier*, i parametri forniti in ingresso sono in comune con quelli di *TSCMP*, e sono i seguenti: *criterio di split*=entropia, *maxDepth*=3, *minSL*=20.

Dai risultati ottenuti, riportati in Tabella 4.2, si nota il vantaggio ottenuto dall'algoritmo TSCMP. Per quanto riguarda i valori di *accuratezza nella classificazione*, *TSCMP* ottiene valori migliori dei competitor 5 volte su 6; il *Decision Tree Classifier* ottiene pessimi risultati, mostrando la bassa capacità di apprendimento di tale modello su problemi di classificazione di serie temporali. Il *gap in accuratezza* evidente, tra il *TSCMP* e il *Decision Tree Classifier*, mostra il beneficio apportato dalla scelta accurata e specializzata, effettuata in ogni nodo, dal *TSCMP*, rispetto alla scelta del *Decision Tree Classifier* guidata solamente dalla massimizzazione dell'entropia. Lo *ShapeletTransformation* ottiene risultati inferiori, ma molto vicini a quelli del *TSCMP*.

Il vantaggio concreto, introdotto dall'algoritmo *TSCMP* rispetto a *ShapeletTransformation* è visibile nei *tempi di esecuzione dell'apprendimento*. Infatti, TSCMP mostra in ogni esperimento tempi *estremamente* inferiori rispetto a quelli impiegati dallo *ShapeletTransformation*. E' interessante notare come al crescere della dimensione del dataset di apprendimento, cresce il *gap* tra i tempi di esecuzione del *TSCMP* e *ShapeletTransformation*. Il *gap nei tempi di esecuzione*, tra il *TSCMP* e lo *ShapeletTransformation*, mostra il beneficio introdotto dall'uso di metodi e

strutture estremamente efficienti come: il *Matrix Profile* per l'estrazione di *Motifs e Discords* e il *K-Medoids* per l'individuazione dei candidati più significativi.

Capitolo 5

Conclusioni

Nella tesi è stato proposto un nuovo modello di apprendimento supervisionato, realizzato tramite l'algoritmo *TSCMP* capace di *classificare serie temporali* in maniera *accurata, efficiente ed interpretabile*. L'*interpretabilità* è stata ottenuta grazie all'uso di un *Decision Tree* come modello che permette di osservare le regole di classificazione generate da un punto di vista *globale* (insieme delle regole definite dai cammini nodo radice-nodo foglia), e da un punto di vista *locale* (regole definite in ogni nodo). L'*accuratezza* è stata ottenuta grazie alla generazione di regole di classificazione specializzate, basate sulla ricerca di candidati shapelet specifici, per ogni insieme di istanze considerato. L'*efficienza* è stata ottenuta grazie all'uso di metodi estremamente rapidi, quali: *Matrix Profile e Distance Profile* per la generazione dell'insieme dei candidati shapelet, seguito dal *K-Medoids* per la riduzione dello spazio di ricerca. Gli esperimenti effettuati mostrano il vantaggio introdotto da *TSCMP* rispetto ai modelli dello stato dell'arte.

Tra i possibili lavori futuri, si includono: la realizzazione di esperimenti su altri dataset, eventualmente con dimensioni molto maggiori; il confronto del modello proposto con metodi non interpretabili; l'estensione dell'algoritmo *TSCMP* finalizzata ad aggiungere la possibilità di lavorare su serie temporali multivariate.

Ringraziamenti

Vorrei dedicare qualche riga a tutti coloro che mi sono stati vicini in questo percorso di crescita personale e professionale.

Ringrazio innanzitutto il mio relatore Riccardo Guidotti, per il supporto dato e l'enorme mole di conoscenza che mi ha trasmesso in questo percorso.

Ringrazio i miei genitori e mio fratello, a cui devo tutto ciò che ho ottenuto nella vita, la cui presenza ed influenza costante mi ha permesso di raggiungere anche questo obiettivo.

Ringrazio la mia fidanzata Giulia, per esserci stata in ogni momento, garantendomi un supporto morale inestimabile.

Ringrazio i miei amici, per la loro costante vicinanza.

Infine un ringraziamento speciale, va al mio amico, collega e mentore Giuseppe, senza il quale il mio rapporto con le discipline scientifiche non sarebbe la stesso.

Bibliografia

[1] Andrew Van Benschoten, Austin Ouyang, Francisco Bischoff, and Tyler Marrs. Mpa: a novel cross-language api for time series analysis. *Journal of Open Source Software*, 5(49):2179, 2020.

[2] Eamonn Keogh, Li Wei, Xiaopeng Xi, Sang-Hee Lee, and Michail Vlachos. Lb_keogh supports exact indexing of shapes under rotation invariance with arbitrary representations and distance measures. In *Proceedings of the 32nd international conference on Very large data bases*, pages 882–893. Citeseer, 2006.

[3] A Mueen and E Keogh. Time series data mining using the matrix profile: A unifiying view of motif discovery, anomaly detection, segmentation, classification, clustering and similarity joins. *IEEE Big Data 2017*, 2017.

[4] Abdullah Mueen, Yan Zhu, Michael Yeh, Kaveh Kamgar, Krishnamurthy Viswanathan, Chetan Gupta, and Eamonn Keogh. The fastest similarity search algorithm for time series subsequences under euclidean distance, August 2017. `http://www.cs.unm.edu/mueen/FastestSimilaritySearch.html`.

[5] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to data mining*. Pearson Education India, 2016.

[6] Lexiang Ye and Eamonn Keogh. Time series shapelets: a new primitive for data mining. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 947–956, 2009.

[7] Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova, Nurjahan Begum, Yifei Ding, Hoang Anh Dau, Diego Furtado Silva, Abdullah Mueen, and Eamonn Keogh. Matrix profile i: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets. In *2016 IEEE 16th international conference on data mining (ICDM)*, pages 1317–1322. Ieee, 2016.