Analisi Time Series

Progetto di Data Science e Machine Learning

De Stefano Manuel, Esposito Flavio, Iannone Emanuele

Luglio 2019

Indice

Capitolo 1

Introduzione

1.1 Il contesto

Una **time series** (TS), o serie temporale, è una sequenza di dati ordinati che variano nel tempo, tipicamente rilevati ad intervalli regolari. Alcuni esempi sono gli elettrocardiogrammi, l'andamento delle maree, i valori azionari.

Figura 1.1: Un insieme di TS, ciascuna di colore diverso

Le TS consistono in una sequenza finita di *osservazioni* periodiche fatte in un certo intervallo di tempo, ciascuna caratterizzata da uno o più valori discreti.

Il numero di osservazioni registrate nell'intervallo temporale definisce la cosiddetta **lunghezza** della time series, cioè la sua **dimensionalità**.

L'**analisi delle time series** si occupa di studiare questi dati per estrarre informazioni statisticamente rilevanti per poi applicarle per diversi scopi:

- **Previsioni**: prevedere l'andamento di alcuni eventi nel futuro, come ad esempio l'evoluzione delle azioni in borsa e le oscillazioni della Terra, tramite un *modello statistico*;
- **Stime**: approssimare le informazioni per ricavare conoscenza generale dei fenomeni analizzati;
- **Anomaly detection**: identificare e studiare la presenza di comportamenti anomali nei fenomeni analizzati.

I migliori risultati in questo ambito si ottengono nei task di:

- **Classificazione**, che consiste nella costruzione di un modello di ML[1] che apprende dalle TS il modo in cui assegnare ad una nuova TS una certa etichetta appartenente ad un insieme finito di classi. L'apprendimento è di tipo **supervisionato**.
  Tipicamente, i modelli usati per affrontare questo task sono basati su reti neurali ricorrenti, in particolare le **LSTM**, usate in questo lavoro e descritte nel seguito;
- **Clustering**, il cui obiettivo è quello di partizionare le serie temporali in gruppi, chiamati *cluster*, a seconda della loro distanza in modo che le TS assegnate allo stesso cluster siano quanto più simili possibile. L'apprendimento è di tipo **non supervisionato**.
  Particolare attenzione va data alla scelta della metrica di similarità e dell'algoritmo di clustering: usare metriche e tecniche di clustering differenti potrebbe portare a risultati completamente diversi.
  In questo lavoro verrà usato l'algoritmo di clustering **k-Means**, con **distanza euclidea** come metrica di similarità, e verrà presentata un'altra soluzione che fa uso della distanza **DTW**[2], descritta nel seguito.

L'analisi di TS e la costruzione di un modello di ML devono affrontare problematiche specifiche per questo tipo di dati, ad esempio:

[1] Machine Learning
[2] Dynamic Time Warping

- Ogni singola osservazione di una TS è memorizzata in una colonna del dataset, quindi una TS con un alto numero di osservazioni determina un forte aumento della sua dimensionalità, rendendo molto lunga la fase di training e creando modelli non sempre performanti[3];

- E' difficile trovare una metrica di similarità che sia *valida*, ovvero che restituisca un valore alto di similarità per le sole TS con andamenti realmente simili, ed *efficiente*, ovvero che il suo tempo di calcolo risulti accettabile;

- Un dataset potrebbe contenere TS di **lunghezze differenti**, andando a complicare ulteriormente l'analisi per via del non allineamento di tutte le serie;

- Una singola osservazione può contenere più di un singolo valore discreto andando a complicare ulteriormente l'analisi per via della grande quantità di dati da elaborare.

Per risolvere il problema della dimensionalità, un buon approccio è quello di utilizzare tecniche di **feature extraction**, ovvero algoritmi che estraggono feature non note dalle TS, permettendo ai modelli di allenarsi su queste nuove feature piuttosto che sulle singole osservazioni.
Inoltre, è possibile anche applicare delle tecniche di **feature selection**, che selezionano solamente le feature rilevanti, andandone a ridurne di numero.
In maniera concettualmente simile, si potrebbe codificare l'input, ovvero la singola TS, in un formato "compresso", di dimensione ridotta ma che, allo stesso tempo, ne preservi le principali caratteristiche.

Per risolvere il problema della metrica di similarità bisognerebbe usare ana funzione di distanza che tenga conto della natura delle TS e di tutte le loro particolarità.

1.2 Il problema

Il progetto è svolto in sinergia con altri gruppi e ha come obiettivo quello di confrontare approcci differenti per effettuare il **clustering** di TS, in modo

[3]La cosiddetta **Curse of dimensionality**

da determinare la validità delle varie soluzioni.
Per il problema in questione sono stati assegnati vari dataset e ogni gruppo ha ricevuto l'incarico di determinare delle soluzioni valide e di evidenziarne i punti di forza e le debolezze. Ai fini di effettuare un'analisi comparativa adeguata, vengono usate alcune **misure di valutazione di clustering** comuni a tutte le soluzioni.

1.3 Stato dell'arte

L'analisi delle time series è uno dei principali campi di **ricerca** e sul quale si sta investendo maggiormente attualmente. Non sono pochi, infatti, i progetti nati negli ultimi anni che hanno contribuito allo sviluppo in merito.

TSFresh[1] è una libreria per *Python*, compatibile con *Scikit-Learn*, per l'estrazione automatica delle feature delle time series. La libreria permette di estrarre le caratteristiche rilevanti di una TS come la media, il valore minimo, il numero di picchi, la mediana, ecc.
La libreria, tra l'altro, assegna un *valore di rilevanza* alle feature, così da permettere la selezione di quelle più rilevanti. Viene offerto anche un supporto al calcolo parallelo e distribuito.
L'estrazione e selezione delle feature permettono di superare i problemi discussi prima, su tutti quello della lunghezza differente delle serie temporali, rendendo l'analisi molto più precisa ed efficace.
TSFresh funziona solo come estrattore e selettore di feature, e non fornisce alcun modello di classificazione, regressione o clustering.

TSLearn[2] è una libreria per Python scritta in *C++* per l'analisi di time series. A differenza di TSFresh, essa offre, tra l'altro, dei modelli di ML ad hoc per le TS. E' in grado di leggere i dataset, di farne del preprocessing, di effettuare classificazione e regressione con SVM e clustering, usando metriche di distanza ad hoc per le TS, come il già citato DTW.
La libreria è piuttosto giovane (v:0.12.0) e sono pochi gli algoritmi implementati; anche la documentazione risulta poco curata nel dettaglio.

1.4 Soluzione proposta

Come approccio principale per affrontare il problema descritto, si è scelto di fare uso di un **autoencoder**, una particolare rete neurale in grado di codificare l'input, qui le TS, in una forma "compressa", chiamata **vettore latente**, ricavata rispetto agli iperparametri della rete.
Questa rappresentazione codificata dell'input viene usata per effettuare il clustering attraverso k-Means, non solo **reiterando l'esecuzione più volte per ottenere una maggiore precisione**, ma anche verificando la suddivisione **usando un diverso numero di cluster**.
Per ogni esecuzione di k-Means vengono mostrati i risultati tramite le misure di validazione di clustering in comune con gli altri gruppi.

I risultati del clustering con l'autoencoder verranno confrontati con un **clustering basato su metrica DTW** tramite la libreria **TSLearn**.
Successivamente, il confronto verrà fatto con i risultati degli altri gruppi, che hanno fatto uso di clustering con **tecniche di feature extraction and selection** basate su **TSFresh** e altri algoritmi.
L'analisi comparativa ha lo di scopo di valutare le prestazioni sui soli dataset in esame e su determinate categorie di TS, cercando di dare un'idea di quali tecniche possano funzionare meglio e su quali tipologie di serie temporali.

Per realizzare questo lavoro è stato scelto il linguaggio *Python* per via dell'alto supporto fornito dalle sue librerie, quali *Scikit-Learn*[3], *NumPy*[4], *Matplotlib*[5] e *Pandas*[6]. Il codice è stato sviluppato e curato con l'utilizzo di notebook *Jupyter*, così da poter permettere una veloce ed efficace gestione del progetto: per le varie tecniche pensate è stato necessario valutarne la bontà con una prototipazione veloce e semplice che permettesse inoltre di rieseguire determinate porzioni di codice e mostrare graficamente i risultati ottenuti.

1.5 Validazione della soluzione

Per poter valutare le prestazioni dell'autoencoder si adotteranno alcune **misure interne** e **misure esterne**[7] di valutazione di clustering. Esse saranno usate per questi scopi:

- Valutare la **bontà del modello** proposto e il **valore ottimale dei suoi iperparametri**;
- Valutare **l'efficacia dell'algoritmo k-Means**;
- Valutare il **numero ottimale di cluster** per k-Means;
- Valutare la **qualità dei dataset in esame**.

Le misure sono anche dette **indici**.

1.5.1 Misure di validazione interne

Una **misura di validazione interna** si occupa di validare i risultati di un algoritmo di clustering guardando quanto bene ha accomunato (messi in uno stesso cluster) item simili e quanto bene ha separato (messi in cluster diversi) item differenti.
In altre parole, si vuole minimizzare la distanza intra-cluster e massimizzare la distanza inter-cluster[4].

La **compactness** (o cohesion) misura quanto bene sono vicini gli item in ciascun cluster ed è, tipicamente, la somma dei quadrati di tutte le distanze degli item di un cluster dal relativo centroide. E' bene minimizzarla il più possibile.
La **separation** misura quanto bene sono distanziati gli item in diversi cluster ed è, tipicamente, la somma dei quadrati di tutte le distanze tra tutti i centroidi. E' bene massimizzarla il più possibile.
Le misure di interne sono basate sui concetti di compactness e separation.

E' bene ricordare che alcuni algoritmi di clustering già hanno come scopo quello di ottimizzare uno solo dei due aspetti oppure entrambi, quindi potrebbe capitare che alcune misure risultino quasi sempre alte se si applica un determinato algoritmo, altre quasi sempre basse.

Silhouette Coefficient

Il **Silhouette Coefficient di un item** i, o semplcemente la *silhouette*, serve a stabilire quanto bene i è stato assegnato al giusto cluster.

[4]Queste distanze sono calcolate a seconda della metrica di distanza scelta

Sia i un'item e sia d una qualsiasi funzione di distanza. Il Silhouette Coefficient di i, $S(i)$ si calcola nel seguente modo:

$$S(i) = \frac{b_i - a_i}{max(a_i, b_i)} \tag{1.1}$$

dove:

(i)
$$a_i = \frac{1}{|C| - 1} \sum_{i \neq j \in C} d(i, j)$$

è la distanza media dell'item i rispetto agli altri item j nello stesso cluster C.
Più il valore è piccolo, più l'item i è vicino agli altri item dello stesso cluster;

(ii)
$$b_i = \min_{k \neq i} \frac{1}{|C_k|} \sum_{j \in C_k} d(i, j)$$

è la distanza media dell'item i rispetto agli item j nel più vicino cluster[5] diverso da quello che contiene i.

E' compreso tra -1 e 1, con -1 indicante un clustering totalmente errato, 1 indicante un clustering altamente denso e ben separato e 0 indicante clustering sovrapposti (il caso medio).
Si osserva che se $b_i > a_i$, allora $S(i)$ risulterà compreso tra -1 e 0, sintomo di un'assegnazione di i al cluster sbagliato: sarebbe più opportuno assegnarlo al cluster più vicino.
Se $b_i < a_i$, allora $S(i)$ risulterà compreso tra 0 e 1. Più a_i tende a 0, più $S(i)$ tenderà ad 1, sintomo di perfetta assegnazione di i al cluster giusto.
Se $S(i)$ è circa 0 allora l'item non dovrebbe stare in nessuno dei due cluster proposti, e sarebbe quindi opportuno rivalutare il numero di cluster scelti, se non proprio l'algoritmo di clustering stesso.

[5] Anche detto *neighbour cluster*

E' possibile calcolare il Silhouette Coefficient medio di tutto il clustering. Sia D l'insieme di tutti gli item:

$$S = \frac{1}{|D|} \sum_{i \in D} S(i) \tag{1.2}$$

Più S tende ad 1, migliore è l'assegnazione degli item.
La sua elaborazione può richiedere molto tempo se la metrica di distanza è complessa.

Davies-Bouldin Index

Il **Davies-Bouldin Index** (DB) di un clustering misura la media di similarità di ciascun cluster con il suo cluster più simile.

Sia m il numero di cluster, il Davies-Bouldin Index è calcolabile nel seguente modo:

$$DB = \frac{1}{m} \sum_{i=1}^{m} \max_{i \neq j} sim(C_i, C_j) \tag{1.3}$$

dove:

(i)
$$sim(C_i, C_j) = \frac{c_i + c_j}{d(C_i, C_j)}$$

è una misura di *similarità tra due cluster*.
c_i è la compactness del cluster C_i calcolata come distanza media di tutti gli item in C_i rispetto al proprio centroide.
$d(C_i, C_j)$ è la separation tra i cluster C_i e C_j, calcolata come distanza tra i rispettivi centroidi.

Ha 0 come lower bound e, a differenza di altri indici, un buon clustering è rappresentato da un DB tendente a 0. Per minimizzare DB, bisogna rendere i cluster il meno simili possibile, minimizzando le compactness di ciascun cluster (i valori c_i) oppure massimizzando la separation tra cluster ($d(C_i, C_j)$). Cluster molto densi e distanti hanno DB minimo. Con *DBSCAN* si raggiungono quasi sempre score tendenti a 0 poiché l'algoritmo ottimizza la densità dei singoli cluster.

Altre misure interne

Non esistono molte altre misure interne affermate, ma è bene citare il **Dunn Index** che misura quanto bene i cluster siano compatti e separati l'uno dall'altro.

Sia m il numero di cluster, il Dunn Index è calcolabile nel seguente modo:

$$D = \frac{\min_{1 \leq i \leq j \leq m} \delta(C_i, C_j)}{\max_{1 \leq k \leq m}(diam(C_k))} \tag{1.4}$$

dove:

(i) $\delta(C_i, C_j)$ è la separation tra due cluster C_i e C_j, calcolata attraverso una qualsiasi metrica di distanza.
Il numeratore di D considera la più piccola di queste separazioni, escludendo quelle già calcolate;

(ii) $diam(C_k)$ è una funzione che calcola il *diametro* del cluster C_k, ovvero la massima distanza (di qualsiasi tipo) tra due dei suoi item. E' una particolare misura di compactness.

Cluster molto grandi hanno una maggiore probabilità di avere un diametro maggiore, e quindi una maggiore probabilità di aumentare il denominatore di D, andando a diminuire l'indice. Aumentare di molto il numero di cluster andrà ad aumentare D con alta probabilità.
E' un indice rilevante in dataset molto grandi e/o quando è previsto avere un grande numero di cluster.

1.5.2 Misure di validazione esterne

Una **misura di validazione esterna** si occupa di validare i risultati di un algoritmo di clustering rispetto a delle **ground truth**[6], ovvero delle informazioni relative ad un altro tipo di clustering o ad una classificazione degli stessi dati non usata per compiere il clustering in valutazione.
Nelle ground truth, quindi, **ogni item possiede una classe** (o label), da usare come oracolo.

[6] Verità di base

A differenza delle misure interne, che si pongono di misurare cluster compatti e separati, **le misure esterne misurano la *fedeltà* dei cluster alle ground truth**. La fedeltà è spesso calcolata tramite il **pair counting**[7], ovvero contando:

- Il numero di coppie di item clusterizzati insieme se essi sono della stessa classe. Anche detto numero di **True Positive** (TP);
- Il numero di coppie di item clusterizzati insieme se essi sono in classi diverse. Anche detto numero di **False Positive** (FP);
- Il numero di coppie di item NON clusterizzati insieme se essi sono della stessa classe. Anche detto numero di **False Negative** (FN);
- Il numero di coppie di item NON clusterizzati insieme se essi sono in classi diverse. Anche detto numero di **True Negative** (TN);

Chiaramente, come nel task della classificazione, lo scopo generale è quello di massimizzare TP e TN e di minimizzare FP e FN.
Molti degli indici esterni sono, quindi, **basati su conteggi non pesati**, che assegnano un peso uguale a tutte le diverse tipologie di coppie.

Contingency Matrix

La **Contingency Matrix**, più che un vero e proprio indice, è una matrice che riporta la cardinalità delle intersezioni dei cluster con tutte le classi delle ground truth. L'elemento x_{ij} rappresenta il numero di item della classe i presenti nel cluster j.
Quanti più valori tendenti a 0 possiene una colonna, tanto più quel cluster è fedele ad una delle classi. Se il numero di cluster è uguale al numero di classi ($i = j$), è bene che un cluster ne ricopra una e una sola.

[7] Conteggio delle coppie

	A_1	A_2	...	A_m
B_1	x_{11}	x_{12}	...	x_{1m}
B_2	x_{21}	x_{22}	...	x_{2m}
...	...	...	...	...
B_n	x_{n1}	x_{n2}	...	x_{nm}

Tabella 1.1: Contingency Matrix che confronta un labelling di n classi e un clustering di m cluster. x_{ij} è il numero di item della classe i presenti nel cluster j

Non essendo un valore scalare, risulta di difficile interpretazione se il numero di cluster e/o di classi aumenta molto. Allo stesso tempo, è molto utile per dare una prima interpretazione della fedeltà del clustering rispetto alle ground truth.

Purity

La **Purity** è una misura che stabilisce quanto bene un clustering copra le ground truth. Precisamente, misura quanto un cluster contiene elementi di una singola classe.

Formalmente, sia C il clustering analizzato, L l'insieme di classi di riferimento, e sia N l'insieme degli item:

$$P = \frac{1}{|N|} \sum_{c \in C} \max_{l \in L} |c \cap l| \tag{1.5}$$

E' compresa tra 0 e 1, con 0 indicante nessuna copertura, e 1 indicante copertura massima.
Ogni cluster partecipa alla sommatoria con il numero di item della classe assolutamente più diffusa al suo interno. Guardando alla Contingency Matrix, si sommano i valori massimi di ciascuna colonna.

Non è consigliata con classi sbilanciate poiché risulterà facilmente alta per via dell'alta probabilità che un cluster abbia come classe più diffusa quella massima.
Per ridurre questo effetto negativo si potrebbe pensare di considerare il massimo relativo di ciascun cluster invece che del massimo assoluto: ogni cluster partecipa alla sommatoria con il numero di item della classe relativamente

più diffusa al suo interno.
La divisione viene comunque fatta sul numero totale di elementi. Questa misura può essere chiamata **Relative Purity**, anche essa compresa tra 0 e 1 e con identico significato.

Rand index e Adjusted Rand index

Il **Rand index** (RI), o **Accuracy**, è il rapporto tra il numero di coppie correttamente clusterizzate (ottenuto dalla somma di TP e TN) sul numero di coppie non ordinate totali.
Sia n il numero di item, il Rand Index si calcola nel seguente modo:

$$RI = \frac{TP + TN}{C_2^n} \tag{1.6}$$

Il numero di coppie non ordinate può anche essere calcolato sommando TP, FP, FN e TN.
E' compreso tra 0 e 1, dove 0 indica che il clustering e il labelling non concordano su alcun punto, mentre 1 indica che c'è concordanza massima.

Il Rand index presenta un problema grave: valutare un *clustering casuale uniforme*, anche detto **random labelling**, rispetto ad ground truth determina sempre un buon valore, crescente con il numero di cluster creati. Chiaramente questa situazione non è desiderabile, ed è casuata dal fatto che i quattro valori pesano tutti allo stesso modo sul risultato.
Per risolvere questo problema si usa una versione "aggiustata" dell'indice, chiamata **Adjusted Rand Index** (ARI)[8].
L'ARI tiene conto del RI del random labelling e lo usa come riferimento:

$$ARI = \frac{RI - E[RI]}{max(RI) - E[RI]} \tag{1.7}$$

dove:

(i) RI è il Rand index calcolato al solito modo;

(ii) $E[RI]$ è il valore atteso del Rand Index, calcolabile con la formula del valore atteso di una variabile aleatoria ipergeometrica (il Rand Index può essere modellato come ipergeometrica) oppure calcolando il RI di un clustering uniforme casuale;

(iii) $max(RI)$ è il massimo valore che può assumere il Rand Index, calcolato come media del numero di coppie di item di ciascun cluster e di ciascuna classe.

E' compreso tra -1 e 1, dove -1 indica che il clustering è paragonabile ad un clustering casuale rispetto al labelling, mentre 1 indica match perfetto.

Pairwise Precision, Recall, F1-Score e Fowlkes-Mallows index

Precision, Recall ed F1-Score sono misure molto usate nel task di classificazione, e possono essere riproposte anche nel clustering se considerate nella loro versione pairwise, cioè che contano il numero di coppie invece che i singoli item.
La **Precision** di un clustering rispetto a delle ground truth è calcolata nel seguente modo:

$$Pr = \frac{TP}{TP + FP} \tag{1.8}$$

La **Recall** di un clustering rispetto a delle ground truth è calcolata nel seguente modo:

$$Re = \frac{TP}{TP + FN} \tag{1.9}$$

L'**F1-Score** di un clustering rispetto a delle ground truth è la media armonica di Precision e Recall; calcolata nel seguente modo:

$$F_1 = 2\,\frac{Pr * Re}{Pr + Re} \tag{1.10}$$

Il **Fowlkes-Mallows index** (FMI), o **G-Measure**, di un clustering rispetto a delle ground truth è la media geometrica di Precision e Recall; calcolato nel seguente modo:

$$FM = \sqrt{Pr * Re} \tag{1.11}$$

Tutte e quattro le misure sono comprese tra 0 e 1, con 0 indicante totale indipendenza tra clustering e labelling, e con 1 indicante match perfetto.
Il Fowlkes-Mallows è immune al random labelling: in quel caso ritorna uno score tendente a 0.
Nessuna delle quattro misure tiene conto dei TN, cosa che potrebbe essere utile in alcuni casi.

Altre misure esterne

Esistono numerose altre misure esterne, tra le quali:

- **Homogeneity**, misura quanto un cluster contenga item di una singola classe;
- **Completeness**, misura quanto gli item di una classe appartengano allo stesso cluster;
- **V-Measure**, media armonica tra Homogeneity e Completeness. Non è immune al random labelling;
- **Jaccard index**, basata sulla similarità di Jaccard ed è il rapporto tra TP e la somma di TP, FP e FN;
- **Dice index**, come il Jaccard, ma da un peso doppio alle coppie TP;
- **Mutual Information Score**, basati sulle misure della teoria dell'informazione, come l'entropia. Misura l'informazione condivisa tra il clustering e il labelling usando similarità non lineari (logaritmiche).
  Non essendo immune al random labelling, si usano spesso la **Normalized Mutual Information Score** e la **Adjusted Mutual Information Score**.

1.6 Struttura del documento

Nel capitolo 2 verranno presentati i dataset scelti per valutare la soluzione proposta, illustrandone le caratteristiche principali.

Nel capitolo 3 verrà presentata la soluzione proposta, ovvero quella che ricorre all'uso di un autoencoder. Nello stesso capitolo verranno anche mostrati i risultati del modello.

Nel capitolo 4 verranno confrontati i risultati dell'autoencoder con il clustering con DTW di TSLearn e con il clustering degli altri gruppi.

Infine, nel capitolo 5 verranno tratte le conclusioni riguardanti l'autoencoder, i dataset in esame e l'algoritmo di clustering scelto.

Capitolo 2

Dataset in esame

2.1 Introduzione

In questo capitolo, nella sezione **2.2** vengono elencati e descritti in generale i dataset presi in esame, mentre nella sezione **2.3** viene fatto del data profiling per poter ottenere qualche insight sui dati, in particolare nella sottosezione **2.3.1** i dati vengono puliti e preparati all'uso.
Infine, nella sezione **2.4** vengono tratte alcune considerazioni su tutti i dataset.

2.2 Descrizione

Per l'esperimento sono stati selezionati alcuni dataset di time series dal sito `http://www.timeseriesclassification.com/dataset.php` in formato TSV (Tab Separated Value).
Ciascuna TS ha una **label** associata, corrispondente ad una classe di riferimento assegnata dagli autori dei dataset.
Sono stati scelti di dimensioni e caratteristiche differenti l'uno dall'altro:

- **ECG5000**, contenente 5000 elettrocardiogrammi (ECG) ottenuti da pazienti affetti da insufficienza cardiaca. Le cinque classi sono state assegnati per annotazione automatica dagli autori;
- **ECG200**, contenente 200 elettrocardiogrammi ottenuti da pazienti affetti da insufficienza cardiaca. Le due classi rappresentano gli ECG di soggetti sani e di soggetti affetti da infarto al miocardio;

- **ChlorineConcentration**, contenente dati relativi alla presenza di cloro nell'acqua, recuperati usando 166 sensori lungo una rete idraulica per 15 giorni;
- **FordA**, contenente dati relativi al rumore di motori di autovetture ottenuti sotto normali condizioni d'uso;
- **FordB**, contenente dati relativi al rumore di motori di autovetture, alcuni sotto normali condizioni d'uso, altri sotto condizioni di elevato rumore;
- **PhalangesOutlinesCorrect**, contenente dati relativi ad outline di falangi ottenuti usando algoritmi di estrazione di outline da immagini di radiografie delle mani. Questi outline possonoe essere corretti o errati;
- **RefrigerationDevices**, contenente dati relativi al consumo di tre diversi dispositivi di refrigerazione domestici;
- **TwoLeadECG**, contenente elettrocardiogrammi di due tipi diversi di segnale;
- **TwoPatterns**, contenente TS generate artificalmente, ciascuna caratterizzata da una coppia di pattern nel segnale: up-up, up-down, down-up e down-down.

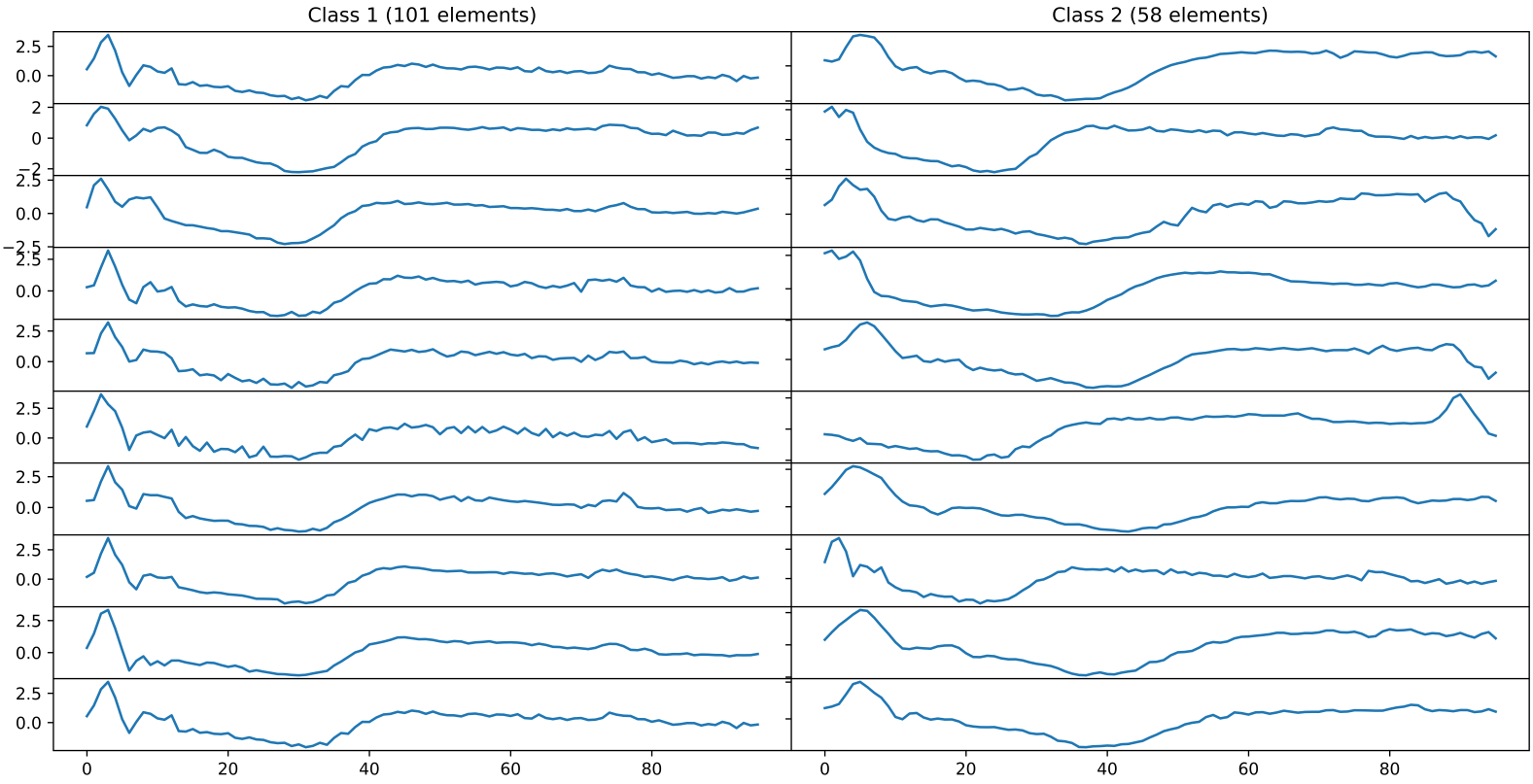


Figura 2.1: Plot di alcune TS classificate in ECG200. Si nota facilmente che, sebbene classificate insieme, le TS della classe 2 non presentano sembrano avere andamenti simili.

Di seguito sono riportate le caratteristiche principali dei dataset:

Name	Train size	Test size	Sequence length	Classes
ECG5000	500	4500	140	5
ECG200	100	100	96	2
Chlorine	476	3840	166	3
FordA	3601	1320	500	2
FordB	3636	810	500	2
Phalanges	1800	858	80	2
Refrigeration	375	375	720	3
TwoLeadECG	23	1139	82	2
TwoPatterns	1000	4000	128	4

Tabella 2.1: Caratteristiche principali dei dataset (i nomi sono stati abbreviati per una migliora leggibilità).

2.3 Data Profiling

Attraverso la libreria Python *Pandas* è stato possibile fare un po' di data profiling, andando ad osservare nel dettaglio la struttura di questi dataset. Sono emerse alcune informazioni interessanti:

- Tipicamente le label sono valori interi non negativi, che partono da 0 o da 1, ma in alcuni dataset sono state usate **label negative**;
- Le classi di alcuni dataset sono **sbilanciate**, ad esempio in ECG5000 ci sono 2919 sample nella classe 1, mentre solo 24 nella classe 5;
- Tutte le **time series sono della stessa lunghezza**;
- **Ciascuna osservazione è un singolo valore scalare**, quindi i dataset hanno profondità pari a 1;
- Non ci sono **valori nulli**, quindi tutte le osservazioni sono definite;
- Sono stati identificati molti valori **outlier**, infatti alcune colonne hanno una distrubuzione di valori molto più ampia rispetto alle altre, probabilmente dovuti ad errori di misurazione dei sensori;
- Le dimensioni di training e test set sono sbilanciate e con una cattiva distribuzione delle TS rispetto alle diverse classi.

Lo sbilanciamento è stato individuato facendo il **raggruppamento rispetto alla colonna 0**, ovvero quella relativa alle label assegnate alle TS, mentre gli outlier sono stati individuati **guardando la media, la deviazione standard e i quartili di ciascuna colonna**, oltre che il plot dei dati.
L'assenza di valori null è stata riscontrata grazie al conteggio dei valori null di Pandas.

Di seguito un riassunto della distribuzione dei valori nelle classi di ciascun dataset (il numero associato ad ogni classe non rispecchia l'effetivo valore delle label):

Name	Class 1	Class 2	Class 3	Class 4	Class 5
ECG5000	2919	1767	96	194	24
ECG200	133	67	/	/	/
ChlorineConc.	1000	1000	2307	/	/
FordA	2394	2527	/	/	/
FordB	2185	2261	/	/	/
Phalanges	960	1698	/	/	/
Refrigeration	250	250	250	/	/
TwoLeadECG	581	581	/	/	/
TwoPatterns	1306	1248	1245	1201	/

Tabella 2.2: Distribuzione delle classi dei dataset

Com'è possibile notare dalla Figura 2.1, alcuni dataset presentano una classificazione che non è basata sull'andamento dei dati nel tempo, bensì su altre caratteristiche dei dati stessi (nel caso di ecg200 se i pazienti fossero sani o infartuati).
Questo implica necessariamente che un clustering effettuato basandosi unicamente sull'andamento della TS non potrà mai coincidere con una classificazione del genere.
Questo aspetto verrà tenuto conto durante le valutazioni dei risultati.

2.3.1 Data cleaning and preparation

In risposta ai problemi individuati durante la fase di data profiling, sono state adottate alcune azioni correttive:

- Tutte **le label sono state modificate** in modo tale da essere degli **interi positivi che partono da 1 e che si incrementano unariamente**, rispecchiando la numerazione presente nella tabella 2.2.
  Questa pulizia è stata fatta direttamente sui file dei dataset;

- Il problema dello sbilanciamento delle classi non poteva essere risolto poiché è un elemento intrinseco dei dataset. Tuttavia, è stato possibile risolvere il problema degli outlier, **"tagliando" tutte le osservazioni che andavano al di sotto del 3-percentile oppure oltre il 97-percentile** e ricondurle a loro.
  Questa pulizia viene fatta ogni volta che viene caricato un dataset in memoria.

- Il problema dello sbilanciamento di training e test set è stato risolto **unendo i due insiemi in memoria e poi dividerli casualmente con la regola dell'80/20**[1] **di default**.
  Questa operazione viene fatta ogni volta che viene caricato un dataset in memoria.

2.4 Considerazioni

Lo **sbilanciamento delle classi potrebbe causare qualche problema nella fase di valutazione esterna del clustering**, poiché diventa difficile per k-Means creare cluster molto piccoli se i dati sono effettivamente molto simili tra loro.

[1] 80% dei dati è di train, mentre il restante 20% è di test.

Capitolo 3

Soluzione proposta

3.1 Introduzione

In questo capitolo le sezioni **3.2** e **3.3** saranno introduttive agli elementi fondamentali che compongono l'architettura del modello proposto.
La sezione **3.4** descrive l'approccio sviluppato e come lo si è utilizzato.
La sezione **3.5**, infine, tratta dei risultati ottenuti.

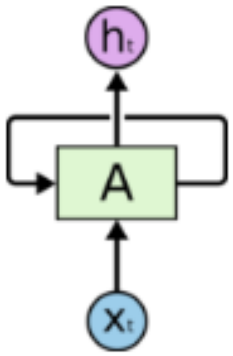
3.2 RNN

Una TS è una **sequenza** di punti, ognuno di questi descrive il valore di una data variabile registrata in momenti diversi, solitamente, ad intervalli regolari.
Da questa definizione è facile intuire come i dati siano tra loro correlati, e non indipendenti, come spesso si assume in diverse tecniche di machine learning. Gli approcci tradizionali quindi peccano da questo punto di vista, e l'analisi risulta difficile anche per il fatto che le serie possono avere lunghezze diverse in base al numero di osservazioni. Per una classica rete neurale, ad esempio, bisognerebbe adattare l'input layer in modo da allinearsi alla lunghezza delle TS e dovrebbe anche tenere conto della correlazione di osservazioni temporalmente vicine.

Soluzioni provate in letteratura prevedono che si utilizzino diverse istanze della stessa rete neurale affinché ciascuna operi su porzioni diverse della stessa serie, per poi unire i risultati, riprendendo un po' l'idea di un ensemble.

Questa soluzione, per quanto semplice, non funziona bene con sequenze di dati, poiché, come detto, i dati sono tra loro correlati nel tempo: ogni valore nel tempo t è legato al valore nel tempo $t-1$. Con questa soluzione le reti si addestrano su sottosequenze indipendenti della stessa TS, ma si perdono le correlazioni esistenti.

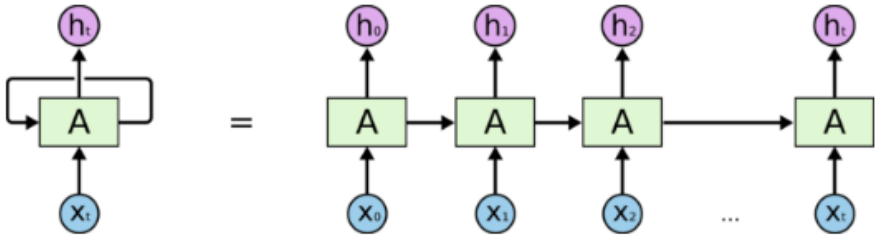
Il concetto, dunque, è quello di non perdere le informazioni del passato. Questo approccio ricorda molto quello umano, che non lascia che i pensieri vengano generati partendo dal nulla, ma che siano determinati a partire dal passato, cioé da quello che si conosce. Una **rete neurale ricorrente** (Recurrent Neural Network, RNN) è fa proprio questo: rispetto una rete neurale consueta non ha solamente archi che confluiscono in un'unica direzione, dall'input all'output, ma ne ha altri che possono "tornare indietro".
In questo modo viene risolto il problema di dover tenere memoria delle sottosequenze analizzate nel passato e di prendere decisioni in funzione di esse. Una RNN usa l'output del **layer ricorrente** come input dello stesso per un certo numero di iterazioni (ripetizioni). Tra una ripetizione e l'altra, il risultato degli input precedenti va a condizionare quelli successivi, così da tenere la memoria della sottosequenza di dati visti in precedenza e conservare le correlazioni.
I pesi che evolvono nel tempo all'interno del layer ricorrente formano il cosiddetto *hidden state*, o stato nascosto. Questa tecnica viene chiamata **parameter sharing**.
La dimensione del layer ricorrente è fissata ed andrà a determinare le dimensioni dell'input e dell'output, e corrisponderà alla dimensione della sottosequenza che si desidera analizzare ad ogni iterazione.



Recurrent Neural Networks have loops.

Figura 3.1: L'architettura base di una RNN

E' molto comune dare una rappresentazione *unrolled* della rete per poterla ricondurre ad una rete neurale classica e comprenderne il funzionamento nel dettaglio.



An unrolled recurrent neural network.

Figura 3.2: La rappresentazione unrolled di una RNN

Chiaramente, è possibile rendere più profonda una RNN, ed esistono diversi modi per farlo:

- Aggiungere dei layer nascosti ricorrenti. Detta soluzione **stacked**;
- Rendere il layer ricorrente una sotto-rete profonda;

- Aggiungere dei layer tra l'input e quello ricorrente;
- Aggiungere dei layer tra il ricorrente e l'output;

3.2.1 LSTM

Come evidenziato da Bengio et al.[9], il modello standard di RNN nella pratica soffre di un problema relativo alle **long-term dependencies**: man mano che la rete compie un alto numero di ripetizioni del layer ricorrente, le caratteristiche apprese dalla prime iterazioni iniziano a diventare meno rilevanti, quasi come se la rete iniziasse a dimenticare cosa ha fatto nel passato non recente. In molti casi è importante mantenere le informazioni che risalgono ad un passato meno recente.

Per risolvere questo problema, Hochreiter et al.[10] hanno presentato la più applicata implementazione di RNN, ovvero le reti **LSTM, Long Short Term Memory**.
Il layer ricorrente di una LSTM segue una precisa struttura, ed è chiamato **cella LSTM**. Mentre nelle RNN standard i layer ricorrenti sono composti da neuroni che applicano una qualsiasi funzione di attivazione, un layer LSTM è invece composto da quattro sotto-layer, che applicano delle funzioni più complesse.

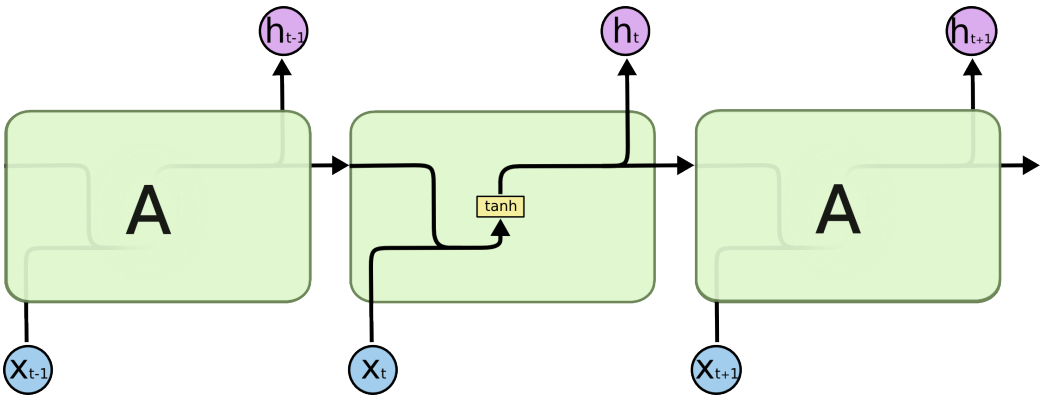


Figura 3.3: RNN unrolled standard. Nell'esempio viene usata la funzione di attivazione tangente iperbolica.

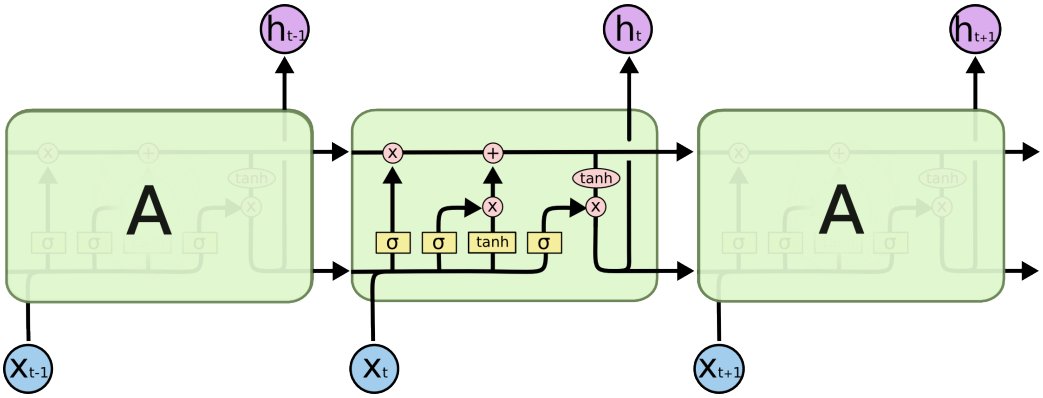


Figura 3.4: LSTM unrolled. I rettangoli gialli sono i quattro sotto-layer, di cui tre di essi applicano la semplice sigmoide, mentre l'altro la tangente iperbolica, mentre le figure rosa sono operazioni su vettori semplici.

La linea superiore della cella trasporta l'output superiore della cella precedente direttamente alla cella successiva, a meno di alcune operazioni lineari. Serve a mandare in avanti la conoscenza del passato senza modificarla troppo.

La cella LSTM non segue necessariamente lo schema suddetto, ed esistono altre varianti, tutte accomunate dall'idea di avere un flusso che preservi le long-term dependencies.
La **hidden size**, cioè dimensione dell'hidden state, determina la dimensione dei quattro sotto-layer, ovvero il numero dei loro neuroni. Il numero totale dei neuroni presenti in una LSTM è allora dato dalla $hiddensize * 4$. Tipicamente, **al crescere della hidden size cresce anche la capacità di memorizzazione della LSTM**.

3.3 Autoencoder

Un **autoencoder** è una tipologia di rete neurale che cerca di apprendere una rappresentazione "compressa" di un input, per poi tentare di ricostruirlo correttamente come output.
Nonostante questo possa sembrare confusionario ed inutile, una rete che impara una versione compressa dei dati si presta in maniera ottimale a risolvere una serie di task che altrimenti non avrebbero soluzioni adeguate. Sostanzialmente un autoencoder è un metodo per ottenere una rappresentazione di

dimensione ridotta di un input di dimensionalità superiore.

Un autoencoder prende i dati dall'input layer e li trasforma in una rappresentazione interna, detta **codifica** o **vettore latente**. Questa rappresentazione è ottima quando l'autoencoder con essa è in grado di ricostruire correttamente l'input, infatti il training di questa particolare rete consiste nel tentare di fornire un vettore latente ottimale per ciascun input.
Lo scopo finale non è quello di ricostruire perfettamente l'input (sarebbe abbastanza inutile) ma quello di estrarre dagli input **questo vettore latente ottimale che conterrà le loro principali caratteristiche**. In altre parole, si pone di estrarre feature rilevanti dall'input.
La compressione e la decompressione vengono effettuate attraverso delle funzioni che imparano automaticamente, senza l'ausilio dell'uomo (**unsupervised**), dai sample di un training set. In quasi tutti i contesti in cui sono usati gli autoencoder, le funzioni di compressione e decompressione sono implementare con reti neurali (nel nostro caso **LSTM**).

Per compiere la ricostruzione è necessario che input e output layer siano della stessa dimensione, tuttavia, al fine di permettere alla rete di estrarre le feature dagli input, il layer nascosto che genera la codifica, detto **coding layer**, dovrà necessariamente essere di dimensione minore degli input e output layer. Si parla di autoencoder **undercomplete**.

Come detto un autoencoder è sempre composto da:

- Un **encoder**, solitamente una rete neurale che riduce l'input nella codifica. Anche detta *rete ricognitiva*;
- Un **decoder**, solitamente una rete neurale che ricostruisce l'input dell'encoder a partire dalla codifica. Anche detta *rete generativa*;

Encoder e decoder sono simmetrici rispetto al coding layer.

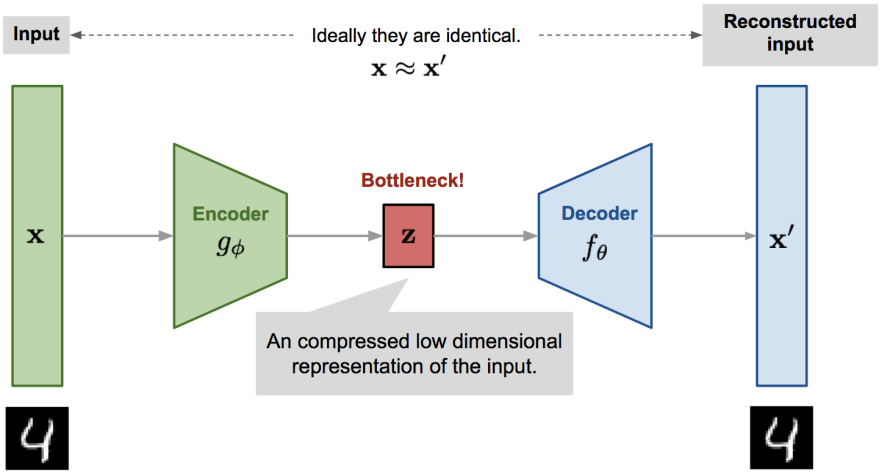


Figura 3.5: Schema di un autoencoder.

Gli autoencoder sono usati principalmente per fare **dimensionality reduction**, dato che la codifica è una rappresentazione ridotta di ogni input, ma anche **generazione di sample**, ovvero generare nuovi dati a partire da codifiche arbitrarie.

Come una qualsiasi altra rete neurale, gli autoencoder possono essere resi più profondi aggiungendo layer nascosti, sia dal lato dell'encoder che dal lato del decoder. In tal caso si parla di **autoencoder stacked**.
In generale, un maggior livello di profondità permette alla rete di comprendere pattern più complessi, tuttavia una profondità troppo alta rende il rischio di overfitting più alto.

3.3.1 Variational Autoencoders

Come i tradizionali autoencoder, i **Variational Autoencoders** (VAEs) cercano di ricostruire gli input utilizzando un encoder e un decoder. Anche in questo caso, l'output dell'encoder è una rappresentazione compressa dell'input, e il decoder impara a ricostruire l'input iniziale prendendo come punto di partenza la rappresentazione compressa generata dall'encoder.
Ciò che rende particolare questa tipologia di autoencoder è che il modello prevede un **approccio probabilistico** per descrivere le osservazioni nello spazio latente.

Anziché costruire un autoencoder il cui output è un singolo valore per ogni attributo del vettore dei fattori latenti, questo viene formulato in maniera tale da descrivere una **distribuzione di probabilità** per ogni attributo latente, tipicamente una distribuzione normale.
Costruendo l'encoder in questo modo otteniamo un range di possibili valori (la distribuzione, appunto), e l'esito è quello di forzare il modello a rappresentare l'input in uno spazio latente continuo e regolare rispetto ai valori iniziali. Per ogni campione della **distribuzione latente** ci aspettiamo che il decoder sia in grado di ricostruire l'input. I valori che sono vicini uno all'altro nello spazio latente in questo modo vengono ricostruiti in maniera simile.

3.4 Modello proposto

3.4.1 Architettura e motivazione

Il modello proposto è composto da quattro componenti fondamentali:

- Il primo componente è l'**encoder** costituito da una rete **Stacked LSTM**. Questa prende in input un vettore che descrive una TS;
- Successivamente, vi è una componente adibita al mappare l'**hidden vector** ottenuto dall'encoder in un vettore dei **fattori latenti**. In questo layer viene effettuato il train dei punti nello spazio latente per avere media nulla e varianza unitaria (il perché è discusso nella prossima sezione relativa al training);
- La componente successiva deve fare il processo inverso al precedente, cioè mappare il vettore latente nel vettore input per il decoder;
- l'ultima componente è il **decoder** che è una rete Stacked LSTM speculare a quella dell'encoder che prende il singolo vettore ricevuto e lo mappa in una sequenza di vettori in output, cioè cerca di ricostruire gli input.

L'idea dietro il modello è quella che, attraverso le LSTM, si possano ottenere le feature più significative delle serie temporali in input.
Se il modello converge vuol dire che il vettore latente è sufficientemente preciso nel ricostruire l'input iniziale. Il vettore latente che si ottiene viene poi utilizzato per effettuare il clustering con k-means.

Il cuore del modello è sicuramente nel'utilizzo delle LSTM: nello studio delle serie temporali, le LSTM sono attualmente considerate il miglior "strumento" per ottenere buoni risultati nella maggior parte dei casi. Il problema delle serie temporali è infatti quello di avere molto rumore e di nascondere influenze nascoste nei dati.
Modelli troppo semplici non si comportano adeguatamente su questo tipo di dati, mentre le LSTM riescono a superare questi problemi, ma anche quelli relativi agli andamenti irregolari, come ad esempio l'avere linearità e non linearità nella stessa serie temporale.

3.4.2 Training e iperparametri

L'obiettivo del training è quello di imparare la funzione di identità in maniera tale che la sequenza dei vettori di input e output sia simile.
Come detto, è stato usato un approccio probabilistico: l'obiettivo è ricavare i parametri della distribuzione normale dell'hidden vector, cioè μ e σ, cercando di minimizzare il **logaritmo di verosimiglianza** dell'input sotto questa distribuzione.
La scelta di imporre la distribuzione normale è stata dettata sia dalla letteratura disponibile, che consiglia questa distribuzione, ma anche del fatto che altrimenti non sarebbe stato possibile utilizzare la tecnica della discesa stocastica del gradiente nella variante **Adam** (ADAptive Moment estimation), anch'esso in letteratura ritenuto il miglior optimizer per questo tipo di problemi.
Durante il training è stato necessario lavorare sui tanti **iperparametri** della rete. I principali, sui quali si è data maggiore attenzione, sono:

- **Number of layers**, indicante il numero di LSTM messe in stack. E' stato scelto 2 per tutti i dataset, per evitare di dare troppa complessità al modello;

- **Hidden size**, indicante la dimensione dell'hidden state, e quindi della singola cella LSTM. E' stato scelto **un valore dipendente dalla tipologia di time series: valori minori se esse avevano andamenti tendenzialmente lineari, altrimenti maggiori.**, così da avere migliori capacità di memorizzazione;

- **Vector size**, indicante la dimensione dei vettori latenti. E' stato scelto **un valore proporzionato alla lunghezza delle time series dei diversi dataset**, così da avere buone capacità di compressione;
- **Learning rate**, indicante la "velocità" con la quale i pesi del modello vengono cambiati in base al Gradient Descent. E' stato scelto **osservando come si comportava la funzione di loss durante il training, diminuendolo in caso di reti molto rapide nella convergenza oppure con grandi fluttuazioni della loss**;
- **Max iterations**, indicante il numero massimo di sessioni di training, ovvero il numero di applicazioni del Gradient Descent per correggere i pesi della rete. E' stato scelto **osservando come si comportava la funzione di loss durante il training, aumentandole in caso di reti con difficoltà nella convergenza**;

Tutti gli iperaparametri sono riportati nel notebook relativo all'autoencoder. Ciascun dataset ha il suo setting più opportuno. Il setting non è stato immediato e si è ricorso alla **random search** per poter trovarne una buona combinazione.

3.5 Clustering con k-Means

Per poter sfruttare la riduzione della dimensionalità compiuta dall'autoencoder, tutto il test set (di default corrispondente al 20% del dataset) viene dato in input all'autoencoder per estrarne i relativi vettori latenti, ovvero la codifica delle TS di test.
Il k-Means viene lanciato cinque volte, seguendo questa regola. Sia n il numero di classi di un dataset:

(i) Se $n \leq 3$, allora si esegue da $2-$Means a $6-$Means;

(ii) Se $n > 2$, allora si esegue da $n-2-$Means ad $n+2-$Means.

L'implementazione utilizzata è quella di Scikit-Learn, la quale viene retirata 100 volte (tramite il parametro n_init), in modo tale che venga scelto il clustering migliore su 100 tentativi. Non viene fornito nessun random state in input.

Il k-Means usa come funzione di distanza quella euclidea, che risulta accettabile poiché i vettori latenti non sono delle TS, quindi la distanza DTW non è pienamente adatta, oltre che dispendiosa in termini di computazione.

Per ciascuna esecuzione di k-Means viene mostrato il plot del clustering sui sample di test. Le metriche di valutazione, sia interne che esterne, sono tutte accumulate in un DataFrame di Pandas, esportato successivamente in CSV.

3.5.1 Valutazione dei risultati

Per alcuni dataset, come ECG5000, ECG200, TwoLeadECG, PhalangesOutlineCorrect, ChlorineConcentration, questa ricerca ha portato miglioramenti in tutti gli score, mentre per gli altri dataset la ricerca non comportava alcun cambiamento sostanziale. Di seguito sono riportati i risultati del clustering effettuato sui vari dataset in esame, con i valori migliori per gli iperapametri.

ECG5000

ECG5000	Internal		External			
Clusters	Silhouette	DB	Purity	Rel. Purity	ARI	FMI
3	0.392	1.169	0.901	0.89	0.748	0.87
4	0.292	1.743	0.867	0.79	0.653	0.807
5	0.162	2.063	0.907	0.638	0.472	0.684
6	0.174	1.826	0.909	0.638	0.477	0.686
7	0.108	2.189	0.911	0.559	0.375	0.607

Tabella 3.1: k-Means sui vettori latenti del test set di ECG5000.

ECG200

ECG200	Internal		External			
Clusters	Silhouette	DB	Purity	Rel. Purity	ARI	FMI
2	0.405	0.942	0.775	0.775	0.284	0.652
3	0.416	1.081	0.775	0.75	0.252	0.575
4	0.396	0.97	0.8	0.8	0.257	0.575
5	0.269	1.237	0.8	0.8	0.244	0.536
6	0.206	1.372	0.8	0.8	0.234	0.523

Tabella 3.2: k-Means sui vettori latenti del test set di ECG200.

ChlorineConcentration

Chlorine	Internal		External			
Clusters	Silhouette	DB	Purity	Rel. Purity	ARI	FMI
2	0.453	0.93	0.535	0.237	-0.001	0.458
3	0.408	0.915	0.535	0.323	-0.006	0.393
4	0.422	0.809	0.535	0.368	-0.004	0.332
5	0.353	0.912	0.535	0.4	0.001	0.29
6	0.342	0.891	0.535	0.409	-0.002	0.271

Tabella 3.3: k-Means sui vettori latenti del test set di ChlorineConcentration.

FordA

FordA	Internal		External			
Clusters	Silhouette	DB	Purity	Rel. Purity	ARI	FMI
2	0.051	4.191	0.528	0.499	-0.001	0.505
3	0.042	3.896	0.528	0.521	-0.001	0.41
4	0.03	4.3	0.533	0.532	0.001	0.356
5	0.032	4.588	0.547	0.534	0.003	0.323
6	0.026	4.971	0.567	0.567	0.006	0.299

Tabella 3.4: k-Means sui vettori latenti del test set di FordA.

FordB

FordB	Internal		External			
Clusters	Silhouette	DB	Purity	Rel. Purity	ARI	FMI
2	0.019	7.002	0.547	0.547	0.008	0.503
3	0.016	5.898	0.536	0.536	0.002	0.419
4	0.014	6.392	0.538	0.538	0.001	0.354
5	0.012	5.977	0.544	0.544	0.002	0.32
6	0.012	5.664	0.535	0.535	-0.0	0.289

Tabella 3.5: k-Means sui vettori latenti del test set di FordB.

PhalangesOutlinesCorrect

Phalanges	Internal		External			
Clusters	Silhouette	DB	Purity	Rel. Purity	ARI	FMI
2	0.334	1.339	0.627	0.608	0.004	0.665
3	0.284	2.699	0.633	0.432	-0.008	0.621
4	0.236	2.506	0.631	0.47	-0.009	0.549
5	0.229	2.25	0.631	0.527	-0.005	0.494
6	0.216	3.026	0.631	0.534	-0.006	0.438

Tabella 3.6: k-Means sui vettori latenti del test set di PhalangesOutlinesCorrect.

RefrigerationDevices

Refrigeration	Internal		External			
Clusters	Silhouette	DB	Purity	Rel. Purity	ARI	FMI
2	-0.001	8.375	0.367	0.367	-0.007	0.444
3	-0.008	8.284	0.43	0.391	0.005	0.365
4	-0.022	7.338	0.445	0.422	0.01	0.323
5	-0.021	7.029	0.438	0.43	0.004	0.276
6	-0.025	6.703	0.445	0.422	0.004	0.266

Tabella 3.7: k-Means sui vettori latenti del test set di RefrigerationDevices.

TwoLeadECG

TwoLeadECG	Internal		External			
Clusters	Silhouette	DB	Purity	Rel. Purity	ARI	FMI
2	0.455	0.855	0.58	0.58	0.021	0.509
3	0.367	0.974	0.629	0.629	0.043	0.434
4	0.299	1.094	0.621	0.621	0.03	0.375
5	0.235	1.276	0.629	0.629	0.02	0.336
6	0.193	1.514	0.634	0.634	0.02	0.305

Tabella 3.8: k-Means sui vettori latenti del test set di TwoLeadECG.

TwoPatterns

TwoPatterns	Internal		External			
Clusters	Silhouette	DB	Purity	Rel. Purity	ARI	FMI
2	0.029	5.541	0.295	0.283	0.01	0.377
3	0.015	6.673	0.293	0.275	0.004	0.328
4	0.02	5.608	0.309	0.308	0.008	0.286
5	0.024	5.114	0.303	0.303	0.006	0.256
6	0.026	4.852	0.312	0.311	0.011	0.229

Tabella 3.9: k-Means sui vettori latenti del test set di TwoPatterns.

Da questi risultati è possibile fare una serie di osservazioni sulle misure:

1. Le **misure interne sono molto influenzate dal numero di cluster**, andando a peggiorare man mano che il numero cresce.
   Questo, chiaramente, dipende molto dalla disposizione dei dati nello spazio e anche dal tipo di algoritmo di clustering: k-Means, infatti, non ottimizza la densità dei cluster, a differenza di altri come DBSCAN;

2. Le **misure esterne sono molto suscettibili con piccoli dataset**, infatti, nel caso di ECG200, si hanno ARI ed FMI abbastanza bassi.
   Il motivo risiede nel fatto che le metriche esterne sono basate su conteggi non pesati;

3. Le misure di **purity e relative purity sono abbastanza indipendenti dal numero di cluster**.
   Questo potrebbe significare che sono più influenzare dalla natura del dataset e dalle caratteristiche che l'autoencoder è stato in grado di estrarre, piuttosto che dal numero di cluster.

Inoltre, è stato possibile fare le seguenti osservazioni sul modello e sui dataset:

1. Nei **dataset degli elettrocardiogrammi le misure esterne sono tendenzialmente alte**. Questo significa che l'autoencoder è riuscito

a estrarre quelle caratteristiche che gli autori dei dataset hanno usato per fare il labelling degli elettrocardiogrammi.
Questo non accade per gli altri dataset, o almeno non con gli stessi score;

2. **Le caratteristiche delle TS molto "rumorose"[1] non riescono ad essere estratte bene dall'autoencoder**, e questo si rispecchia nelle misure interne basse e nelle misure esterne che ricordano quelle del random labelling. Questo accade per FordA, FordB, RefrigerationDevices e TwoPatterns;

[1] hanno dei pattern che continuamente alternano valori alti e valori bassi

Capitolo 4

Analisi comparativa

Di seguito, il lavoro di features extraction dell'autoencoder verrà messo alla prova, confrontandolo con due approcci differenti. Nella sezione 4.1 verrà descritto il Dynamic Time Warping e come viene utilizzato in ambito ML. Nella sezione 4.2 verrà, poi, comparato il risultato del clustering effettuato usando le features estratte dall'autoencoder con il clustering effettuato usando Dynamic Time Warping. Infine, i risultati dell'autoencoder verranno comparati, nella sezione 4.3, con quelli ottenuti estraendo e selezionando fetures con altre tecniche (es. *TSFresh*).

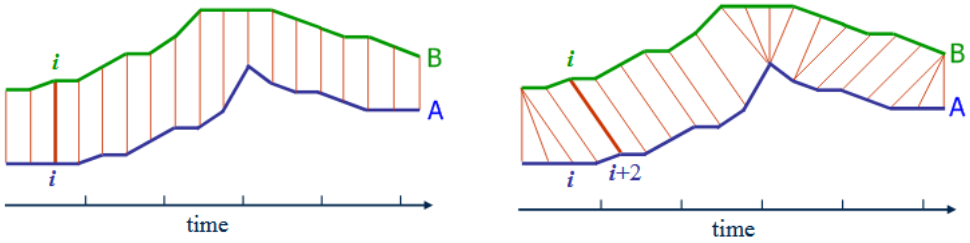
4.1 DTW: Dynamic Time Warping

Il **Dynamic Time Warping** (DTW)[11]è un algoritmo per misurare la similarità tra due TS, calcolandone il match ottimale.

A differenza delle classiche funzioni di distanza, come quella Euclidea o la Manhattan, essa effettua un confronto di tipo **elastico** (non-lineare) tra i punti delle due TS a confronto: esse non sono confrontate 1-a-1, matchando l'i-esimo punto della TS A con l'i-esimo punto della TS B, bensì 1-a-n, matchando un punto della TS A con uno o più punti della TS B.
Il confronto tra due punti matchati avviene sempre con una metrica classica, come quella Euclidea. Tutte le singole distanze tra i punti sono conservate in una **matrice di distanza**.

Siano A e B due TS a confronto, vengono imposti i seguenti vincoli nel calcolo del DTW:

- Un punto di una TS A può essere matchato con uno o più punti della TS B;
- Il match tra due punti può avviene se e solo se le due TS hanno la stessa monotonia nei rispettivi punti;
- Il primo punto della TS A è matchato con il primo della TS B;
- L'ultimo punto della TS A è matchato con l'ultimo della TS B;



(a) Distanza Euclidea (lineare) (b) DTW (elastica)

Figura 4.1: Confronto tra distanza Euclidea e DTW

In altre parole, il DTW cerca di "allineare" al meglio le due TS, facendo in modo che entrambe proseguano con lo stesso andamento.

La similarità del DTW produce buoni risultati per le TS che rappresentano uno stesso evento ma che sono di lunghezza differente, ad esempio quando entra in gioco uno sfasamento temporale.
Un individuo che pronuncia una stessa frase con lo stesso tono ma con velocità differente produrrebbe due TS molto differenti. Se esse venissero confrontate linearmente, non si potrebbe riconoscere che, difatti, l'individuo che ha prodotto tali frequenze è lo stesso. Con un confronto tramite DTW ciò viene ovviato.

L'algoritmo compie una ricerca nella matrice di distanza, di spazio $O(mn)$,

dove m ed n sono le lunghezze delle due TS confrontate. Dunque, al caso pessimo e con TS molto grandi (es. centinaia di punti), il calcolo del DTW richiede un tempo troppo lungo per fornire risultati utili. In letteratura, perciò, sono state definite alcune implementazioni veloci, come *PrunedDTW*, *SparseDTW* e *FastDTW*, che tentano di accelerarne il calcolo. Ciò nonostante, la tecnica resta complessa in termini di tempo e di spazio, sopratutto in caso di dataset ad alta dimensionalità.

4.2 Confronto con k-Means di TSLearn

Il clustering mediante l'impiego di DWT è stato realizzato sfruttando la libreria TSLearn. Essa, infatti, come evidenziato dal nome stesso, consiste in un set di funzionalità di Machine Learning focalizzato sulle Time Series. La libreria non solo offre metodi per il calcolo delle metriche proprie delle TS (*DTW*,*Soft-DTW*, ecc.), ma anche veri e propri algoritmi di Machine Learning, in una variante modificata *ad hoc* per le TS, sia di tipo supervised che unsupervised (es. *SVM*, *TimeSeriesKMeans*).

Proprio uno di questi algoritmi, *TimeSeriesKMeans*, è stato impiegato per il nostro esperimento. Esso consiste in una versione modificata del noto algoritmo di clustering, abile nel calcolo dei cluster usando non solo la metrica euclidea, ma anche il DTW e il Soft-DTW.

Di seguito sono riportati i risultati del clustering dei dataset in esame, a cui vanno anteposte alcune premesse:

- **Il Silhouette Coefficient non è presente in alcun dataset**, data l'impraticabilità di calcolo del DTW su dataset di grandi dimensioni;
- **Non viene riportata la Relative Purity**, in quanto introdotta successivamente al calcolo di questi risultati. Una sua valutazione su questi risultati avrebbe richiesto un computo troppo oneroso;
- Per avere un riscontro diretto con le label presenti nei dataset, **il numero di cluster calcolati corrisponde al numero di classi presenti nel dataset**;

			Internal	External		
Dataset	Clusters	Samples	DB	Purity	ARI	FMI
ECG5000	5	500	1.8	0.924	0.434	0.649
ECG200	2	100	1.832	0.69	0.116	0.596
Chlorine	3	467	0.981	0.561	0	0.375
FordA	2	100	2.384	0.739	0.192	0.579
FordB	2	100	2.559	0.522	-0.041	0.481
Phalanges	2	1800	1.187	0.651	-0.001	0.523
Refrigeration	3	120	2.157	0.522	-0.043	0.468
TwoLeadECG	2	23	2.248	0.565	-0.015	0.548
TwoPatterns	4	1000	12.020	0.974	0.932	0.950

Tabella 4.1: k-Means con DTW sui dataset in esame con relative misure.

Da questi risultati è possibile fare una serie di osservazioni:

1. In *ECG5000*, **l'autoencoder e TSLearn con 5-Means hanno tutti gli score molto simili**. L'autoencoder è riuscito ad estrarre bene le caratteristiche degli elettrocardiogrammi, come intuito dall'analisi precedente;

2. In *ECG200*, **l'autoencoder con 2-Means ha tutti gli score leggermente migliori rispetto a TSLearn**. Le motivazioni sono le stesse di ECG5000;

3. In *ChlorineConcentration*, **l'autoencoder e TSLearn su 3-Means hanno tutti gli score molto simili**, se non per una leggera differenza per FMI. L'autoencoder è riuscito ad estrarre bene le caratteristiche di questo tipo di TS, cosa non intuita dall'analisi precedente;

4. In *FordA*, **l'autoencoder con 2-Means ha tutti gli score ben peggiori rispetto a TSLearn**, ad eccezione del FMI, in entrambi i casi non ottimale. L'autoencoder, dunque, non è riuscito ad estrarre bene le caratteristiche di queste TS "rumorose", come intuito dall'analisi precedente;

5. In *FordB*, **l'autoencoder con 2-Means ha il DB ben peggiore di quelli di TSLearn**, mentre gli score esterni risultano simili. Similmente a FordA, l'autoencoder non è riuscito ad estrarre bene le caratteristiche di queste TS, come intuito dall'analisi precedente;

6. In *PhalangesOutlineCorrect*, **l'autoencoder con 2-Means ha qualche score leggermente peggiore rispetto a TSLearn**. L'autoencoder è riuscito ad estrarre abbastanza bene le caratteristiche di questo tipo di TS, magari c'è bisogno di alcune modifiche agli iperparametri per poter migliorarne la qualità;

7. In *RefrigerationDevices*, **l'autoencoder con 3-Means ha tutti gli score ben peggiori rispetto a TSLearn**, ad eccezione del FMI, che in entrambi i casi è comunque non ottimale. Quasi identicamente a FordA, l'autoencoder non è riuscito ad estrarre bene le caratteristiche di queste TS "rumorose", come intuito dall'analisi precedente;

8. In *TwoLeadECG*, **l'autoencoder e TSLearn con 2-Means hanno tutti gli score molto simili**, ad eccezione del DB, dove l'autoencoder è migliore. Questa analisi non è pienamente affidabile perché il test set fornito è troppo piccolo, quindi gli score sono soggetti a molte variazioni in caso di ulteriori esecuzioni.

9. In *TwoPatterns*, **l'autoencoder con 4-Means ha tutti gli score ben peggiori rispetto a TSLearn**, ad eccezione del DB, che resta comunque non ottimale.
   Similmente a FordA, l'autoencoder non è riuscito ad estrarre bene le caratteristiche di queste TS "rumorose", come intuito dall'analisi precedente;

4.3 Confronto con altre tecniche di feature extraction and selection

Sono stati reperiti alcuni risultati di clustering sui dataset in esame attuando però altre tecniche di feature extraction and selection. Il clustering è stato fatto con k-Means con k pari al numero di classi del dataset e su un numero differente di feature estratte in modo diverso. In particolare su:

- **Tutte le feature estratte** da *TSFresh*;
- **Feature rilevanti** selezionate da *TSFresh*;
- Feature rilevanti selezionate dall'algoritmo **NDFS**;

- Feature rilevanti selezionate dall'algoritmo **Lap Score**.

I seguenti risultati riportano solo gli score silhouette, Davies-Bouldin e purity, quindi verranno usati solamente questi per il confronto.

ECG5000

K-means sulle feature tutte le feature estratte da TSFresh					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
673	-0.4207	2.5094	3145.4262	0.4306	0.898

K-means sulle feature rilevanti estratte da TSFresh					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
544	-0.1782	2.0196	3197.5343	0.428	0.8978

K-means sulle feature selezionate con NDFS					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
7	-0.1011	1.6834	10.1081	0.4699	0.8984
9	-0.4456	1.7668	14.5391	0.4072	0.8738
11	-0.0941	1.9589	16.8514	0.4353	0.8882
13	-0.2806	1.966	16.9081	0.3013	0.7813
15	-0.155	1.8058	23.9307	0.4536	0.89

K-means sulle feature selezionate con Lap Score					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
7	0.043	8.9808	3141.6717	0.0345	0.6062
9	-0.1523	11.1203	3142.766	0.0369	0.5929
11	-0.0417	6.9041	3141.0401	0.0435	0.6047
13	-0.2848	9.2098	3137.707	0.0428	0.6024
15	-0.3364	17.5129	3138.4997	0.0538	0.614

Figura 4.2: ECG5000 con tecniche di feature extraction and selection.

L'autoencoder ha migliore silhouette rispetto a tutti gli algoritmi. DB e Purity sono simili per autoencoder, TSFresh ed NDFS. Lap Score ha gli score peggiori.

ECG200

K-means sulle feature tutte le feature estratte da TSFresh					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
585	0.1022	2.8403	11.659	0.1553	0.7

K-means sulle feature rilevanti estratte da TSFresh					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
103	0.4462	1.0026	64.5152	0.3115	0.83

K-means sulle feature selezionate con NDFS					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
7	0.2239	2.0629	19.8874	0.0506	0.64
9	0.2428	1.5441	34.9789	0.1492	0.68
11	0.1861	2.1369	19.1039	0.1071	0.64
13	0.1787	2.2118	17.1624	0.123	0.65
15	0.1895	2.0265	20.9179	0.0923	0.64

K-means sulle feature selezionate con Lap Score					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
7	0.7557	0.3393	468.2109	0.0899	0.71
9	0.6519	0.6016	173.0043	0.104	0.72
11	0.6511	0.6026	172.7486	0.104	0.72
13	0.5917	0.8408	109.7737	0.1344	0.74
15	0.6004	0.7827	117.8776	0.1511	0.75

Figura 4.3: ECG200 con tecniche di feature extraction and selection.

L'autoencoder ha migliore DB rispetto a tutti gli algoritmi, e silhouette migliore rispetto a TSFresh e NDFS, ma peggiore rispetto a Lap Score. La purity è simile per autoencoder, TSFresh e Lap Score.

ChlorineConcentration

K-means sulle feature tutte le feature estratte da TSFresh					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
725	0.9963	0.8476	3793.0383	0.0001	0.5326

K-means sulle feature rilevanti estratte da TSFresh					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
13	0.5788	0.6079	631.5483	0.0032	0.5326

K-means sulle feature selezionate con NDFS					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
7	0.9942	0.9505	8087.1893	0.0004	0.5342
9	0.9915	0.8276	11342.2858	0.0001	0.5326
11	0.9923	0.6579	14772.1537	0.0002	0.5326
13	0.9927	1.3332	6688.5934	0.0009	0.5328
15	0.9917	1.1488	8540.9293	0.0001	0.5326

K-means sulle feature selezionate con Lap Score					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
7	0.9969	0.841	3838.2612	0.0001	0.5326
9	0.9969	0.841	3838.2592	0.0001	0.5326
11	0.9968	0.841	3838.2576	0.0001	0.5326
13	0.9968	0.841	3838.2568	0.0001	0.5326
15	0.9968	0.841	3838.2549	0.0001	0.5326

Figura 4.4: ChlorineConcentration con tecniche di feature extraction and selection.

L'autoencoder ha purity simile a tutti e quattro gli algoritmi, DB peggiore solo rispetto a TSFresh per feature rilevanti, e silhouette peggiore in tutti i casi, specialmente rispetto ad NDFS e Lap Score, nei quali è quasi massima.

FordA

K-means sulle feature tutte le feature estratte da TSFresh					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
794	0.9837	0.3569	3529.2094	0.0	0.5159

K-means sulle feature rilevanti estratte da TSFresh					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
173	0.9830	0.478	1455.4045	0.0006	0.5159

K-means sulle feature selezionate con NDFS					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
7	0.9801	0.1917	511.6239	0	0.5159
9	0.98	0.2034	510.1606	0	0.5159
11	0.9801	0.2599	511.4536	0	0.5159
13	0.2597	0.2597	511.2199	0	0.5159
15	0.9801	0.2657	509.3579	0	0.5159

K-means sulle feature selezionate con Lap Score					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
7	0.9844	0.3543	3541.5323	0	0.5159
9	0.9843	0.3543	3541.5305	0	0.5159
11	0.9843	0.3543	3541.5267	0	0.5159
13	0.9843	0.3543	3541.5253	0	0.5159
15	0.9843	0.3543	3541.522	0	0.5159

Figura 4.5: FordA con tecniche di feature extraction and selection.

L'autoencoder ha silhouette e DB di gran lunga peggiori rispetto a quattro gli algoritmi, mentre la purity è simile in tutti i casi. DB è quasi ottimale con NDFS.

FordB

K-means sulle feature tutte le feature estratte da TSFresh					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
794	0.9917	0.2071	7090.6613	0.0006	0.5049

K-means sulle feature rilevanti estratte da TSFresh					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
45	0.9886	0.3165	3506.3681	0.0017	0.5049

K-means sulle feature selezionate con NDFS					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
7	0.9902	0.2912	2448.9173	0.0032	0.5049
9	0.989	0.3332	1826.6212	0.0032	0.5049
11	0.9889	0.3349	1814.5178	0.0032	0.5049
13	0.9889	0.3374	1800.6041	0.0032	0.5049
15	0.9888	0.3404	1786.002	0.0032	0.5049

K-means sulle feature selezionate con Lap Score					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
7	0.9923	0.1995	7154.3425	0.0006	0.5049
9	0.9923	0.1995	7154.3208	0.0006	0.5049
11	0.9923	0.1995	7154.3087	0.0006	0.5049
13	0.9923	0.1995	7154.2856	0.0006	0.5049
15	0.9923	0.1995	7154.28	0.0006	0.5049

Figura 4.6: FordB con tecniche di feature extraction and selection.

L'autoencoder ha silhouette e DB di gran lunga peggiori rispetto a quattro gli algoritmi, mentre la purity è simile in tutti i casi. DB è quasi ottimale con Lap Score. E' una situazione simile a quella di FordA.

PhalangesOutlineCorrect

K-means sulle feature tutte le feature estratte da TSFresh					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
//	//	//	//	//	//

K-means sulle feature rilevanti estratte da TSFresh					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
187	0.3561	1.9304	397.4104	0.0673	0.634

K-means sulle feature selezionate con NDFS					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
7	0.1799	1.7353	204.3345	0	0.6131
9	0.1874	1.6204	235.7244	0.002	0.6131
11	0.27	1.3767	306.4012	0.002	0.6131
13	0.4511	0.8107	1201.0924	0.0164	0.6131
15	0.4279	0.8417	1129.7783	0.0191	0.6131

K-means sulle feature selezionate con Lap Score					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
7	0.7914	0.2561	6236.8238	0.0339	0.6235
9	0.5346	0.6643	1114.0528	0.0196	0.6142
11	0.5346	0.6643	1114.0171	0.0196	0.6142
13	0.5346	0.6643	1113.9761	0.0196	0.6142
15	0.4811	0.7718	718.9615	0.0415	0.6399

Figura 4.7: PhalangesOutlineCorrect con tecniche di feature extraction and selection.

L'autoencoder ha purity simile a tutti e quattro gli algoritmi, mentre ha silhouette e DB variabili: migliori rispetto a TSFresh e una parte di NDFS, peggiori rispetto a Lap Score e un'altra parte di NDFS.

RefrigerationDevices

K-means sulle feature tutte le feature estratte da TSFresh					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
789	0.1116	2.1706	59.0132	0.0696	0.464

K-means sulle feature rilevanti estratte da TSFresh					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
124	0.1611	2.2021	73.2554	0.0875	0.4587

K-means sulle feature selezionate con NDFS					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
7	0.4452	1.6277	76.8477	0.0542	0.424
9	0.3894	1.7769	66.4993	0.0173	0.3813
11	0.3167	1.914	59.1698	0.0426	0.432
13	0.2066	1.657	41.13	0.0243	0.392
15	0.2678	2.1764	50.3536	0.0313	0.3947

K-means sulle feature selezionate con Lap Score					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
7	0.3183	1.011	201.7324	0.006	0.3707
9	0.3033	1.056	177.5988	0.0053	0.368
11	0.2917	1.0876	165.5028	0.0068	0.3733
13	0.2798	1.1241	153.4737	0.0068	0.3733
15	0.2047	1.4342	99.206	0.0062	0.3707

Figura 4.8: RefrigerationDevices con tecniche di feature extraction and selection.

L'autoencoder ha purity simile a tutti e quattro gli algoritmi, a meno di leggere differenza, mentre ha silhouette e DB ben peggiori rispetto a tutti gli algoritmi. In particolare, la silhouette migliora con NDFS e Lap Score.

TwoLeadECG

K-means sulle feature tutte le feature estratte da TSFresh					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
557	-0.0345	2.0194	2.7019	0.0	0.5004

K-means sulle feature rilevanti estratte da TSFresh					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
13	0.0334	1.9892	2.5878	0.6421	0.9315

K-means sulle feature selezionate con NDFS					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
7	0.3173	1.331	538.3531	0.0001	0.5057
9	0.2465	1.627	366.4782	0.0002	0.5083
11	0.2055	1.8632	279.4128	0.0002	0.5092
13	0.2144	1.9617	212.8351	0.0026	0.5259
15	0.1939	2.1265	182.7193	0.003	0.5277

K-means sulle feature selezionate con Lap Score					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
7	0.5282	0.7971	285.8217	0.0042	0.5373
9	0.522	0.8277	190.9035	0.0042	0.5373
11	0.4015	1.0992	153.5861	0.0036	0.5347
13	0.397	1.1206	124.0816	0.0036	0.5347
15	0.3931	1.1356	108.8131	0.0034	0.5338

Figura 4.9: TwoLeadECG con tecniche di feature extraction and selection.

L'autoencoder ha silhouette e DB mediamente migliori di tutti e quattro gli algoritmi, infatti ci sono alcune esecuzioni di Lap Score dove invece si ha una situazione ribaltata. La purity è simile in quasi tutti i casi.

TwoPatterns

K-means sulle feature tutte le feature estratte da TSFresh					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
649	0.0314	5.8752	14048.3675	0.001	0.2608

K-means sulle feature rilevanti estratte da TSFresh					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
134	0.1892	1.99	889.8677	0.0073	0.2912

K-means sulle feature selezionate con NDFS					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
7	0.1555	2.3137	4.1802	0.0078	0.2975
9	0.1208	1.4675	66760.3453	0.0081	0.2978
11	0.1349	2.02	5.25	0.0007	0.267
13	0.1105	2.1147	7.2608	0.0014	0.2702
15	0.1112	2.4126	7.8197	0.011	0.3015

K-means sulle feature selezionate con Lap Score					
Numero Features	Silhouette	Indice Davies-Bouldin	Indice Calinski-Harabasz	NMI	Purity
7	0.2058	1.3041	16675.4159	0.001	0.2635
9	0.2039	1.3025	16675.1319	0.0013	0.2652
11	0.1393	1.5041	16623.4347	0.0007	0.2612
13	0.1324	1.5785	16618.5481	0.0015	0.2678
15	0.1279	1.6976	16571.6396	0.0191	0.3073

Figura 4.10: TwoPatterns con tecniche di feature extraction and selection.

L'autoencoder ha silhouette e DB di gran lunga peggiori rispetto ai quattro algoritmi, tuttavia la silhouette dei quattro algoritmi è comunque molto bassa. La purity è molto simile in tutti casi, sebbene anche essa risulti molto bassa.

Capitolo 5

Conclusioni

Con questo esperimento si è voluto valutare diverse tecniche di clustering su Time Series, andando a verificare come tecniche di feature extraction and selection potessero influenzare (sia in positivo che in negativo) i risultati dei clustering. A tal fine si sono confrontate due diverse tecniche di feature extraction/selection (TSFresh ed autoencoder) e si sono confrontati con un clustering applicato su time series pure, mediante la dovuta metrica di distanza (k-Means con DTW di TSLearn). I risultati ottenuti ci permettono di formulare varie considerazioni.

Innanzitutto, **l'autoencoder si è mostrato particolarmenre efficace ad estrarre le principali caratteristiche dai dataset relativi agli elettrocardiogrammi**, ovvero ECG5000, ECG200 e TwoLeadECG. Per i dataset "rumorosi", quali FordA, FordB, RefrigerationDevices e TwoPatterns, i risultati sono stati abbastanza scadenti: allo stato attuale, il nostro modello non è stato in grado di cogliere le caratteristiche di queste TS. Gli altri dataset, ChlorineConcetration e PhalangesOutlineCorrect, hanno dato risultati buoni, ma migliorabili con alcune modifiche agli iperaparametri del modello.

Tutte le tecniche hanno presentato **difficoltà nell'ottenere score interni alti per alcuni dataset**. Questo può essere causato dal fatto che alcuni dataset potrebbero essere densi, quindi molte TS risultano "vicine" nello spazio, andando a danneggiare compattezza e separazione dei cluster.
Inoltre, k-Means è un algoritmo che minimizza la varianza intra-cluster piuttosto che massimizzare la densità dei cluster, quindi gli score come Davies-Bouldin non sono sempre ottimali, essendo basati proprio sulla densità.

Tutte le tecniche hanno riscontrato **dei limiti nella massimizzazione di score esterni per alcuni dataset**. Questo può essere causato dal fatto che alcuni dataset presentano classi fortemente sbilanciate, oppure classi con TS molto diverse tra loro, cosa che è stata notata durante la fase di data profiling.

Si è verificato, inoltre, che per i vari clustering calcolati sui vettori latenti estratti dall'autoencoder, si ottenevano **misure di qualità interne migliori** per un**numero di cluster basso**, come due o tre, mentre per valori di cluster alti, come cinque o sei, esse peggioravano. Tale andamento, seppur in maniera più ridotta, si è riscontrato anche per le misure esterne.

In generale, gli score sono stati **utili per confrontare tra loro le diverse tecniche**, accomunate dall'algoritmo k-Means, **ma non sono significativi al fine di valutare la bontà in assoluto di una tecnica di clustering**.

Bibliografia

[1] Maximilian Christ et al. "Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (tsfresh – A Python package)". In: *Neurocomputing* 307 (2018), pp. 72 –77. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2018.03.067. URL: http://www.sciencedirect.com/science/article/pii/S0925231218304843.

[2] Romain Tavenard. *tslearn: A machine learning toolkit dedicated to time-series data*. https://github.com/rtavenar/tslearn. 2017.

[3] Lars Buitinck et al. *API design for machine learning software: experiences from the scikit-learn project*. 2013, pp. 108–122.

[4] Travis Oliphant. *NumPy: A guide to NumPy*. USA: Trelgol Publishing. [Online; accessed ¡today¿]. 2006–. URL: http://www.numpy.org/.

[5] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[6] Wes McKinney. *Data Structures for Statistical Computing in Python*. A cura di Stéfan van der Walt e Jarrod Millman. 2010, pp. 51 –56.

[7] *Clustering - Scikit Learn*. https://scikit-learn.org/stable/modules/clustering.html. Accessed: 2019-07-23.

[8] Ka Yee Yeung e Walter Ruzzo. "Details of the Adjusted Rand index and Clustering algorithms Supplement to the paper "An empirical study on Principal Component Analysis for clustering gene expression data" (to appear in Bioinformatics)". In: *Science* 17 (gen. 2001).

[9] Y. Bengio, P. Simard e P. Frasconi. "Learning long-term dependencies with gradient descent is difficult". In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 157–166. ISSN: 1045-9227. DOI: 10.1109/72.279181.

[10] Sepp Hochreiter e Jürgen Schmidhuber. “Long Short-term Memory”. In: *Neural computation* 9 (dic. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[11] H. Sakoe e S. Chiba. “Dynamic programming algorithm optimization for spoken word recognition”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 26.1 (1978), pp. 43–49. ISSN: 0096-3518. DOI: 10.1109/TASSP.1978.1163055.