UNIVERSITÀ DEGLI STUDI DI GENOVA

Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi

Corso di Laurea Triennale in Informatica
Anno Accademico 2021/2022

Analisi di serie temporali

Aspetti Applicativi

Candidato
Alex Valle

Relatore
Prof. Francesca Odone

Indice

1 Introduzione

In tale capitolo introduttivo verranno spiegati gli obbiettivi e la suddivisione del tirocinio in merito al tempo disponibile, cercando di spiegare, in maniera sintetica, come è stato svolto ed organizzato.

1.1 Premessa

Lo scopo di questo elaborato è quello di descrivere l'esperienza di tirocinio svoltasi presso l'Università Degli Studi di Genova con durata complessiva di 300 ore con inizio 25 novembre 2022 e fine 27 febbraio 2023. Il tirocinio è stato svolto per la maggior parte del tempo da remoto con incontri, volti all'andamento di esso, mediante l'utilizzo di Teams (piattaforma sviluppata da Microsoft) che in presenza presso il dipartimento.

1.2 Obbiettivi

L'obbiettivo di questo tirocinio è stato quello di provare a individuare, se esistenti, uno o più metodi, relativi all'analisi di serie temporali, che permettano l'analisi di una serie di dati relative a soggeti sani e petologici con lo scopo di analizzare il cammino.
I dati utilizzati sono stati analizzati precedentemente da un gruppo di ricerca del dipartimento con tecniche differenti da quelle utilizzate durante il tirocinio quindi in caso di un metodo sicuro e soddisfacente all'analisi, i risultati ottenuti sarebbero serviti al gruppo di ricerca come un'ulteriore conferma delle analisi da loro eseguite.

1.3 Tecnologie utilizzate

Come scelta tecnologica principale per lo sviluppo, l'intero svolgimento del tirocinio, si è basato sul linguaggio di programmazione python per la sua semplicità, velocità nella scrittura di codice e ampia community che fornisce molti dei pacchetti utilizzati per eseguire analisi di ogni genere.
Per tenere traccia del lavoro e per esplorare velocemente le modifiche eseguite è stato utilizzato Git come VCS (version control system) e Github come servizio di hosting per i repository.

1.4 Suddivisione del lavoro

Per poter garantire la conoscienza necessaria allo sviluppo di un metodo relativo allo scopo del tirocinio, il lavoro è stato principalmente suddiviso in due fasi: studio dei metodi relativi all'analisi di serie temporali ed effettivo sviluppo, e ricerca, di un metodo per l'analisi del problema posto.

Nella prima fase sono stati studiati i metodi ed applicazioni di tecniche volte allo studio di serie temporali, sia da un lato pratico che da un lato teorico. Per quanto riguarda il lato pratico, queste tecniche sono state studiante mediante la ricerca di articoli e videotutorial online sull'applicazione di esse e poi, in una fase successiva, messe in pratica con piccoli esempi mediante l'utilizzo di dataset di vario genere, forniti in maniera gratuita da siti web trovati su internet, per poterne capire meglio il funzionamento.
Per quanto riguarda il lato teorico di esse è stato necessario studiare una piccola base di statistica inferenziale ed altre nozioni generali per interpretare al meglio i risultati e le tecniche utilizzate in ambito applicativo.

Nella seconda fase si è passati alla ricerca di un metodo, che utilizzi tecniche dell'analisi di serie temporali, per risolvere il problema richiesto. Prima di essere passare allo sviluppo vero e proprio, essendo che i dati forniti erano in uno stato "grezzo", è stato necessario applicare tutte le tecniche di manipolazione dei dati come filtraggio, rinomina delle colonne, tecniche per la sostituzione di valori nulli etc ... per poter ottenere un dataset pulito e lavorabile dal punto di vista applicativo.

2 Studio dei metodi e delle applicazioni

In tale capitolo verranno spiegati i metodi, le applicazioni ed i concetti generali studiati durante la prima fase del tirocinio, accostando ad ogni di essi una relativa implementazione e/o utilizzo.
In primo luogo si troverà una definizione di serie temporale, seguita da una spiegazione dei metodi per manipolare i dati inerenti a serie temporali su python, le componenti principali di una serie temporale e la loro visualizzazione ed infine metodi per l'analisi di essi.

Molti degli esempi forniti in questa sezione fanno riferimento a dataset disponibili al sito "UCI Machine Learning Repository" [1] più precisamente, come esempio, è stato utilizzato un dataset relativo alla qualità dell'aria della città di Beijing [2] (Pechino).

2.1 Cos'è una serie temporale

In statistica descrittiva, una serie storica (o temporale) si definisce come un insieme di variabili casuali ordinate rispetto al tempo, ed esprime la dinamica di un certo fenomeno nel tempo. Le serie storiche vengono studiate sia per interpretare un fenomeno, individuando componenti di trend, di ciclicità, di stagionalità e/o di accidentalità, sia per prevedere il suo andamento futuro [3]. In altre parole una serie storica (o temporale) è un'insieme/serie di dati capionati ed indicizzati nel tempo ad intervalli regolari come ore, giorni o anni.

In termini più matematici: indichiamo con $\boldsymbol{Y}$ il fenomeno (ad esempio il prezzo della benzina dall'anno 1970 all'anno 2010) ed indichiamo con $\boldsymbol{Y_t}$ un'ossevazione al tempo $\boldsymbol{t}$, con $\boldsymbol{t}$ un numero intero compreso tra $\mathbf{1}$ a $\boldsymbol{T}$, dove $\boldsymbol{T}$ è il numero totale degli intervalli o periodi. Una serie temporale viene espressa in questa maniera $\boldsymbol{Y_t} = \{\boldsymbol{Y_1}, \boldsymbol{Y_2}, \ldots, \boldsymbol{Y_T}\}$.

Esempio (*Prezzo della benzina*)**.** Se consideriamo come fenomeno $\boldsymbol{Y}$ il prezzo della benzina dal 1970 al 2010 avremmo come numero totale di osservazioni (o numero totale di periodi) $\boldsymbol{T} = \mathbf{40}$ dove:

- $\boldsymbol{Y_1}$: prezzo della benzina all'anno 1970
- $\boldsymbol{Y_2}$: prezzo della benzina all'anno 1971
- $\boldsymbol{Y_T} = \boldsymbol{Y_{40}}$: prezzo della benzina all'anno 2010

2.2 Metodi per la gestione dei dati

In questo sottocapitolo verranno spiegati i metodi studiati ed utilizzati per la gestione dei dati in python, più nello specifico: come caricare un dataset, una possibile rinomina delle colonne ralative alle serie per una maggiore comprensione, scelta di un indice, individuazione dei valori nulli ed infine una sezione relativa al filtraggio.

Cosa viene inteso per dataset Per dataset si intende un insieme di serie (nella nostra applicazione temporali) relative ad un'unica applicazione.

Esempio (*Qualità dell'aria*)**.** Consideriamo, per esempio, come applicazione le misurazioni di diversi parametri relativi alla qualità dell'aria di Genova: livello di CO_2, livello di SO_2, livello di NO_2 e temperatura in $^\circ\mathsf{C}$. In un dataset possiamo considerare ogni parametro come una serie temporale diversa ma indicizzata nel tempo in ugual maniera, quindi se queste misurazioni avvengono ogni ora avemo per ogni istante di tempo t le misurazioni per ogni parametro in quell'istante.

- Y_1: livello di CO_2, SO_2, NO_2 e temperatura in $^\circ\mathsf{C}$ all'istante 1.
- Y_2: livello di CO_2, SO_2, NO_2 e temperatura in $^\circ\mathsf{C}$ all'istante 2.
- ...
- Y_T: livello di CO_2, SO_2, NO_2 e temperatura in $^\circ\mathsf{C}$ all'istante T.

Ogni parametro in un dataset viene rappresentato come una colonna.

Da qusto momento in poi, nel report, quando si parlerà di dataset varrà inteso il tipo `pandas.DataFrame`, cioè il modo in cui un dataset viene intrpretato all'interno di python, mentre per serie si intenderà il tipo `pandas.Series` oppure un semplice tipo `list` di python.

2.2.1 Pacchetti utilizzati

Per effettuare tutte le manovre relative all'elaborazione e manipolazione dei dati sono state utilizzate funzionalità fornite da pacchetti python come `pandas` e `numpy`, essi semplificano la scrittura di codice e velocizzano il tempo di sviluppo organizzando in maniera ottimale i dati. Per essere utilizzati essi necessitano prima di essere installati tramite il gestore di pacchetti di python `pip`.

Snippet per l'installazione dei pacchetti

```
pip install pandas
pip install numpy
```

Snippet per il caricamento in python

```
import pandas as pd
import numpy as np
```

2.2.2 Caricamento di un dataset

Per poter utilizzare i dati all'interno di python è stata utilizzata la funzionalità di pandas `read_csv` dove, ogni colonna fa riferimento ad una serie.

Snippet

```
dataset = pd.read_csv('data.csv')
```

2.2.3 Rinomina delle colonne relative alle serie

In qualche case, il primo passo da eseguire, è quello di rinominare le colonne relative ad ogni serie così da poter avere una rappresentazione più accurata del dataset. Tramite l'utilizzo della funzione `display` e del metodo `head` del dataset possiamo controllare i primi 5 valori di un dataset controllando anche così il nome di ogni colonna.

Snippet

```
display(dataset.head())
```

	NaN	NaN.1	NaN.2	NaN.3	NaN.4	NaN.5	NaN.6	NaN.7	NaN.8	NaN.9	NaN.10	NaN.11	NaN.12	NaN.13	NaN.14	NaN.15	NaN.16	NaN.17
0	1	2013	3	1	0	6.0	6.0	4.0	8.0	300.0	81.0	-0.5	1024.5	-21.4	0.0	NNW	5.7	Tiantan
1	2	2013	3	1	1	6.0	29.0	5.0	9.0	300.0	80.0	-0.7	1025.1	-22.1	0.0	NW	3.9	Tiantan
2	3	2013	3	1	2	6.0	6.0	4.0	12.0	300.0	75.0	-1.2	1025.3	-24.6	0.0	NNW	5.3	Tiantan
3	4	2013	3	1	3	6.0	6.0	4.0	12.0	300.0	74.0	-1.4	1026.2	-25.5	0.0	N	4.9	Tiantan
4	5	2013	3	1	4	5.0	5.0	7.0	15.0	400.0	70.0	-1.9	1027.1	-24.5	0.0	NNW	3.2	Tiantan

Figure 1: output del metodo head prima della rinomina delle colonne

Come si può notare nell'immagine 1 i nomi delle colonne non hanno nessun nome significativo, con il seguente esempio potremmo cambiare il nome delle colonne.

Snippet

```
# definizione di una lista di nomi per le colonne del dataset
dataset.columns = [ 'no',    'year',  'month', 'day', 'hour',
                    'pm2_5', 'pm10',  'so2',   'no2', 'co',
                    'o3',    'temp',  'pres',  'dewp','rain',
                    'wd',    'wspm',  'station']
display(dataset.head())
```

	no	year	month	day	hour	pm2_5	pm10	so2	no2	co	o3	temp	pres	dewp	rain	wd	wspm	station
0	1	2013	3	1	0	6.0	6.0	4.0	8.0	300.0	81.0	-0.5	1024.5	-21.4	0.0	NNW	5.7	Tiantan
1	2	2013	3	1	1	6.0	29.0	5.0	9.0	300.0	80.0	-0.7	1025.1	-22.1	0.0	NW	3.9	Tiantan
2	3	2013	3	1	2	6.0	6.0	4.0	12.0	300.0	75.0	-1.2	1025.3	-24.6	0.0	NNW	5.3	Tiantan
3	4	2013	3	1	3	6.0	6.0	4.0	12.0	300.0	74.0	-1.4	1026.2	-25.5	0.0	N	4.9	Tiantan
4	5	2013	3	1	4	5.0	5.0	7.0	15.0	400.0	70.0	-1.9	1027.1	-24.5	0.0	NNW	3.2	Tiantan

Figure 2: output del metodo head dopo la rinomina delle colonne

Come si può notare nell'immagine 2 assegnando la lista delle colonne come attuali nomi per le serie del dataset riusciamo ad ottenere un'interpretazione più accurata.

2.2.4 Scelta dell'indice

Molte delle funzionalità fornite da `pandas` ed altri pacchetti python richiedono che il dataset sia indicizzato nel tempo nel corretto modo. In figura 2 si può notare che la prima colonna senza nome e la seconda colonna con nome `no`, indichino il numero di riga per ogni misurazione, la differenza è che la prima colonna è generata automaticamente dal pacchetto `pandas`, ed impostata di default come indice, mentre la seconda con nome `no` è fornita direttamente dal file csv precedentemente caricato. Per l'analisi della maggior parte delle serie temporali un'idicizzazione per numero di riga non è significativa, sarebbe molto più conveniente lavorare avendo come indice di tabella la data ed ora di ogni effettiva misurazione. A questo proposito il dataset caricato ci fornisce delle colonne (`year` , `month`, `day` e `hour`) indicizzate per tempo

e relative ad ogni misurazione, che possono essere riformattate insieme ed usate come indice per il dataset.

Snippet

```
# unificazione delle colonne relative al tempo per ogni
# istante di tempo t in una nuova colonna
new_index_column = []
for i in range(len(dataset.year)):
new_index_column.append("%s/%s/%s %s:0:0"
    % (dataset.day[i], dataset.month[i],
       dataset.year[i], dataset.hour[i]) )

# elimina le colonne relative al tempo
del dataset["year"], dataset["month"],
    dataset["day"], dataset["hour"],
    dataset["no"]

# imposta/crea la nuova colonna date e converti in datetime
dataset['date'] = new_index_column
dataset['date'] = pd.to_datetime(dataset.date, dayfirst=True)

# imposta come index la nuova colonna date
dataset.set_index("date", inplace=True)
display(dataset.head())
```

date	pm2_5	pm10	so2	no2	co	o3	temp	pres	dewp	rain	wd	wspm	station
2013-03-01 00:00:00	6.0	6.0	4.0	8.0	300.0	81.0	-0.5	1024.5	-21.4	0.0	NNW	5.7	Tiantan
2013-03-01 01:00:00	6.0	29.0	5.0	9.0	300.0	80.0	-0.7	1025.1	-22.1	0.0	NW	3.9	Tiantan
2013-03-01 02:00:00	6.0	6.0	4.0	12.0	300.0	75.0	-1.2	1025.3	-24.6	0.0	NNW	5.3	Tiantan
2013-03-01 03:00:00	6.0	6.0	4.0	12.0	300.0	74.0	-1.4	1026.2	-25.5	0.0	N	4.9	Tiantan
2013-03-01 04:00:00	5.0	5.0	7.0	15.0	400.0	70.0	-1.9	1027.1	-24.5	0.0	NNW	3.2	Tiantan

Figure 3: output del metodo head dopo aver impostato l'indice

Come si può notare dall'output del metodo `head` nell'immagine 3 `date` è stato impostato come indice di tabella e quindi da questo momento in poi possiamo accedere al dataset, scegliendo le misurazioni interessate, utilizzando la data.

Periodo di campionamento Un'altra importante modifica è impostare il periodo di campionamento del dataset, in poche parole impostare un corretto indice non basta a massimizzare il corretto funzionamento delle funzionalità di analisi delle serie temporali, bisogna anche specificare l'istante di tempo che occorre tra una misurazione e l'altra. Per fare ciò `pandas` fornise un metodo che imposta il periodo di campionamento a quello deiderato, nel nostro caso sappiamo che le misurazioni sono state campionate ogni ora.

Snippet

```
# imposta il periodo di capionamento
# del dataset ad ogni ora
dataset = dataset.asfreq("h")
```

2.2.5 Individuazione dei valori nulli e possibili soluzioni

La presenza di valori nulli in una serie può avere molteplici cause, ad esempio, l'impossibilità da parte dello strumento di capionare ad un certo istante di tempo t, oppure se pensiamo ad una fotocamera che acquisisce delle coordinate relative ad un soggeto, l'usicta di esso dall'obbiettivo.

Indifferentemente dal motivo per cui dei valori nulli sono presenti, in una serie o un dataset, la loro presenza può causare molti problemi sia nel corretto funzionamento di alcune funzionalità per l'analisi sia perchè non avere dei valori in determinati punti della serie, in certe applicazioni, potrebbe essere un problema. `pandas` fornisce dei metodi utili, e semplici, alla soluzione di questo problema ma ovviamente ogni problema è diverso e quindi, per applicazioni specifiche, potrebbe essere necessario cercare soluzioni differenti. In questa sezione ci limitiamo a descrivere le funzionalità fornite da `pandas` per la soluzione a questo problema.

Controllo Per controllare la presenza di valori nulli `pandas` fornisce un metodo chiamato `isna` che ritorna un dataset dove, per ogni misurazione, indica `True` se la misurazione è `NaN` altrimenti `False`. Sommando i valori `True` per ogni colonna possiamo controllare quanti valori nulli sono presenti per ogni serie.

Snippet

```
# controllo valori nulli
dataset.isna().sum()
```

```
pm2_5       677
pm10        597
so2        1118
no2         744
co         1126
o3          843
temp         20
pres         20
dewp         20
rain         20
wd           78
wspm         14
station       0
dtype: int64
```

Figure 4: output della somma dei valori nulli relativi ad ogni serie del dataset

Come si puo notare dall'output del comando in figura 4 il nostro dataset contiene molteplici valori nulli, ora vediamo come poter risolvere il segente problema

Back fill o Forward fill `pandas` fornisce la possibilità di "riempire" i valori nulli in due diverse modalità tramite l'utilizzo del metodo `fillna`

- **Back fill**: permette di sostituire i valori nulli con la succesiva misurazione valida.
- **Forward fill**: permette di sostituire i valori nulli propagando l'ultima valida misurazione alla prossima valida.

Entrambi i metodi gestiscono i valori nulli più o meno nella stessa maniera ma la scelta di uno piuttosto che l'altro cambia da caso in caso dipendendtemente dal risultato ottenuto dopo l'utilizzo di essi.

Per i nostri esempi il risulato nell'utilizzo di un metodo piuttosto che l'altro portava comunque ad un risultato soddisfacente.

Snippet

```
# rimepimento dei valori nulli
# mediante il metodo di forward fill
dataset = dataset.fillna(method="ffill")
dataset.isna().sum()
```

```
pm2_5      0
pm10       0
so2        0
no2        0
co         0
o3         0
temp       0
pres       0
dewp       0
rain       0
wd         0
wspm       0
station    0
dtype: int64
```

Figure 5: output della somma dei valori nulli relativi ad ogni serie del dataset dopo l'utilizzo del metodo `fillna`.

Interpolazione Un possibile metodo, non fornito dalle funzionalità del pacchetto `pandas`, che potrebbe essere utilizzato è quello di interpolare i dati così da poter colmare i vuoti creati dai valori nulli. Questa funzionalità non è stata sviluppata in quanto non fine allo scopo di questo tirocinio ma, in certe applicazioni, si potrebbe voler utilizzare una metodologia basata su questa tecnica per ottenere una rappresentazione più accurata dei dati.

Altro metodo non convenzionale Un altro metodo non convenzionale, non presente tra le funzionalità fornite, è quello di poter eliminare i valori nulli dalle serie semplicemente eliminandoli. Ovviamente questo concetto di eliminare una misurazione nulla va contro a tutte le premesse fatte fino ad ora, non avendo così una "reale" serie temporale poichè mancherebbe una misurazione ad un determinato istante di tempo t, e molte delle analisi che si vorrebbero poter fare su una serie risulterebbero inapplicabili. Tuttavia, in qualche applicazione particolare (come vedremo in seguito), una funzionalità che semplicemente elimina i valori nulli da una serie potrebbe tornare comoda.

Snippet

```
import math as math # import del pacchetto math

def delete_nan(series: pd.Series | list):
""" emlimina i valori nulla da una serie
"""
new_series = []
for idx, value in enumerate(series):
    if not math.isnan(value):
        new_series.append(value)
return new_series
```

2.2.6 Filtraggio

2.3 Componenti di una serie temporale

2.3.1 Decomposizione di una serie

2.3.2 Trend

2.3.3 Stagionalità

2.3.4 Residui

2.4 Stazionarietà

2.4.1 Dickey Fuller Test

2.5 Autocorrelazione e Autocorrelazione parziale

2.5.1 ACF

2.5.2 PACF

3 Bibliografia

Bibliografia

[1] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

[2] Zhang S. Guo B. Dong A. He J. Xu Z. Chen S.X. *Cautionary Tales on Air-Quality Improvement in Beijing*. 2017. URL: https://archive.ics.uci.edu/ml/datasets/Beijing+Multi-Site+Air-Quality+Data.

[3] Wikipedia. *Serie Storica — Wikipedia, L'enciclopedia libera*. [Online; controllata il 30-gennaio-2022]. 2022. URL: https://it.wikipedia.org/wiki/Serie_storica.