



UNIVERSITÀ DI PISA

**Predizione di anomalie nel traffico
urbano attraverso la geolocalizzazione
dei veicoli ed il machine learning**

Candidato:

Leonardo Poggiani

Relatori:

Prof. Mario G. C. A. Cimino

Prof. Gigliola Vaglini

Ing. Antonio Luca Alfeo

Dipartimento di Ingegneria dell'Informazione

Corso di laurea in Ingegneria Informatica

Anno accademico 2019/2020

Ringraziamenti

Vorrei ringraziare per primi i miei relatori, il professor Cimino e l'ingegner Alfeo. Entrambi sono stati disponibili e celeri nelle risposte, riuscendo a stimolare il mio interesse verso una materia nella quale non mi ero mai cimentato prima.

Vorrei ringraziare anche i miei amici, che hanno reso questo percorso più leggero e la mia famiglia, che mi ha sempre sostenuto credendo in me anche nei momenti più difficili.

Ma soprattutto vorrei ringraziare Francesca, mia fidanzata e spalla di appoggio, che non mi ha mai lasciato solo ed è sempre riuscita a farmi andare avanti anche quando non credevo di farcela.

Grazie a tutti.

Abstract

In questa tesi si è deciso di affrontare un tema molto interessante per lo sviluppo delle moderne *smart cities*, quello dell'individuazione delle anomalie nella mobilità urbana.

Infatti una città moderna deve essere in grado di gestire e controllare i flussi di traffico dovuti al crescente addensamento delle attività economiche nelle aree urbane, integrando il sistema di trasporti e pianificando correttamente la viabilità.

Le informazioni ricavate dalle analisi svolte possono essere utili per supportare le decisioni riguardanti la mobilità. In particolare si è scelto di analizzare i dati riguardanti il traffico dei taxi dotati di GPS dell'hotspot D di Manhattan, East Village- Gramercy - MurrayHill, durante tutto l'arco del 2015.

L'approccio utilizzato è stato quello di predire le serie temporali "normali" attraverso un *Predittore* composto da una rete neurale LSTM che saranno usate per il confronto con le serie temporali "anomale" note (come ad esempio il giorno di Natale o il giorno del Ringraziamento). I risultati

verranno usati per addestrare un sistema *Classificatore*, composto da un MLP, a catalogare giorni anomali e giorni normali.

Si è deciso di usare questo approccio in quanto consente di concentrare lo sforzo sull'analisi dei dati ottenuti come risultato del modello che, una volta progettato e costruito, può essere applicato ed adattato al variare degli input.

Si è osservato che la catalogazione è riuscita con uno *score* del 75% nel caso della configurazione migliore, ottenuta variando i parametri e misurando il relativo output, riuscendo meglio nelle giornate in cui la differenza tra le time-series è maggiore.

Indice

Ringraziamenti	i
Abstract	ii
Lista delle figure	viii
1 Introduzione	1
1.1 Motivazioni	1
1.2 Ambiente di programmazione	2
2 Related Works	5
2.1 Lavori precedenti	5
2.2 Vantaggi del metodo utilizzato	6
3 Design e implementazione	8
3.1 Use case	8
3.2 Diagramma delle classi	10
3.2.1 Preprocessing	10
3.2.2 Predittore	12
3.2.3 Classificatore	14
3.3 Classi e implementazione	14

4	Case study	18
4.1	Analisi dei dati utilizzati	18
5	Risultati sperimentali	21
5.1	Preprocessing dei dati	21
5.2	Predizione	25
5.2.1	Cause delle anomalie	26
5.3	Metriche usate	26
5.4	Predizioni al variare dei parametri	27
5.4.1	Generazione time-series giornata normale	27
5.4.2	Classificazione giornate anomale	28
6	Conclusioni	31
A	Codice e manuale d'uso	35
B	Grafici	48

Lista delle figure

1.1	Esecuzione di una predizione LSTM su PyCharm.	3
1.2	Stessa predizione su Google Colab.	3
3.1	Use case del sistema Predittore.	9
3.2	Blocchi funzionali principali.	10
3.3	Diagramma delle classi.	10
3.4	Diagramma di flusso della fase di preprocessing dei dati. . .	11
3.5	Celle LSTM in cascata.	12
3.6	Esempio di un confronto tra time-series reale e predetta. . .	14
3.7	Schema di un multilayer perceptron.	15
4.1	Disposizione degli hotspot di Manhattan.	19
4.2	Dataset nel formato originale.	20
5.1	Time-series delle giornate del cluster 1 con le giornate anomale evidenziate.	23
5.2	Colonna finestra.	24
5.3	Valori tipici della colonna finestra.	24
5.4	Confronti tra i valori di cosine distance.	29

5.5	Esempio di confronto tra time-series: in blu la serie temporale predetta, in rosso quella reale di una giornata normale reale.	29
5.6	Esempio di confronto tra time-series: in blu la serie temporale della giornata anomala reale, in rosso la predizione della giornata normale	30
B.1	In blu la giornata normale predetta e in rosso la giornata reale con indice 25.	48
B.2	In rosso la giornata normale predetta e in blu la giornata anomala reale con indice 26.	49
B.3	In blu la giornata normale predetta e in rosso la giornata reale con indice 81.	49
B.4	In rosso la giornata normale predetta e in blu la giornata anomala reale con indice 82.	50
B.5	In blu la giornata normale predetta e in rosso la giornata reale con indice 141.	50
B.6	In rosso la giornata normale predetta e in blu la giornata anomala reale con indice 142.	51
B.7	In blu la giornata normale predetta e in rosso la giornata reale con indice 152.	51
B.8	In rosso la giornata normale predetta e in blu la giornata anomala reale con indice 153.	52
B.9	In blu la giornata normale predetta e in rosso la giornata reale con indice 187.	52

B.10 In rosso la giornata normale predetta e in blu la giornata anomala reale con indice 188.	53
B.11 In blu la giornata normale predetta e in rosso la giornata reale con indice 204.	53
B.12 In rosso la giornata normale predetta e in blu la giornata anomala reale con indice 205.	54

Lista degli acronimi

GPS	Global Position System.
GPU	Graphical Processing Unit.
IDE	Integrated Development Enviroment.
LSTM	Long Short-Term Memory.
MLP	Multi Layer Perceptron.
RNN	Recurrent Neural Network.
TPU	Tensor Processing Unit.

Chapter 1

Introduzione

1.1 Motivazioni

Gli approcci di anomaly detection nell'ambito dell'analisi del traffico urbano sono recentemente divenuti di grande interesse nella ricerca e permessi dal notevole sviluppo delle tecnologie GPS ad alta precisione.

Riconoscere un'anomalia (o anche *outlier*) può rivelarsi fondamentale nell'analisi del flusso di traffico perchè permetterebbe ad esempio di reindirizzare le auto verso percorsi alternativi in caso di congestioni oppure potenziare le infrastrutture in zone risultate particolarmente soggette a correnti di traffico anomale.

Questo può essere (ed è tuttora fatto) mediante l'uso di tecnici specializzati (ad esempio urbanisti) che valutano gli aspetti sociali, economici e ambientali del trasporto stradale, ferroviario, navale e aereo.

Questo approccio però risulta essere difficilmente scalabile per un grande

numero di siti poichè sarebbe necessario un pool di esperti per ogni diversa città.

Per questi motivi si è iniziato a pensare ad un modo di automatizzare il processo di rilevazione delle anomalie urbane attraverso l'uso di algoritmi di *machine learning*.

La precisione dei risultati inoltre potrebbe risultare anche maggiore se viene scelto in maniera oculata il modello da utilizzare e i parametri da passargli.

Una volta estratti alcuni pattern utili attraverso il modello si possono fare delle considerazioni interessanti, come raggruppare i giorni in base al tipo di mobilità che può variare in base al giorno della settimana oppure, come fatto nel caso di questa tesi, trovare i possibili *outliers*.

1.2 Ambiente di programmazione

L'ambiente di programmazione usato per tutta la durata del progetto di tesi è stato **Google Colab**.

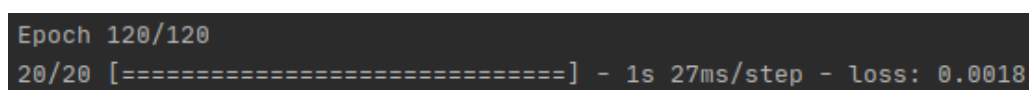
Colab è un servizio cloud offerto da Google basato su *Jupyter notebook* che è un ambiente di programmazione *web-based* usato per creare *notebooks*.

Jupyter notebook è un'applicazione basata sul modello client-server che permette la creazione e la condivisione di documenti strutturati (i *notebooks*) come una lista ordinata di celle. Il contenuto di una cella può

essere codice, testo semplice o anche testo formattato in HTML o Markdown.

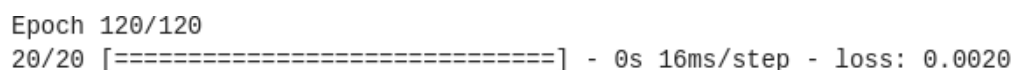
Colab è un'applicazione web che sfrutta il principio dei Jupyter Notebook integrandolo con i servizi Google di condivisione ma soprattutto hosting. Il principale vantaggio di **Colab** infatti è che essendo un servizio cloud si appoggia su server Google per l'esecuzione del codice, garantendo maggiore rapidità di esecuzione rispetto ad un normale pc dato che fornisce la potenza computazionale necessaria, indipendentemente dal dispositivo che si usa.

Il codice viene infatti eseguito in una macchina virtuale e processato attraverso le GPU (Nvidia K80, T4, P4 e P100) e le TPU su Google Cloud.



```
Epoch 120/120
20/20 [=====] - 1s 27ms/step - loss: 0.0018
```

Figure 1.1: Esecuzione di una predizione LSTM su PyCharm.



```
Epoch 120/120
20/20 [=====] - 0s 16ms/step - loss: 0.0020
```

Figure 1.2: Stessa predizione su Google Colab.

Dalle Figure 1.1 e 1.2 risulta evidente il divario tra i tempi di esecuzione. Si è scelto quindi **Colab** al posto di **PyCharm**, un IDE di stampo professionale per lo sviluppo in Python, perchè il primo è appositamente pensato per la *data science* e garantisce anche la gestione automatica delle (numerosi) dipendenze, che altrimenti avrebbero causato

notevoli perdite di tempo.

Chapter 2

Related Works

2.1 Lavori precedenti

In letteratura si possono trovare diversi esempi di *anomaly detection* applicata alla mobilità urbana, che usano approcci eterogenei.

In particolare inizialmente si procedeva attraverso un'individuazione delle anomalie manuale, attraverso l'analisi dei dati forniti dai rapporti annuali o mensili sul traffico. Questa operazione è diventata via via più complicata con l'aumentare del traffico e il conseguente aumento del numero di dati da analizzare. Inoltre anche le città sono profondamente cambiate, soprattutto nella tipologia di traffico che adesso risulta essere molto più eterogeneo e addensato.

Si è reso quindi necessario un meccanismo per automatizzare e riuscire a gestire questa mole di dati. Ci vengono in aiuto gli algoritmi di machine learning che possono essere progettati anche da non esperti nel dominio di

applicazione, i quali invece si occuperanno di analizzarne i risultati.

Gli approcci principalmente usati sono 4:

- statistics-based
- cluster-based
- classification-based
- distance-based
- prediction-based

Si possono anche distinguere diversi tipi di anomalie, come una *singola strada congestionata*, un'*anomalia di flusso* oppure un *incrocio intasato*.

In questa tesi si analizzerà l'incidenza di singolo tipo di anomalie, quelle legate all'intero traffico giornaliero, definendo quindi il concetto di normalità attraverso le time-series caratteristiche di giorni riconosciuti come regolari. Tutto questo sarà fatto usando un modello *prediction-based* in cui la distanza viene misurata rispetto alla predizione della time-series di una giornata normale.

2.2 Vantaggi del metodo utilizzato

L'approccio utilizzato consente di addestrare un modello di machine learning in modo relativamente semplice, fornendo una serie di giornate normali, molto più frequenti rispetto alle giornate anomale senza necessariamente essere esperti programmatori o statistici e nel minor

tempo possibile. Inoltre questo consente sia di fare un'analisi visiva dei risultati, ad esempio attraverso il plotting delle time-series delle giornate si può osservare quanto queste si discostino l'una dall'altra per trarre delle conclusioni che poi potranno essere confermate o meno dal modello.

Un approccio basato sul machine learning risulta inoltre essere più scalabile rispetto ad un approccio statistico anche per il fatto che quest'ultimo deve essere portato avanti per lo più manualmente da un esperto che stabilisce delle soglie che nel modello di machine learning vengono individuate automaticamente.

Inoltre un sistema basato su machine learning riesce a scalare con il numero di dati mantenendo alti standard di performance anche al variare delle condizioni (le città mutano e si evolvono e con loro il traffico), mentre altri approcci, più statici, no.

In particolare reti artificiali ricorrenti come LSTM si sono dimostrate molto più precise rispetto ad altre per questo particolare tipo di anomalie, essendo costruite per lavorare con dati sequenziali proprio come le time-series.

Infatti le RNN LSTM sono ormai diventate lo standard per i modelli di deep learning per la loro abilità a riconoscere pattern di ampiezza elevata.

Chapter 3

Design e implementazione

3.1 Use case

Il sistema pensato permette all'utente di predire anomalie in un set di giorni. La scelta del set di giorni è effettuata in base al numero di giorni normali che precedono un giorno anomalo come verrà spiegato in seguito. Lo use case è riportato in Figura 3.1.

Un utente che avvia il sistema può mettersi alla **ricerca di anomalie** avviando l'interfaccia da linea di comando. Per fare questo basta eseguire il programma che si preoccupa di chiedere all'utente i **parametri di ricerca del Predittore**, ovvero il path dei dati da analizzare (su Colab va fatto l'upload dei dati da usare che in seguito saranno accessibili attraverso il loro nome) e il numero di iterazioni da far eseguire al predittore.

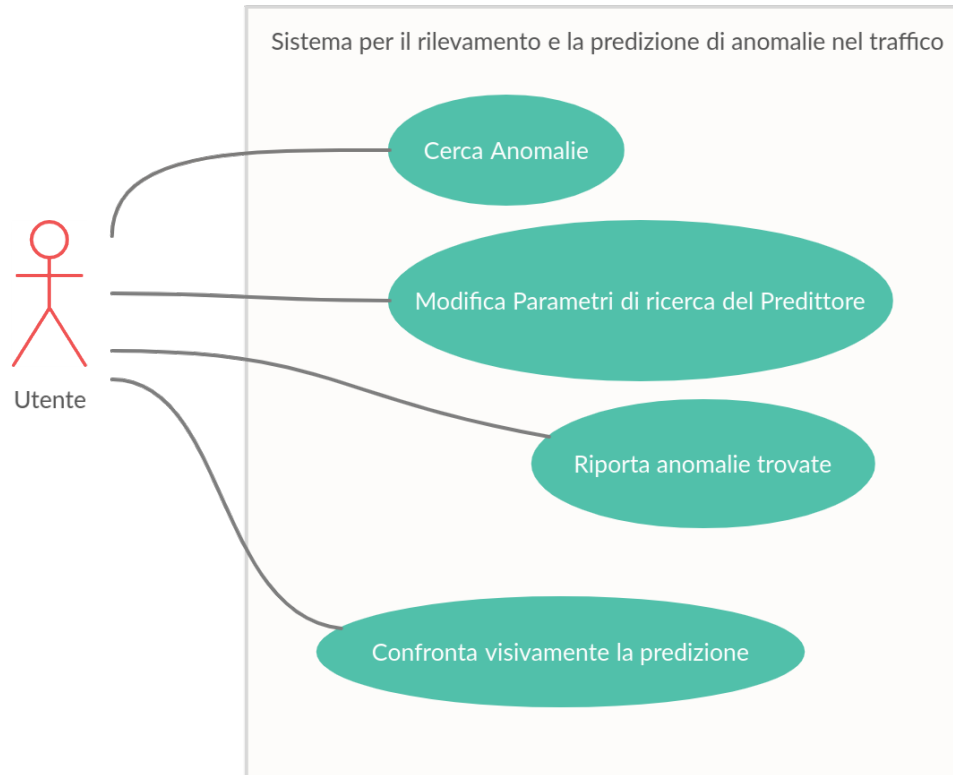


Figure 3.1: Use case del sistema Predittore.

In seguito il sistema si occupa di **riportare le anomalie** rilevate dalla classificazione attraverso una stampa a video dei risultati e dello precisione della misura, inoltre permette all'utente di **confrontare visivamente la predizione** attraverso il salvataggio dei confronti tra anomalie e giornate normali come immagini che hanno come nome l'indice della giornata alle quali si riferiscono.

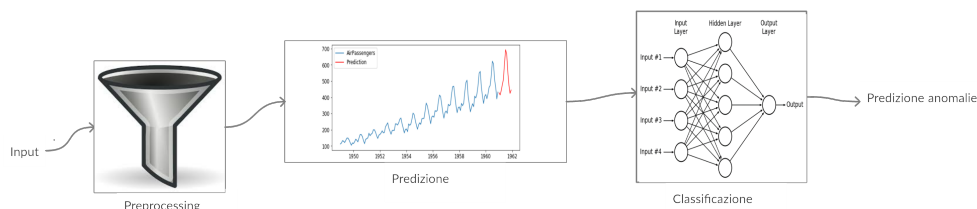


Figure 3.2: Blocchi funzionali principali.

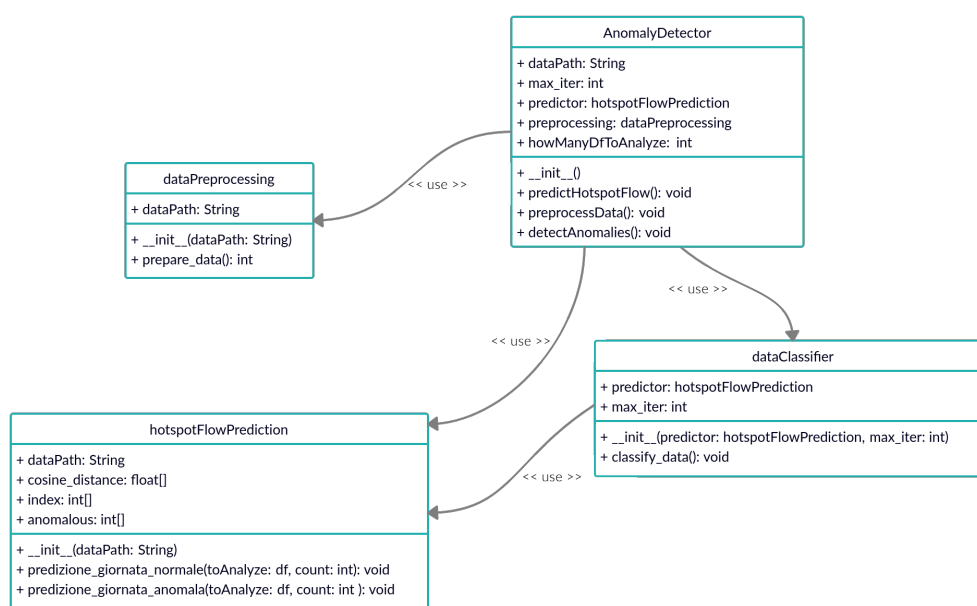


Figure 3.3: Diagramma delle classi.

3.2 Diagramma delle classi

Il diagramma delle classi riportato in Figura 3.2 mostra gli attributi, le operazioni e la relazione tra le classi usate nel codice.

3.2.1 Preprocessing

La fase di preprocessing dei dati si distingue per due scansioni del dataset: la prima per calcolare i valori della "finestra di giorni normali", ovvero il numero i giorni normali che precedono un giorno anomalo. La seconda per

creare i "dataframe target", ovvero i dataframe composti dalle entrate relative ai giorni normali precedenti ad un dato giorno anomalo e dal giorno anomalo stesso.

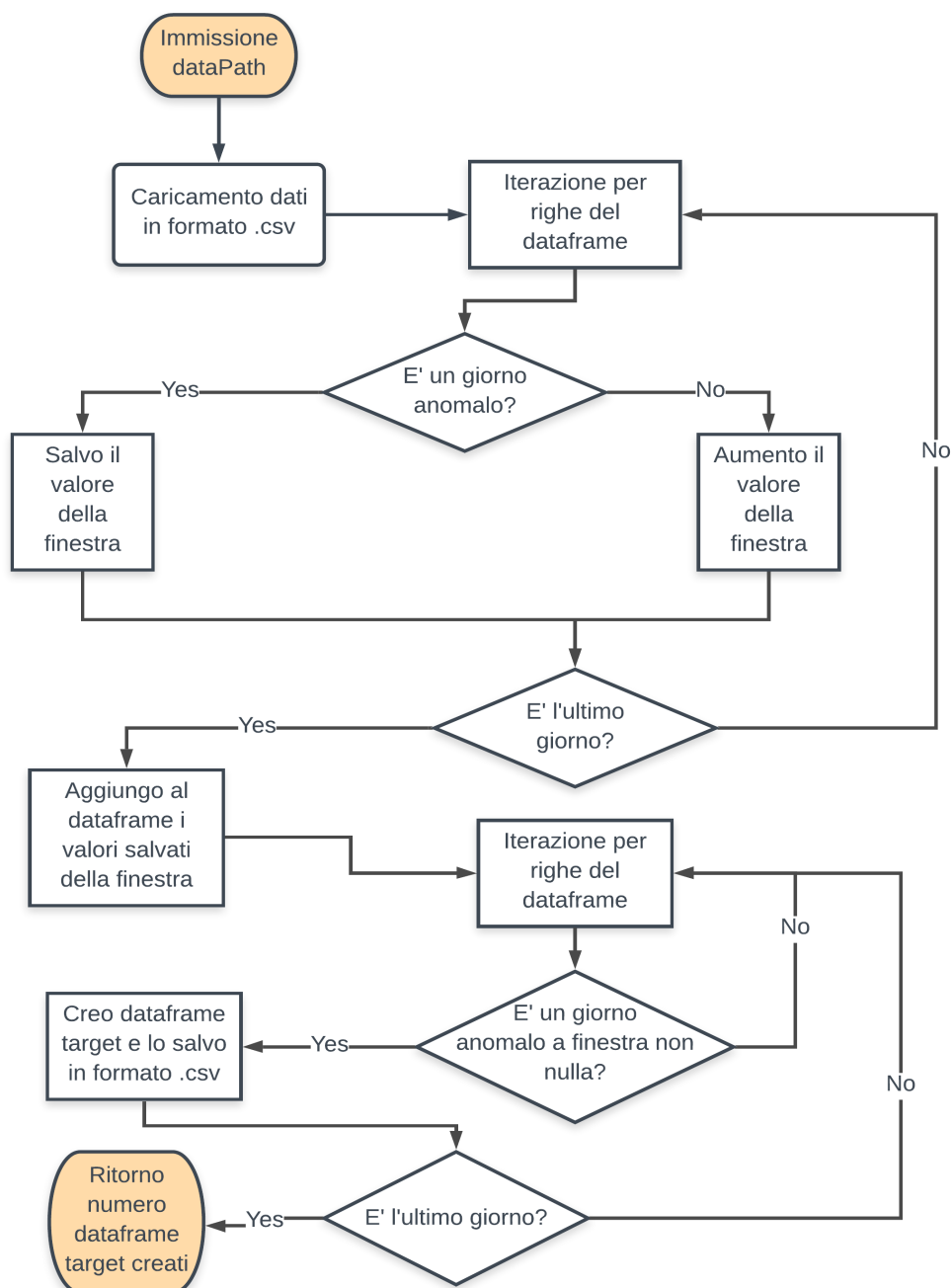


Figure 3.4: Diagramma di flusso della fase di preprocessing dei dati.

3.2.2 Predittore

Il blocco principale è basato su LSTM che si occupa della predizione della serie temporale che verrà usata per il confronto.

LSTM è un tipo di RNN pensata appositamente per analizzare pattern di tipo periodico come quelli temporali, risolvendo i problemi del *vanishing gradient* (scomparsa del gradiente) e del *exploding gradient* (esplosione del gradiente) che rendevano le reti neurali ricorrenti non utilizzabili in questi casi.

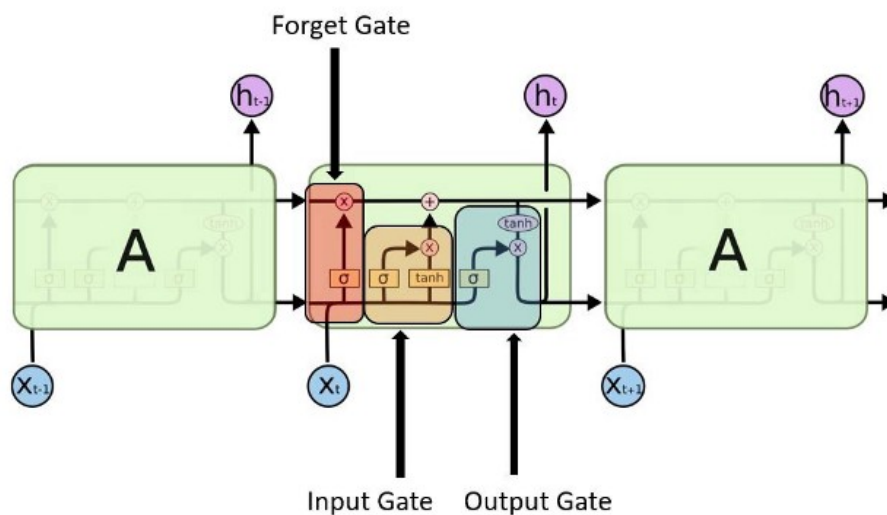


Figure 3.5: Celle LSTM in cascata.

LSTM risolve questi due problemi aggiungendo un'unità (cella) di memoria all'interno della rete. In particolare la rete prende come input il vettore degli ingressi e l'output della precedente cella ma soprattutto la *memoria* della precedente cella. La memoria della cella corrente viene

alterata ogni volta che viene generato un nuovo output che a sua volta andrà in ingresso alla prossima cella e così via.

Un modulo LSTM è composto da 3 *gate*:

- **Forget gate:** decide quali informazioni "dimenticare" e quindi eliminare dalla memoria, decidendo quanto a lungo vanno ricordati i dati.
- **Input gate:** controlla quali informazioni sono state aggiunte alla cella dal nuovo input e decide come modificare la sua memoria in base a questo.
- **Output gate:** decide cosa mostrare in output in base a ciò che è contenuto nella memoria.

Un *gate* è uno strato di una rete neurale con funzione di attivazione *sigmoideale* che si occupa di controllare il flusso di informazioni da/per la memoria.

La rete LSTM è il blocco principale del sistema perchè si occupa di creare le time-series che poi saranno analizzate dal MLP.

Ad esempio è stato riportato un confronto tra una time-series predetta. In blu è possibile vedere la serie temporale generata da LSTM mentre in rosso la serie temporale reale.

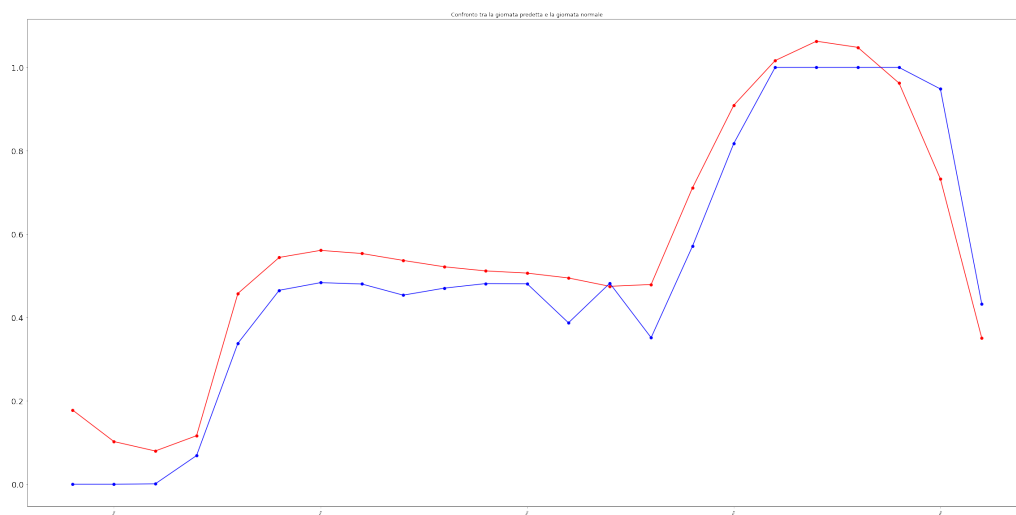


Figure 3.6: Esempio di un confronto tra time-series reale e predetta.

3.2.3 Classificatore

L'ultimo blocco funzionale che compone il sistema è costituito da un *Classificatore*, costituito da un MLP ovvero una rete neurale artificiale che mappa insiemi di dati in ingresso in un insieme di dati in uscita appropriati.

3.3 Classi e implementazione

- **AnomalyDetector:** Classe che si interfaccia con l'utente e che avvia il processo di predizione chiedendo di immettere il *dataPath* dei dati in formato .csv e il *max_iter*, ovvero il numero di epoche per cui deve essere addestrato il MLP. Fa da *wrapper* per le altre classi, invocandole al bisogno e fornisce in output i confronti visivi tra le giornate anomale e normali, lo *score* della predizione e le predizioni

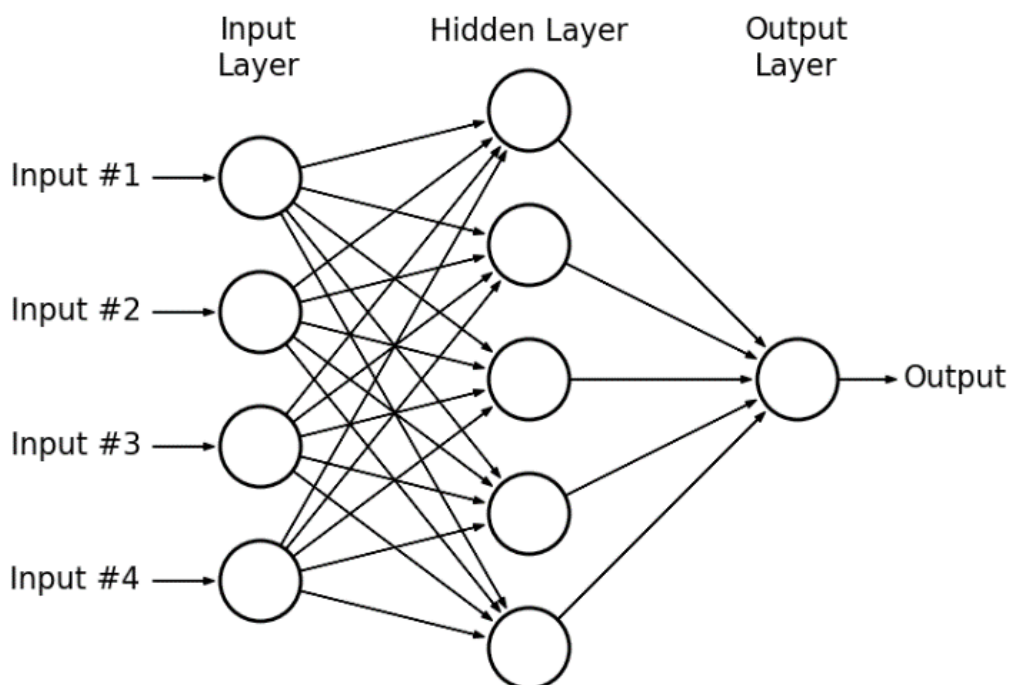


Figure 3.7: Schema di un multilayer perceptron.

stesse.

Metodi:

- *__init__*: costruttore che si occupa di chiedere all'utente il *dataPath* usato per il preprocessing (Figura 3.4) e il *max_iter* per la predizione.
- *predictHotspotFlow*: istanzia la classe *hotspotFlowPrediction* e fa partire la predizione come spiegata nel paragrafo 3.2.2.
- *preprocessData*: fa partire la preprocessazione come mostrata in Figura 3.4.
- *detectAnomalies*: chiama gli altri due metodi e classifica i risultati ottenuti come spiegato al paragrafo 3.2.3.

- **dataClassifier:** Classe che si occupa della classificazione dei dati in *anomalous* o *non anomalous*. Usa la classe *hotspotFlowPrediction* in quanto ha bisogno dei risultati prodotti da questa e in seguito addestra il modello MLP.

Metodi:

- *__init__*: costruttore che inizializza il classificatore.
- *classifyData*: usando la struttura vista nel paragrafo 3.2.3 classifica i risultati ottenuti dalle fasi precedenti e calcola l'*accuracy*.

- **hotspotFlowPrediction:** Classe che si occupa della predizione della time-series per il confronto tra giornate anomale e normali.

Metodi:

- *__init__*: costruttore che inizializza il predittore.
- *predizione_giornata_normale*: predice la serie temporale dell'ultima giornata normale del dataset passato come parametro in base all'architettura vista nel paragrafo 3.2.2 e in Figura 3.5, salva come immagine il confronto come mostrato in Figura 3.6.
- *predizione_giornata_anomala*: predice la serie temporale della giornata anomala del dataset passato come parametro in base all'architettura vista nel paragrafo 3.2.2 e in Figura 3.5, salva come immagine il confronto tra la giornata predetta e quella

anomala.

- **dataPreprocessing:** Classe che prepara i dati in un formato utilizzabile da *hotspotFlowPrediction* e ritorna il numero di dataframe interessanti che sono stati individuati.

Metodi:

- *__init__*: salva il *dataPath* inserito dall'utente all'avvio.
- *prepare_data*: prepara i dati per la predizione e la classificazione come mostrato in Figura 3.4.

Chapter 4

Case study

4.1 Analisi dei dati utilizzati

I dati utilizzati all'interno di questo lavoro sono relativi all'affluenza delle persone all'interno di taxi nell'hotspot D di Manhattan.

Il dataset nel formato originale si presenta come in Figura 4.1.

Ogni riga rappresenta un giorno dell'anno 2015 ed è stata suddivisa in diverse colonne.

La colonna *Anomalous* rappresenta l'indicazione di anomalia di un giorno e può assumere solo due valori interi:

- **0** : La giornata non è anomala.
- **1** : La giornata in questione è risultata anomala.

La colonna successiva è quella che indica il *Cluster* di appartenenza del

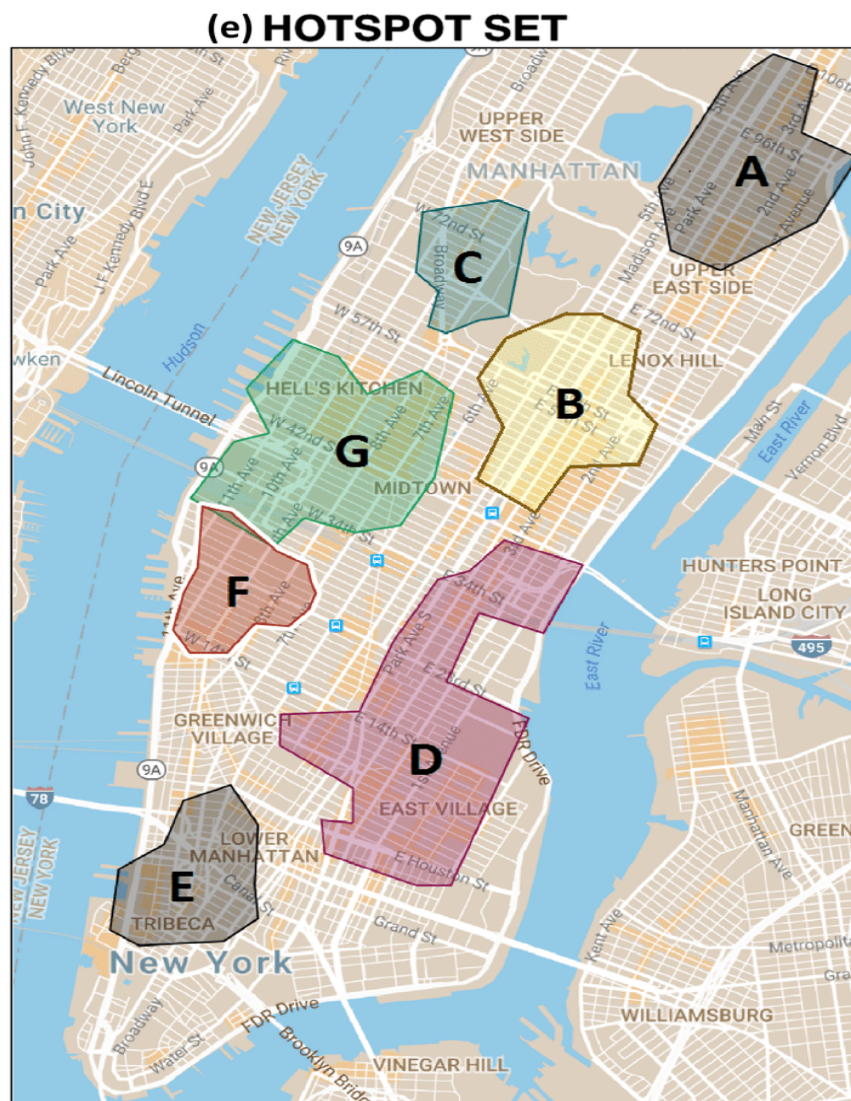


Figure 4.1: Disposizione degli hotspot di Manhattan.

giorno. Il valore di *Cluster* può essere uno tra tre diversi interi:

- **0:** Giorno lavorativo
- **1:** Giorno prefestivo
- **2:** Giorno festivo

La divisione dei giorni in cluster risulta essere molto utile perchè ci permette di analizzare giorni con caratteristiche simili. Infatti all'interno

Anomalous	Cluster	Day	Month	h1	h2	h3	h4
1	0	1	1	1	1	0	0.299890843006513
0	1	2	1	0.174002633779448	0.181794500058363	0.178889335639729	0.195044014019326
0	1	3	1	0.440282061270347	0.244073007665684	0.18111329392414	0.155451962389229
0	2	4	1	0.52200292633957	0.314627518815994	0.218219823286538	0.178846155998051
0	0	5	1	0.0111238823171337	0.0104769275911869	0.0915830617560105	0.212211056932614
0	0	6	1	0.180894763647734	0.179121350642964	0.259688229189631	0.412114195284238
0	0	7	1	0.0132592658560402	0.0117340681323592	0.0749171748221411	0.523400975100692
0	0	8	1	0.00043127904942332	0.0733017724750458	0.347565055273292	0.334623395821251
0	1	9	1	0.178772742422137	0.180862905502745	0.191886106975021	0.347565055273292
0	1	10	1	0.287481127848009	0.207179017641216	0.181431769992891	0
0	2	11	1	0.943815831663501	0.314468267646539	0.197603631618898	0.180103431594463
0	0	12	1	0.18123343741745	0.181228927245506	0.274471483529275	0.195667486364899
0	0	13	1	0.180991582406241	0.180097744937258	0.248268177679953	0.647636353365833
0	0	14	1	0.176611594927608	0.175840388445523	0.255555305832107	0.464878391425721
0	0	15	1	0.180409663741054	0.179439721624205	0.241239421258253	0.501617837652304
0	1	16	1	0.180204689391693	0.181017006625017	0.188776074103501	0.44333310315734
0	1	17	1	0.412309181522376	0.214971021742188	0.181339246830589	0
0	2	18	1	0.862443383544026	0.285559785751612	0.179535861141788	0.180460385732845
0	0	19	1	0.172869681713448	0.184272930360431	0.180068395570137	0.178129729237394
0	0	20	1	0.180542792148107	0.178663580236577	0.257729143326493	0.200361017903047
0	0	21	1	0.180543094311304	0.179445371616453	0.246937358458255	0.531775780826533
0	0	22	1	0.1811333492504	0.18097665146378	0.22174088799184	0.497995297101902
0	1	23	1	0.178055144769271	0.181077910023792	0.183028557319063	0.431409969230582
0	1	24	1	0.287481127848009	0.216367530115318	0.18142267717955	0
0	2	25	1	1	0.333333333333333	0.207106773637479	0.180357026628262
0	0	26	1	0.158485506938439	0.172347567750146	0.431221691772013	0.204879409311249
							0.751018558628533

Figure 4.2: Dataset nel formato originale.

dello stesso cluster i giorni avranno delle time-series abbastanza simili, cosa che ci consentirà di individuare più facilmente le eventuali anomalie.

Dopo *Anomalous* e *Cluster* sono riportate due colonne che rappresentano il giorno e il mese all'interno dell'anno 2015.

In seguito ci sono 23 colonne, una per ogni ora del giorno in cui si sono effettuate le rilevazioni. Queste colonne possono assumere valori che, normalizzati, vanno da 1 a 0.

Il dataset in questo formato non risulta essere particolarmente facile da usare per l'analisi quindi è stata necessaria un'operazione di *preprocessing* attuata al fine di renderli più uniformi e "comodi" per il sistema Predittore.

Chapter 5

Risultati sperimentali

5.1 Preprocessing dei dati

La prima cosa che si è resa necessaria è stata quella di preprocessare i dati, per portarli in un formato consistente ed utilizzabile.

Infatti si possono individuare due maggiori problematiche nei dati in formato originale:

- **Mancanza di un indice:** La mancanza di un indice rende più difficoltoso l'accesso ai dati in quanto ogni volta che si vuole accedere ad un dato indice bisogna controllare, nel caso si voglia considerare la data, sia il campo *Giorno* che quello *Mese*.
- **Presenza di molteplici colonne per ogni riga:** Questo rende più difficile l'analisi dei dati in quanto ogni riga contiene l'informazione relativa all'intera giornata, dovendo quindi ogni volta cancellare le colonne non desiderate.

Per risolvere i problemi citati si è innanzitutto creato un campo *Data* che indicizzasse il *dataframe* andando ad unire il campo *Giorno* e *Mese*, aggiungendo come anno quello di riferimento, il 2015.

Dopo questo si sono giustapposte tutte le colonne relative alle affluenze orarie, andando a creare un'entrata per ogni ora. Quindi ogni giornata è adesso rappresentata da 23 entrate.

Questo da un lato aumenta le dimensioni del *dataframe* ma rende la predizione delle time-series molto più semplice, perchè per predire una giornata non bisognerà fare altro che chiedere di predire 23 input dopo aver passato come set di train tutte le giornate precedenti, senza bisogno di fare altre modifiche se non una *normalizzazione*.

Il **feature scaling** (o *ridimensionamento dei dati*) permette di standardizzare le caratteristiche dei dati in un intervallo prestabilito, nel nostro caso $[0,1]$.

Dopo aver portato i dati in questo formato, è necessario un ulteriore passaggio: il nostro obiettivo è infatti quello di andare a confrontare la time-series di una giornata normale con quella di una giornata anomala. Non potendo scegliere una giornata anomala a caso dal dataset per il confronto (non sarebbe molto indicativo) si è scelto di predire l'andamento di una giornata normale, cosa facilmente fattibile dato che nel dataset sono presenti molte più giornate normali che giornate anomale.

Quindi per prima cosa si è scelto il cluster 1, quello più popolato, per assicurare che tutte le time-series considerate siano abbastanza simili, in

modo da individuare più facilmente le anomalie.

All'interno del dataset ottenuto attraverso tutte le giornate del cluster 1, si sono processati nuovamente i dati per portarli nel loro formato finale.

Innanzitutto si sono dotate le righe di un indice numerico incrementale, più semplice da gestire rispetto alla data, e dovendo andare a confrontare singolarmente ogni giornata anomala con una sola giornata normale predetta di riferimento, si è diviso il dataset in vari dataset, uno per ogni giornata anomala reputata *"di interesse"*.

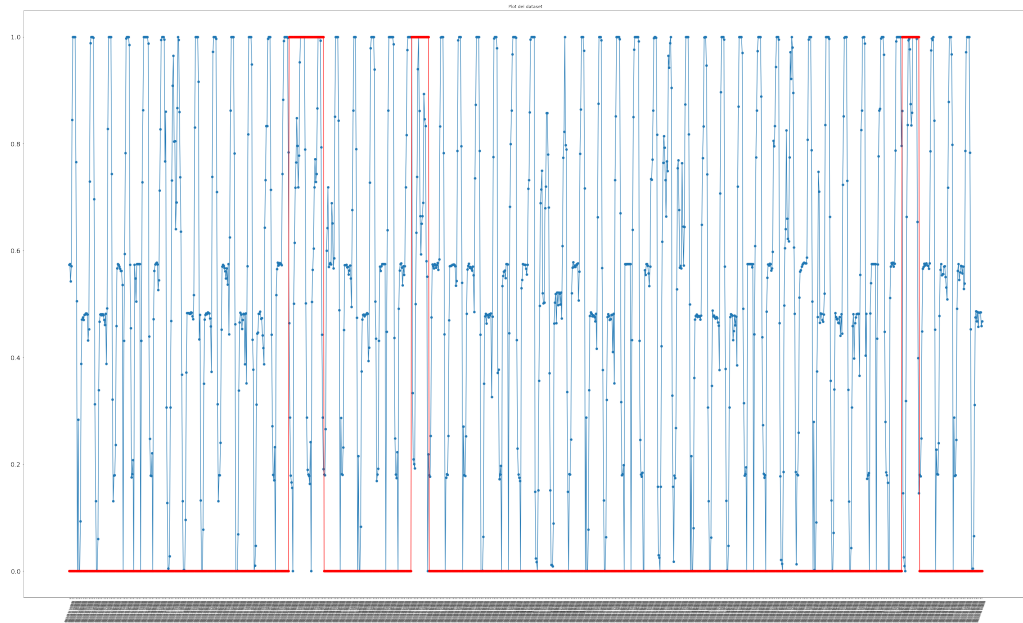


Figure 5.1: Time-series delle giornate del cluster 1 con le giornate anomale evidenziate.

Per essere *"di interesse"* una giornata anomala, con il valore del campo *Anomalous* settato a 1, deve avere un numero minimo di giorni normali che la precedono per rendere la predizione della giornata da usare nel confronto il più precisa possibile. Infatti l'ipotesi è che maggiore è il

numero di giorni della cosiddetta "*Finestra*" (*Window* nel codice in appendice A) di giorni normali, più precisa sarà la predizione della time-series normale e più saranno evidenti le differenze con la time-series della giornata anomala.

	Index	Affluenza	Anomalous	Finestra
0	1	1.0	1	0
1	1	0.0	1	0
2	1	0.29989084300651303	1	0
3	1	0.195044014019326	1	0
4	1	0.198812234343099	1	0
5	1	0.194608629371612	1	0
6	1	0.195873221700534	1	0
7	1	0.0	1	0
8	1	0.5836151837018779	1	0
9	1	0.653947165851926	1	0

Figure 5.2: Colonna finestra.

	Index	Affluenza	Anomalous	Finestra
206	10	0.0	1	184
1862	82	0.37736370310840794	1	1265
3242	142	0.343491640559844	1	1357

Figure 5.3: Valori tipici della colonna finestra.

Il numero medio di giorni normali che compongono la *Finestra* è risultato essere 22, numero che se impostato come *soglia di interesse*, ovvero soglia sotto la quale i dati vengono reputati non interessanti, avrebbe permesso di analizzare le time-series di 3 sole giornate. Quindi è stato scelto un valore di soglia pari a 10, compromesso che permette di ottenere una ancora una buona precisione nella predizione e permette di osservare le anomalie di 6 giornate su un totale di 9 presenti nel dataset.

Dopo aver spezzato il dataset in vari dataframe composti da una serie di

giornate normali e la giornata anomala di riferimento, si può passare alla fase della predizione.

5.2 Predizione

Per ogni giornata anomala del dataset si va a predire, fornendo come input tutte le giornate normali che lo precedono, la time-series della giornata normale. Questo può risultare già utile per un confronto visivo attraverso il quale già si possono notare differenze tra le time-series che potranno essere confermate durante la classificazione come mostrato in Appendice B.

A questo punto però, oltre al semplice confronto visivo si rende necessario un confronto più scientifico. Questo può essere fatto calcolando la *cosine distance* tra il dataframe relativo alla giornata anomala e quello della giornata normale.

La *cosine distance* calcola la distanza tra due vettori che possiamo usare, sempre impostando una soglia, per distinguere un giorno normale da uno anomalo. Di questo si occuperà *MLP* al quale passeremo il vettore contenente tutte le *cosine distance* calcolate.

Infatti sarà il *MLP* a stabilire questa soglia in automatico per ogni giorno anomalo presente nel dataset.

Index	Data	Anomalia/Non anomalia	cosine distance
25	12/02	Non anomalo	0,02778879449117011
26	16/02	Anomalo	0,05931894169752272
81	21/5	Non anomalo	0,010871693489237777
82	25/5	Anomalo	0,07330551350792547
141	3/9	Non anomalo	0,03131250329642299
142	7/9	Anomalo	0,0393965106382177
152	23/9	Non anomalo	0,01938268993867709
153	24/9	Anomalo	0,02598689095459017
187	24/11	Non anomalo	0,009974987075688113
188	25/11	Anomalo	0,020252144437057584
204	23/12	Non anomalo	0,01848881759983234
205	24/12	Anomalo	0,017218302138751196

Table 5.1: Valori di cosine distance per i giorni analizzati.

5.2.1 Cause delle anomalie

- 16/02: Compleanno di George Washington, primo presidente degli Stati Uniti.
- 25/5: Memorial Day.
- 7/9: Festa dei lavoratori.
- 24/9: Visita di Papa Francesco negli Stati Uniti.
- 25/11: Vigilia giorno del ringraziamento
- 24/12: Vigilia di Natale

5.3 Metriche usate

Per stabilire la precisione e l'accuratezza dei risultati ottenuti si è scelto di usare la funzione *score*. La funzione in questione ritorna la bontà della

max_iter	accuracy
5	0.5
200	0.75
350	0.6333
500	0.633
1000	0.583

Table 5.2: Valori di cosine distance per i giorni analizzati.

predizione effettuata confrontandola con i valori di test, attraverso un valore da 0 a 1, dove 1 sta a significare che ogni valore è stato predetto in modo corretto e 0 che nessun valore è stato predetto correttamente.

Sono state effettuate diverse predizioni variando i parametri usati (che saranno illustrati nel paragrafo successivo) e i valori di *score* sono variati ma rimanendo abbastanza affidabili.

Lo *score* migliore (e anche quello più stabile) è stato ottenuto con la configurazione che prevede un valore di *max_iter* pari a 200.

5.4 Predizioni al variare dei parametri

I parametri caratteristici del sistema sono diversi per la generazione della time-series e per la classificazione delle giornate in anomale o non anomale.

5.4.1 Generazione time-series giornata normale

In questo caso i parametri caratteristici sono:

- **batch_size:** Dimensione dei set in cui verranno suddivisi i dati prima di essere dati in pasto al modello.
- **epochs:** Numero di generazioni per le quali verrà addestrata la rete.

Nel nostro caso come configurazione di default è stata scelta quella con una *batch_size* pari a 32 e un numero di *epochs* pari a 100.

5.4.2 Classificazione giornate anomale

In questo caso i parametri caratteristici sono:

- **max_iter:** Analogo al numero di epoche del modello LSTM.
- **hidden_layer_sizes:** Rappresenta la dimensione e il numero di strati di neuroni.

Nella nostra predizione il valore di *max_iter* è stato variato, per trovare la configurazione migliore, che si è dimostrata essere quella di valore 200.

Il valore di *hidden_layer_sizes* è stato settato a (100,100), in questo modo si aggiunge uno strato di neuroni nascosti che migliorano la precisione del sistema.

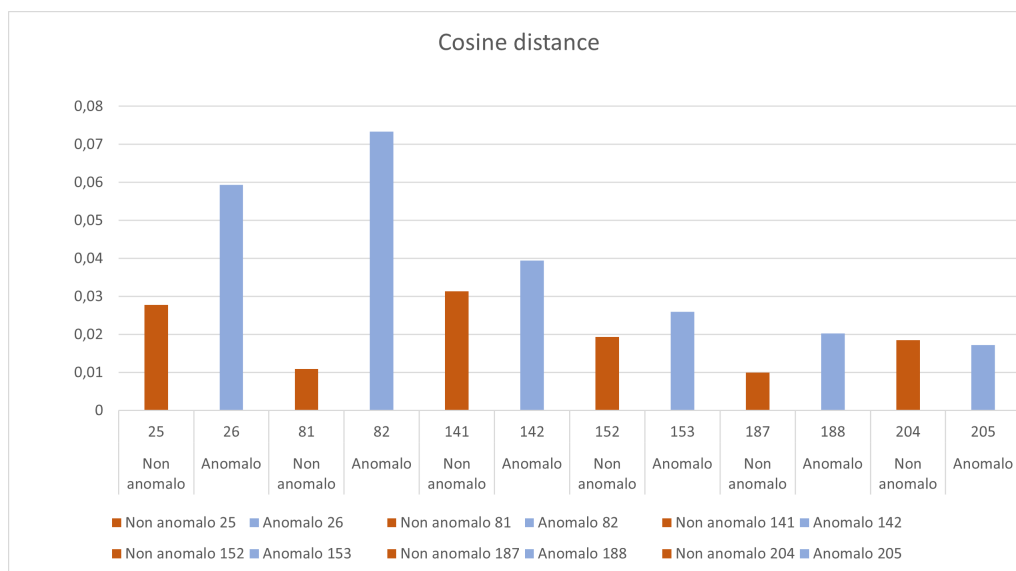


Figure 5.4: Confronti tra i valori di cosine distance.

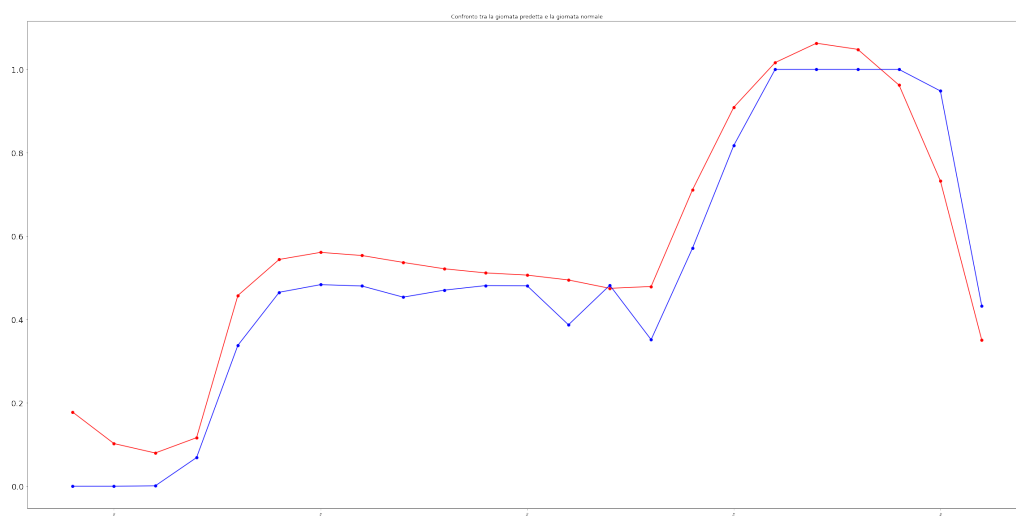


Figure 5.5: Esempio di confronto tra time-series: in blu la serie temporale predetta, in rosso quella reale di una giornata normale reale.

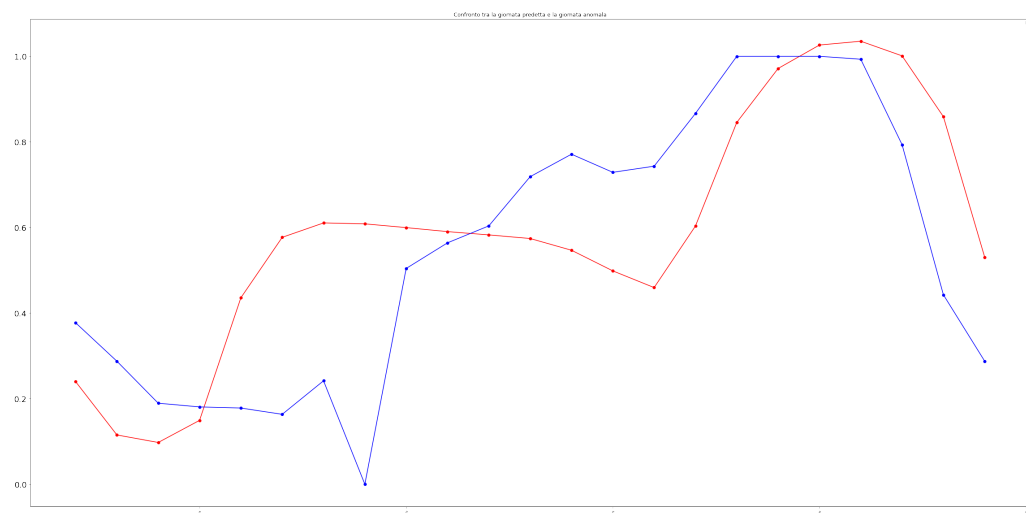


Figure 5.6: Esempio di confronto tra time-series: in blu la serie temporale della giornata anomala reale, in rosso la predizione della giornata normale

Chapter 6

Conclusioni

Alla conclusione del lavoro svolto si può affermare che i risultati dell'analisi sono stati soddisfacenti. Infatti il sistema progettato ed implementato si è dimostrato in grado di predire con successo il 75% delle giornate analizzate, riuscendo a catalogarle come non anomale o anomale correttamente.

Questo nonostante si sia scelto come valore di soglia della finestra un valore abbastanza basso pari a 10, reso però necessario dal desiderio di analizzare quante più giornate possibili in un'unica esecuzione.

Inoltre per i risultati non predetti in modo corretto dal Classificatore ci si può avvalere delle immagini prodotte, che sono un valido ausilio in fase decisionale. Infatti anche attraverso una ispezione visiva delle immagini riportate in appendice è possibile percepire le differenze tra le serie temporali delle giornate normali e quelle delle giornate anomale.

Bibliography

- [1] Raghavendra Chalapathy, Sanjay Chawla, “Deep Learning for anomaly detection: a survey.” <https://arxiv.org/pdf/1901.03407.pdf>, Gennaio 2019. Accessed: 2020-11-07.
- [2] Google Colab, “Colab documentation.” https://colab.research.google.com/github/d2l-ai/d2l-en-colab/blob/master/chapter_computational-performance/hardware.ipynb. Accessed: 2020-11-07.
- [3] Weiming Kuang, Shi An, Huifu Jiang, “Detecting traffic anomalies in urban areas using taxi gps data,” *Mathematical Problems in Engineering*, vol. 2015, no. ID 809582, p. 13 page, Settembre 2015.
- [4] Ian Felton, “A quick example of time-series prediction using long short-term memory (lstm) networks.” <https://medium.com/swlh/a-quick-example-of-time-series-forecasting-using-long-short-term-memory-lstm> 2 Agosto 2019. Accessed: 2020-11-07.
- [5] A. L. Alfeo, M. G. C. A. Cimino, S. Egidi, B. Lepri, A. Pentland, G. Vaglini, “Stigmergy-based modeling to discover urban activity

- patterns from positioning data,” *in proceedings of “Social, Cultural, and Behavioral Modeling: 10th International Conference”, (SBP-BRiMS 2017)*, vol. 10354, pp. 292–302, 2017.
- [6] JetBrains, “Pycharm documentation.” <https://www.jetbrains.com/pycharm/documentation/>. Accessed: 2020-11-07.
- [7] Shi Yan, “Understanding lstm and its diagrams.” <https://medium.com/mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714>, 13 Marzo 2016. Accessed: 2020-11-07.
- [8] Ahmet ÖZLÜ, “Long short term memory (lstm) networks in a nutshell.” <https://medium.com/@ahmetozlu93/long-short-term-memory-lstm-networks-in-a-nutshell-363cd470ccac>, Giugno 2020. Accessed: 2020-11-07.
- [9] Deeplearning.net, “Multilayer Perceptron.” <http://deeplearning.net/tutorial/mlp.html>. Accessed: 2020-11-07.
- [10] Google Colab, “Understanding LSTM networks.” <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>. Accessed: 2020-11-07.
- [11] Department of Civil Service, “Elenco festività civili/religiose.” https://www.cs.ny.gov/attendance_leave/2015_legal_holidays.cfm. Accessed: 2020-11-09.

- [12] Wikipedia, “Visita di Papa Francesco.” https://en.wikipedia.org/wiki/Pope_Francis%27s_2015_visit_to_North_America. Accessed: 2020-11-09.

Appendice A

Codice e manuale d'uso

```
1 import pandas as pd
2 from matplotlib import pyplot as plt
3 from sklearn.preprocessing import StandardScaler,
    MinMaxScaler
4 from keras.preprocessing.sequence import TimeseriesGenerator
5 from sklearn.metrics.pairwise import cosine_similarity
6 from sklearn.neural_network import MLPClassifier
7 from sklearn.datasets import make_classification
8 from keras.models import Sequential
9 from keras.layers import Dense
10 from keras.layers import LSTM
11 from keras.layers import Dropout
12 import numpy as np
13 from scipy.spatial import distance
14 import warnings
15 warnings.filterwarnings("ignore")
16
```

```
17 """Classe della preparazione dei dati:
18 - Prendo in ingresso il path dei dati da preparare e salva i
    csv dei dati opportunamente formattati pronti per essere
    usati dal predittore. In particolare aggiunge una
    colonna "Window" che rappresenta il numero di giorni
    normali che precedono un giorno anomalo.
19 Questa colonna ha come valore di default "0" e contiene
    valori significativi solo nella prima entrata relativa ad
    un giorno anomalo.
20 """
21
22 class dataPreprocessing:
23
24     def __init__(self, dataPath):
25         self.dataPath = dataPath
26
27     def prepare_data(self):
28
29         df = pd.read_csv(self.dataPath)
30         count = 0
31         # mantiene il numero di giorni normali che precedono un
32         # giorno anomalo
33
34         window_normal_day = []
35
36         for row in df.itertuples():
37             if row[3] == 0:
```

```
38         else:
39             window_normal_day.append(count)
40             count = 0
41
42     df["Window"] = window_normal_day
43     df.to_csv("window.csv")
44
45     count = 0
46
47     for row in df.itertuples():
48         if (row[3] == 1) & (row[4] != 0):
49             # considero i giorni normali precedenti pi il
50             giorno anomalo attuale
51
52             df_target = df.iloc[row[0] - row[4] : row[0] + 23]
53             # salvo ogni dataframe in modo separato
54
55             df_target.to_csv("dati" + str(count) + ".csv")
56             count = count + 1
57
58     # ritorno il numero di dataframe prodotti
59     return count
60
61 """Classe predittore:
62 - predizione_giornata_normale:
63 prende come **input** una riga dal file dati_riorganizzati
64 che rappresenta un dataframe nel nuovo formato composto
65 da tutte le normali prima di un'unica giornata anomala e
66 un contatore utile per salvare l'immagine di output.
```

```
61 Come **output** fornisce il confronto visivo e la cosine
    distance tra la predizione effettuata su tutte le
    giornati normali precedenti all'ultima (normale) e quest'
    ultima.
62
63 - predizione_giornata_anomala: stessi **input** del metodo
    predizione_ultima_giornata_non_anomala ma questa volta si
    shifta di una giornata, non considerando quindi la prima
    giornata normale e includendo l'ultima giornata normale.
    L'**output** un confronto visivo e la cosine distance
    tra la predizione e la giornata anomala.
64 """
65
66 class hotspotFlowPrediction:
67
68     def __init__(self, dataPath):
69         self.dataPath = dataPath
70         self.cosine_distance = []
71         self.index = []
72         self.anomalous = []
73
74     def predizione_giornata_normale(self, toAnalyze, count):
75
76         # per predire ultima giornata non anomala uno i valori
77         che vanno da 0 a [(indice_giorno_anomalo - 1) - 23]
78         df_without_anomalies = toAnalyze[toAnalyze["Anomalous"]
79 == 0]
80
81         df_target = df_without_anomalies[0:-23]
```



```
79     df_to_predict = df_without_anomalies[-23:]
80
81     n_input = 23
82     n_features = 1
83
84     # tolgo le colonne che non mi interessano adesso
85     del df_target["Index"]
86     del df_target["Anomalous"]
87     del df_target["Window"]
88
89     print(df_to_predict)
90
91     train = df_target
92     scaler = MinMaxScaler()
93     scaler.fit(train)
94     train = scaler.transform(train)
95
96     # parametri: batch_size=32, epochs=100
97     generator = TimeseriesGenerator(train, train, length=
n_input, batch_size=32)
98     model = Sequential()
99     model.add(LSTM(200, activation='relu', input_shape=(
n_input, n_features)))
100     model.add(Dropout(0.15))
101     model.add(Dense(1))
102     model.compile(optimizer='adam', loss='mean_squared_error
')
103     model.fit(generator, epochs=90)
```

```
104
105     pred_list = []
106     batch = train[-n_input:].reshape((1, n_input, n_features
107 ))
108
109     for i in range(n_input):
110         pred_list.append(model.predict(batch)[0])
111         batch = np.append(batch[:,1:,:], [[pred_list[i]]], axis
112 =1)
113
114     df_predict = pd.DataFrame(scaler.inverse_transform(
115 pred_list), index=df_without_anomalies[-n_input:].index,
116 columns=['Prediction'])
117
118     plt.figure(figsize=(40,20))
119
120     plt.title("Confronto tra la giornata predetta e la
121 giornata normale")
122
123     plt.plot(df_predict.index, df_predict["Prediction"],
124 color='r', marker="o")
125
126     plt.plot(df_to_predict.index, df_to_predict["Affluenza"],
127 color = "b", marker="o")
128
129     plt.legend(loc='best', fontsize='small')
130
131     plt.xticks(fontsize=6)
132
133     plt.yticks(fontsize=18)
134
135     plt.xticks(rotation=70)
136
137     # salvo l'immagine del confronto
```

```
124     plt.savefig('confronto_normale' + str(count) + '.png',
125               bbox_inches='tight')
126
127     # stampo a video e salvo la cosine distance
128     print("Cosine distance: " + str(distance.cosine(
129         df_predict["Prediction"], df_to_predict["Affluenza"])))
130
131     self.index.append(df_to_predict["Index"].max())
132
133     self.cosine_distance.append(distance.cosine(df_predict["
134         Prediction"], df_to_predict["Affluenza"]))
135
136     self.anomalous.append(0)
137
138
139     def predizione_giornata_anomala(self, toAnalyze, count):
140
141         # per predire la giornata anomala dobbiamo prendere le
142         # giornate che vanno dalla seconda all'ultima normale
143
144         df_without_anomalies = toAnalyze[toAnalyze["Anomalous"]
145             == 0]
146
147         df_anomalous_day = toAnalyze[toAnalyze["Anomalous"] ==
148             1]
149
150         # tolgo la prima giornata
151
152         df_target = df_without_anomalies[:23]
153
154         df_to_predict = df_without_anomalies.drop(df_target.
155             index)
156
157         n_input = 23
```

```
145     n_features = 1
146
147     # tolgo le colonne che non mi servono
148     del df_to_predict["Index"]
149     del df_to_predict["Anomalous"]
150     del df_to_predict["Window"]
151
152     print(df_anomalous_day)
153
154     train = df_to_predict
155     scaler = MinMaxScaler()
156     scaler.fit(train)
157     train = scaler.transform(train)
158
159     # parametri: batch_size=32, epochs=100
160     generator = TimeseriesGenerator(train, train, length=
n_input, batch_size=32)
161     model = Sequential()
162     model.add(LSTM(200, activation='relu', input_shape=(
n_input, n_features)))
163     model.add(Dropout(0.15))
164     model.add(Dense(1))
165     model.compile(optimizer='adam', loss='mean_squared_error
')
166     model.fit(generator, epochs=90)
167
168     pred_list = []
```

```
169     batch = train[-n_input:].reshape((1, n_input, n_features
170 ))
171
172     for i in range(n_input):
173         pred_list.append(model.predict(batch)[0])
174         batch = np.append(batch[:,1:,:], [[pred_list[i]]], axis
175 =1)
176
177     df_predict = pd.DataFrame(scaler.inverse_transform(
178 pred_list), index=df_to_predict[-n_input:].index, columns
179 =['Prediction'])
180
181     plt.figure(figsize=(40,20))
182
183     plt.title("Confronto tra la giornata predetta e la
184 giornata anomala")
185
186     plt.plot(df_anomalous_day.index, df_predict["Prediction"
187 ], color='r', marker="o")
188
189     plt.plot(df_anomalous_day.index, df_anomalous_day["
190 Affluenza"], color = "b", marker="o")
191
192     plt.legend(loc='best', fontsize='small')
193
194     plt.xticks(fontsize=6)
195
196     plt.yticks(fontsize=18)
197
198     plt.xticks(rotation=70)
199
200
201     plt.savefig('confronto_anomalia' + str(count) + '.png',
202 bbox_inches='tight')
```

```
188     print("Cosine distance: " + str(distance.cosine(
189         df_predict["Prediction"], df_anomalous_day["Affluenza"])))
190     self.index.append(df_anomalous_day["Index"].max())
191     self.cosine_distance.append(distance.cosine(df_predict["
192         Prediction"], df_anomalous_day["Affluenza"])))
193     self.anomalous.append(1)
194
195     """Classe del classificatore:
196
197     - classify_data: metodo che prende in ingresso i dati
198       preparati e il valore di max_iter inserito dall'utente e
199       stampa come output lo score della predizione e la
200       predizione delle label "Anomalous".
201
202     """
203
204     class dataClassifier:
205
206     def __init__(self, predictor, max_iter):
207
208         self.predictor = predictor
209
210         self.max_iter = max_iter
211
212     def classify_data(self):
213
214         df = pd.DataFrame(self.predictor.cosine_distance, columns
215             =["Cosine distance"], index=self.predictor.index)
216
217         df["Anomali reali"] = self.predictor.anomalous
218
219         X = np.array(self.predictor.cosine_distance).reshape
220             (-1,1)
```

```
209     y = self.predictor.anomalous
210
211     # parametri della classificazione: max_iter= passato
dall'utente, hidden_layer_size = (100,100)
212     clf = MLPClassifier(max_iter = int(self.max_iter),
hidden_layer_sizes=(100,100))
213     clf.fit(X,y)
214
215     # predizione e print dello score
216     predicted = clf.predict(X)
217     print("Score: %f" % clf.score(X, y))
218
219     df["Anomalie predette"] = predicted
220     print(df)
221
222     """Interfaccia da linea di comando che permette di inserire
i parametri necessari."""
223
224 class anomalyDetector:
225
226     def __init__(self):
227         self.dataPath = input('Inserisci il path dei dati da
analizzare: ')
228         self.max_iter = input("Inserisci il max_iter da usare: "
)
229
230     def predictHotspotFlow(self):
231         self.predictor = hotspotFlowPrediction(self.dataPath)
```

```
232
233 def preprocessData(self):
234     self.preprocessing = dataPreprocessing(self.dataPath)
235     # quanti dataframe sono stati preparati dal
236     # preprocessatore
237     self.howManyDfToAnalyze = self.preprocessing.
238     prepare_data()
239
240 def detectAnomalies(self):
241     self.predictHotspotFlow()
242     self.preprocessData()
243
244     counter = 0
245
246     for i in range(self.howManyDfToAnalyze):
247         with open("dati" + str(i) + ".csv") as f:
248             df_to_analyze = pd.read_csv(f, index_col="Unnamed: 0"
249             )
250
251             # 10 il valore di soglia che permette di
252             # considerare 6 giorni anomali su 9
253             # con una discreta precisione
254             if df_to_analyze["Window"].max() >= 10*23:
255                 self.predictor.predizione_giornata_normale(
256                 df_to_analyze, counter)
257                 self.predictor.predizione_giornata_anomala(
258                 df_to_analyze, counter)
```



```
254         counter += 1
255
256         c = dataClassifier(self.predictor, self.max_iter)
257         c.classify_data()
258
259         """Creazione dell'istanza di una classe di Interfaccia."""
260
261     cli = anomalyDetector()
262     cli.detectAnomalies()
```

Appendice B

Grafici

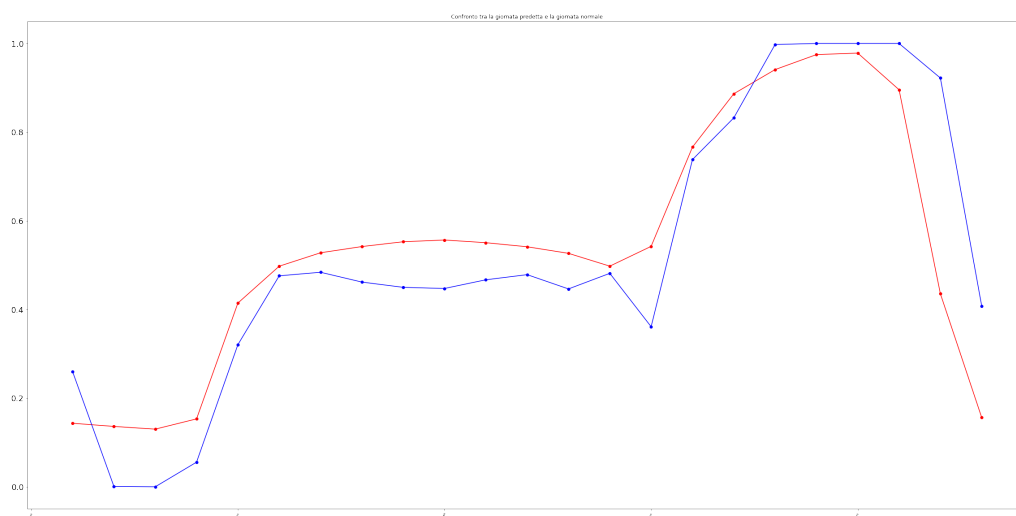


Figure B.1: In blu la giornata normale predetta e in rosso la giornata reale con indice 25.

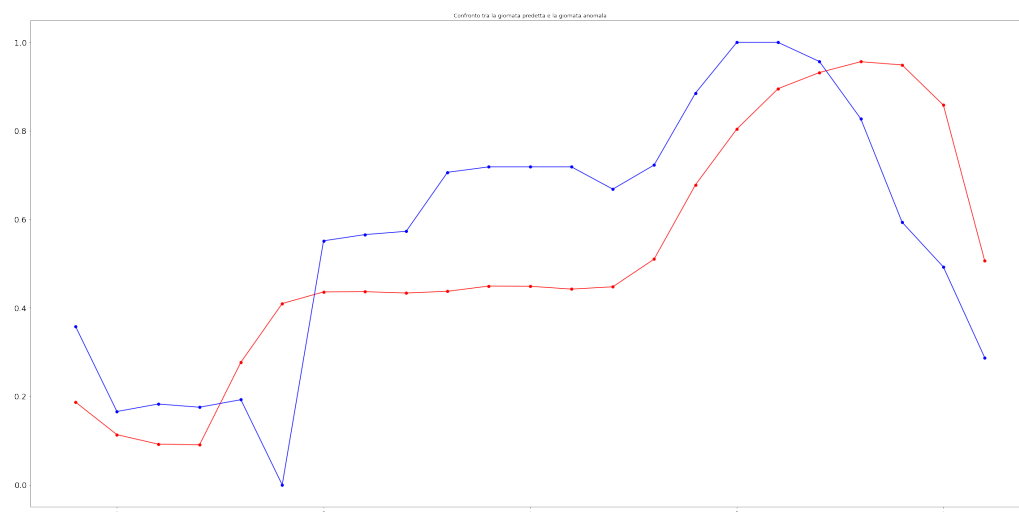


Figure B.2: In rosso la giornata normale predetta e in blu la giornata anomala reale con indice 26.

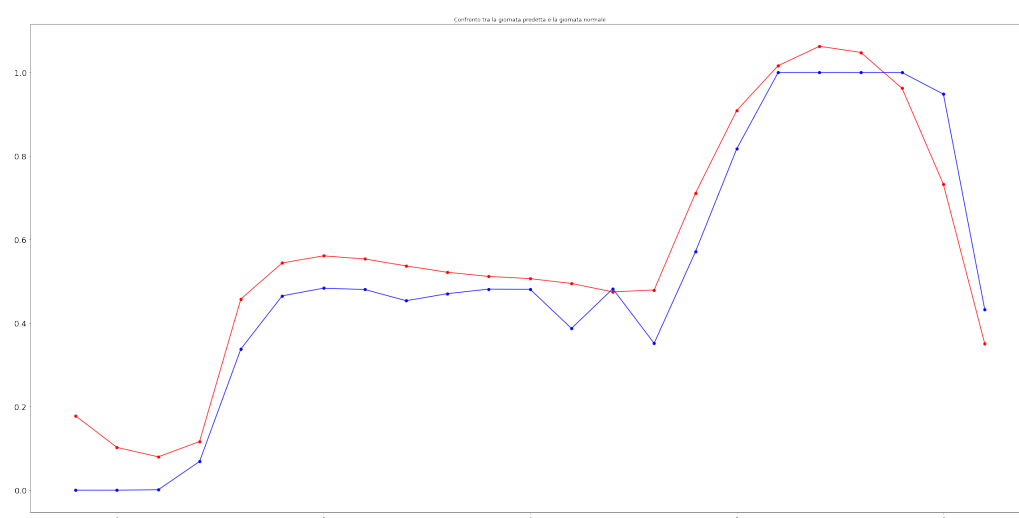


Figure B.3: In blu la giornata normale predetta e in rosso la giornata reale con indice 81.

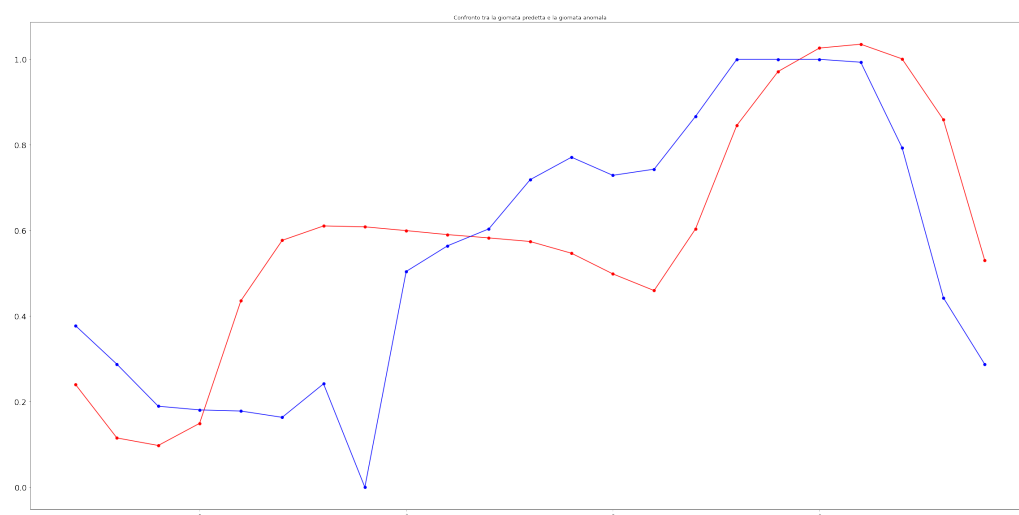


Figure B.4: In rosso la giornata normale predetta e in blu la giornata anomala reale con indice 82.

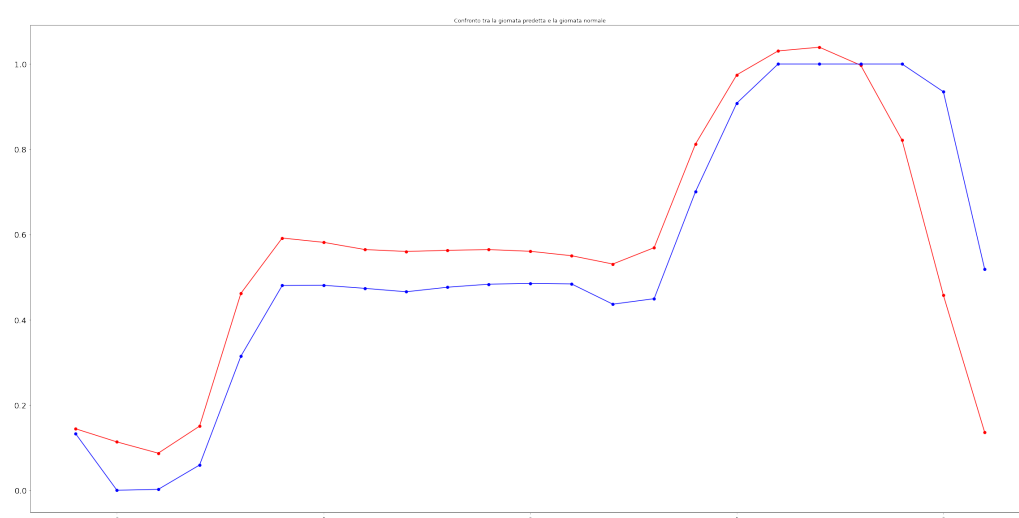


Figure B.5: In blu la giornata normale predetta e in rosso la giornata reale con indice 141.

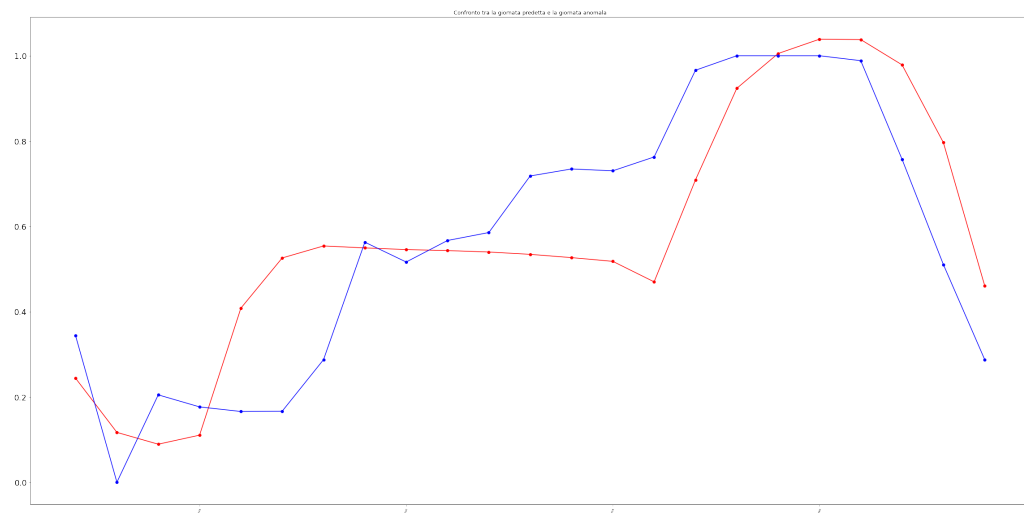


Figure B.6: In rosso la giornata normale predetta e in blu la giornata anomala reale con indice 142.

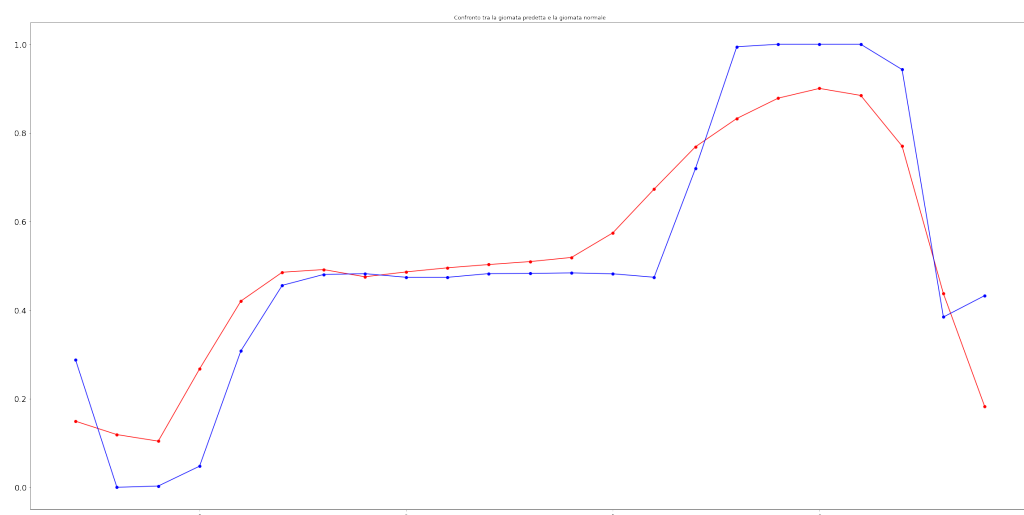


Figure B.7: In blu la giornata normale predetta e in rosso la giornata reale con indice 152.

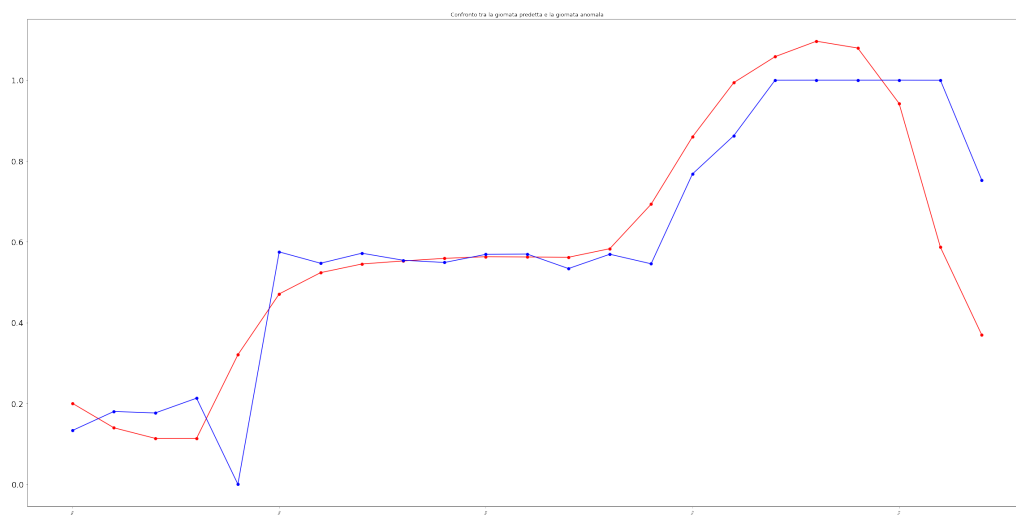


Figure B.8: In rosso la giornata normale predetta e in blu la giornata anomala reale con indice 153.

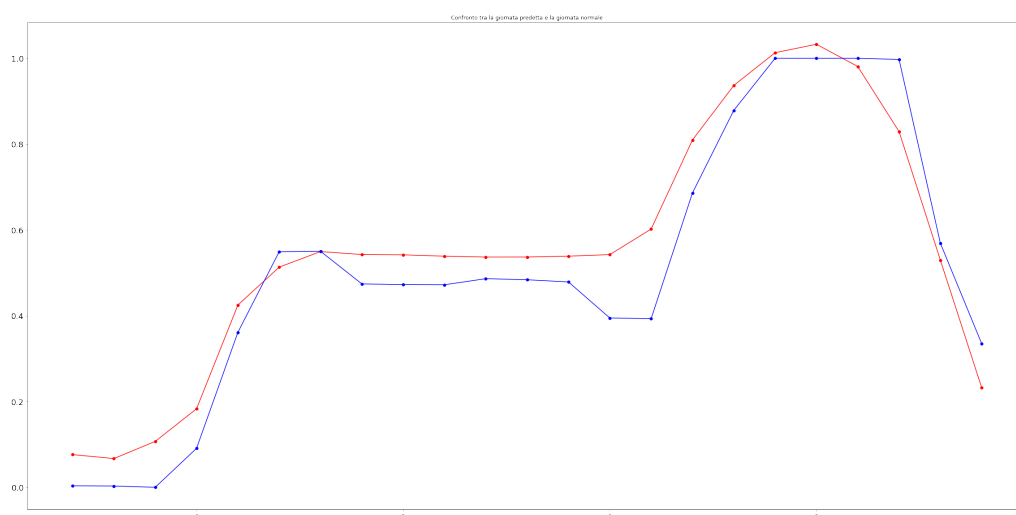


Figure B.9: In blu la giornata normale predetta e in rosso la giornata reale con indice 187.



Figure B.10: In rosso la giornata normale predetta e in blu la giornata anomala reale con indice 188.



Figure B.11: In blu la giornata normale predetta e in rosso la giornata reale con indice 204.

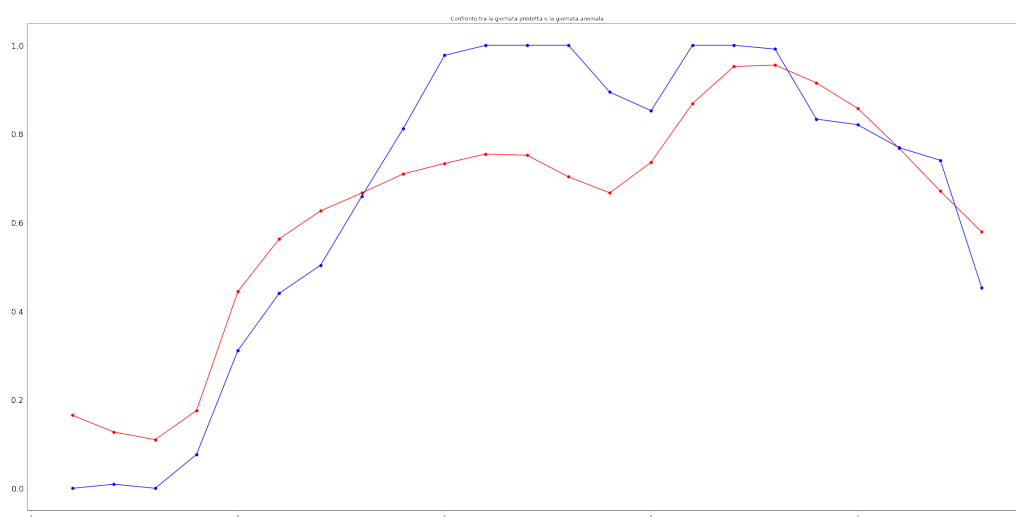


Figure B.12: In rosso la giornata normale predetta e in blu la giornata anomala reale con indice 205.