Progetto Data and Document Mining

Alessandro Mini
alessandro.mini1@stud.unifi.it
Mat: 7060381

Abstract

In questo progetto si andranno ad implementare parti dell'articolo di Darmawan Utomo e Pao-Ann Hsiung in cui viene proposta un architettura hardware e software per prevedere anomalie nel consumo elettrico di strutture (abitazioni) in un certo lasso di tempo nel futuro.

L'articolo può quindi essere diviso in due sezioni, nella prima si eseguono tecniche di preprocessing e clustering basate su Kmeans, nella seconda si usano algoritmi di previsione per prevedere il consumo elettrico.

Gli autori analizzano quindi una possibile implementazione full stack utilizzando come dispositivo edge un raspberry PI.

1. Dataset e implementazione

In questo progetto si implementeranno parti dell' articolo[2] iniziando dal *preprocessing* e *clustering*.

Il progetto verrà eseguito su un *dataset* di consumi elettrici rilevati da 200 abitazioni negli USA e presentati come una griglia di 52562 righe per 200 colonne.

Le colonne rappresentano le case *(smart sensors)* mentre le righe rappresentano le serie temporali del consumo elettrico (in KW) di un anno rilevate ogni 30 minuti.

Questo dataset è reperibile all'indirizzo (Url Dataset) ed è in formato .xlsx.

Per ragioni di performance questo è stato convertito in .csv usando un tool automatico (Microsoft Excel).

Verranno quindi implementate alcune sezioni di questo articolo.

2. Pre-processing

Nell'articolo [2] gli autori documentano la fase di *preprocessing* in cui si prende in ingresso il dataset e si restituisce un *label set*[1], un dataset equivalente a quello di partenza ma contenente valori 0/1 secondo il seguente schema:

$$a_{i,j} = \begin{cases} 0, & \text{se NON anomalo} \\ 1, & \text{se anomalo} \end{cases} \quad (1)$$

Ciò si realizza nelle seguenti fasi:

1. Accumulare tutti i rilevamenti per giorno: Si usa una semplice somma per aggregare tutti i rilevamenti giornalieri la nuova tabella sarà quindi 365 righe x 200 colonne.

2. Normalizzare i valori di ogni colonna: Gli autori non hanno specificato il *normalizzatore* scelto, si è usata quindi una semplice normalizzazione con il massimo per riprodurre l'esperimento:

   $$colonna = \frac{colonna}{colonna.max()}$$

   (Dove $colonna$ è un vettore,$colonna.max()$ è un numero)

   Questa normalizzazione produce gli stessi risultati dell'articolo.

3. Labeling delle anomalie: Per effettuare il labeling gli autori utilizzano una regola euristica [3], il rilevamento r di un certo giorno d è ritenuto anomalo se $r > \delta$ dove $\delta = 3 \cdot \sigma + \mu$ in cui:

   - σ è la deviazione standard della serie temporale di quel sensore (rilevamenti di tutto l'anno).
   - μ è la media della serie.

   Applicare questa regola equivale a richiedere che sia anomalo ciò che "cade al di fuori" del range 0.997.

4. Costruzione del label-set : Si crea un dataset clone discretizzato che contiene valori 0/1 (non anomalo, anomalo), si applica la regola del passo 3

5. Espansione (look-ahead) : Si estendono le anomalie, se un giorno presenta un anomalia allora questa viene estesa a la giorni dove la è un valore di "Look ahead". Nell'articolo si usano valori di $la = 7, 14$.

Le parti studiate verranno implementate in Python con suite Anaconda.

2.1. Implementazione Python

La fase di preprocessing è stata implementata con le seguenti istruzioni:

```
df = import_normalize_dataset("dataset path")
df = sort_dataset_by_date(df)
df = df.T
anomalies =
↪  mark_and_extend_anomalies(df,look_ahead)
```

2.1.1 Import e normalizzazione

La funzione import_normalize_dataset(path) prende in ingresso il percorso del dataset nel sistema operativo ed importa, aggrega e normalizza il dataset (esegue la normalizzazione con il massimo).

L'aggregazione dei dati consiste nel sommare tutti i dati giornalieri e unirli in un unico valore per giorno riducendo così la dimensione del dataset a 365 righe.

Questo viene realizzato implementando in python una *pivoting table* come segue:

```
def import_normalize_dataset(path):
    """
    import_normalize_dataset(path)
    This method import and normalize the base dataset.
    """
    df = pd.read_csv(path,sep=";")
    df["Time"] = pd.to_datetime(df["Time"])
    df['Time'] = df['Time'].dt.strftime('%d/%m/%Y')
    aggregated_df =
    ↪  df.groupby('Time')[list(df.columns)[1:]].sum()
    aggregated_df.sort_values(by="Time")
    aggregated_df = (aggregated_df-aggregated_df.min())/
    (aggregated_df.max()-aggregated_df.min())
    for col in aggregated_df:
        df[col] = df[col]/df[col].max()
    return aggregated_df
```

Il dataset viene poi *ordinato per righe* attraverso l'uso della funzione "sort_dataset_by_date(dataset)":

```
def sort_dataset_by_date(df):
    """
    sort_dataset_by_date(df):
    This method sorts a dataset by the "Time" field
    """
    df.reset_index(inplace=True)
    df['Time']=pd.to_datetime(df['Time'])
    df = df.sort_values(by="Time")
    df = df.set_index("Time")
    return df
```

Questo step è importante poiché nei *plot* le serie sono ordinate per data.

Prima del passo successivo il dataset viene *trasposto* così da rendere più facile il lavoro (e la visualizzazione) delle serie temporali lunghe 365 giorni dei 200 sensori.

Ottengo quindi un dataset trasposto di 200 righe e 365 colonne che è stato aggregato, normalizzato e ordinato per data sulle colonne, as esempio:

	01/01/2010	02/01/2010	..
Sensore 0	$val_{0,1}$	$val_{0,1}$	...
Sensore 1	$val_{1,0}$	$val_{1,1}$	...
Sensore 2	$val_{2,0}$	$val_{2,1}$	...
Sensore 3	$val_{3,0}$	$val_{3,1}$	
...	...	...	...

2.1.2 Labeling delle anomalie

Lo step successivo è quello di eseguire il labeling delle anomalie, per questo si usa la funzione "mark_and_extend_anomalies(df,look_ahead)" la quale prende in ingresso un dataset e il valore di look ahead e restituisce un nuovo dataset copia del primo che è stato *binarizzato ed esteso*, ovvero contiene solo valori 0/1 secondo lo schema non anomalo/anomalo.

Nel caso in cui sia rilevata un anomalia questa viene estesa per la (il valore di *look-ahead*) giorni.

La funzione è stata implementata come segue:

```
def mark_and_extend_anomalies(df_base,look_ahead):
    """
    mark_and_extend_anomalies(df_base,look_ahead):
    This method find and extends the anomalies given a
↪   lookahead.
    it returns a new dataset in order to use multiple
↪   versions.
    """
    df = df_base.copy(deep=True)
    for index, row in df.iterrows():
        mean = row.mean()
        std = row.std()
        for idx, item in enumerate(row):
            if(item>=(mean+3*std)):
                row[idx] = 1
            else:
                row[idx]= 0

    for row in df.iterrows():
        i = 0
        riga = row[1]
        while(i<len(riga)-1):
            if(riga[i]==1):
                for i in range(i,i+look_ahead):
                    try:
                        riga[i] = 1
                    except:
                        pass
            i = i + 1

    return df
```

In output si ha il dataset che viene generato da una *deep copy* di quello di partenza, questo perché per ragioni di test e di plot potrebbe essere utile avere diversi dataset con diversi valori di look ahead.

2.2. Plot

Gli autori nell'articolo [2] presentano un *heatmap* delle anomalie nel dataset, per ragioni di test (al fine di verificare che si generino le *stesse* heatmaps ho utilizzato seaborn per visualizzare i *label set*, ottenendo i risultati di 1, 2.

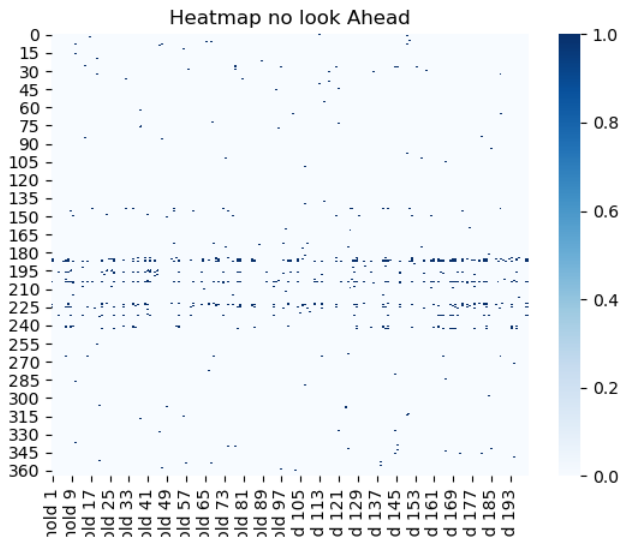

Figure 1. Heatmap senzalook-ahead

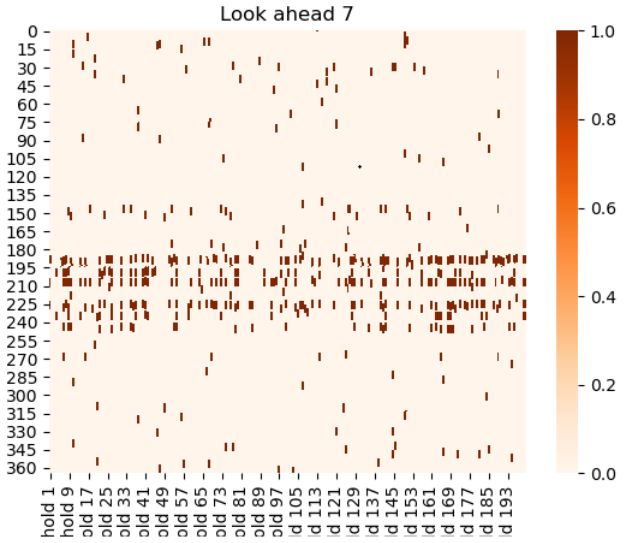

Figure 2. Heatmap con 7 giorni di look-ahead

Che corrispondono con quelle dell' articolo [2, p. 9].

3. Clustering

Gli autori procedono poi ad eseguire il *clustering*[2, p. 14] del dataset utilizzando un algoritmo basato su K-Means eseguendo i seguenti step:

1. Determinazione del *best fitting* k.
2. Clustering delle case (smart sensors) in k clusters.
3. Riassunto del risultato del processo *(Summary)*.

Il valore di k migliore è $k = 2$, è già stato determinato dagli autori attraverso una procedura iterativa sul *silohuette score* dei clusters percui sarà assunto a priori come valore ottimale.

3.1. Implementazione Python

Molte delle librerie principali presenti in python offrono un implementazione di KMeans tra cui si ricorda tslearn e sklearn, tuttavia soltanto tslearn permette di eseguire clustering su serie temporali utilizzando metriche diverse.

La libreria "tslearn.clustering.TimeSeriesKMeans" non sarà più supportata e verrà rimossa dalla versione 0.24 (versione corrente 0.22), per questo motivo ho optato per un implementazione "da 0".

In questa implementazione ottimizzata per le serie temporali sarà possibile utilizzare le due metriche di distanza che erano previste in tslearn ("dtw" e "euclid").

L'algoritmo implementato è il seguente:

3.1.1 Descrizione Algoritmo

1. Si scelgono i valori di k sensori che diventeranno centroidi in modo randomico (es. le serie temporali del sensore 5 e 65 diventeranno centroidi).
2. Si inizializzano k clusters vuoti, dopodichè:
3. Per ogni iterazione:
   (a) Rimuovo i sensori dai cluster.
   (b) Assegno tutti i sensori (serie temporali) ai rispettivi best fitting clusters.
   (c) Ricalcolo il centro di massa dei clusters (quindi i centroidi).

Alla fine dell'esecuzione ottengo:

1. Un set di k clusters contenenti gli ID delle rispettive case del dataset originale.
2. Un set di k centroidi rappresentativi dei clusters.

E' stata realizzata una *classe* python che viene importata nel progetto

```
from KMeans import KMeansClusterizer
```

ed inizializzata/usata:

```
KM = KMeansClusterizer(anomalies,2,10,"dtw")
KM.fit()
clusters = KM.getClusters()
centroids = KM.getCentroids()
```

Il metodo principale è KMeansClusterizer(dataset,num_clusters,itmax,metric), i cui argomenti sono:

- dataset: è il dataset in questione.

- num_clusters: è il numero di clusters attesi (k)
- itmax: è il numero massimo di iterazioni.
- metric: è la metrica (euclid o dtw).

3.1.2 Inizializzazione clusters e centroidi

Viene riportato il codice del costruttore che inizializza i valori delle variabili *locali* e i clusters/centroidi.

```
def __init__(self,anomalies,n,ITMAX,distance_metric):
    """
    __init__(self,anomalies,n,ITMAX,distance_metric):
    Costruttore della classe, prende in ingresso:
        anomalies: dataset labeled.
        n: numero di clusters.
        ITMAX: max iterazioni.
        distance_metric: metrica di distanza.
    """
    self.anomalies = anomalies
    #Inizializzo clusters e centroidi
    clusters = [[]]
    centroids = [[]]
    for i in range(0,n):
        clusters.append([])

        centroids.append(anomalies.iloc
        [random.randint(0,200)].values)
    clusters.pop(0)
    centroids.pop(0)
    self.clusters = clusters
    self.centroids = centroids
    self.ITMAX = ITMAX
    self.distance_metric = distance_metric
```

3.1.3 Assegnamento sensori ai clusters

Viene riportato il codice che assegna ogni sensore (casa) al cluster con il centroide "più vicino" secondo la metrica usata:

```
def k_means_assign_points(self):
    """
    k_means_assign_points(self):
    Metodo che assegna tutti i punti (sensori) al
↪   relativo cluster
    best fitting.
    """
    for i in range(0,self.anomalies.shape[0]):
        riga = self.anomalies.iloc[i]
        #Inizializzo il best fit
        best_fit = 0
        #Inizializzo un valore di distanza
        distance = 10
        #step 1: assegno ognuno al suo cluster
        for j in range(0,len(self.centroids)):
            if self.distance_metric=="euclid":
                distanza_calcolata = np.linalg.norm(riga
                ↪   - self.centroids[j])
            if self.distance_metric == "dtw":
                distanza_calcolata =
                ↪   dtw(riga,self.centroids[j])
            if(distanza_calcolata<distance):
                #Assegno best fit
                best_fit = j
                #Aggiorno la distanza
                distance = distanza_calcolata
        #Aggiungo il punto al cluster best fit.
        self.clusters[best_fit-1].append(i)
```

3.1.4 Ricalcolo dei centroidi

Viene ricalcolato il centroide dei vari cluster, una maggior difficoltà è dovuta al fatto che i "punti" sono serie temporali, quindi è necessario *generalizzare* la formula di ricalcolo del centroide in n dimensioni $n = 365$.

Considerando ad esempio un cluster con m sensori:

$$CL = (s_1, s_2, ...s_m)$$

Con un centroide

$$CD = (c_0, c_1, c_2, ...c_{365})$$

Ricalcolo il centroide come segue:

$$CD_{new} = \frac{(\sum_{i=0}^{m} s(0), \sum_{i=0}^{m} s(1), ..., \sum_{i=0}^{m} s(m))}{m}$$

Volendo fornire un semplice esempio numerico, se si ha un cluster composto da 3 sensori i quali hanno effettuato 4 rilevamenti:

$$C_0 = (1, 0, 1, 0)$$
$$C_1 = (1, 1, 0, 0)$$
$$C_2 = (0, 0, 0, 0)$$

E il relativo centroide calcolato all'iterazione $i - 1$:

$$C^* = 1, 0, 1, 0$$

Ricalcolo le coordinate del centroide al passo i come segue:

$$C^* = \frac{(2, 1, 1, 0)}{3}$$

Da cui

$$C^* = (0.66, 0.33, 0.33, 0)$$

Implementandolo in Python si ottiene la seguente function:

```
def k_means_recalc_centroid(self):
    """
    k_means_recalc_centroid(self):
    Metodo che ricalcola il centroide sulla base del
↪   concetto
    di centro di massa.
    """
    #per ogni cluster
    for i in range(0,len(self.clusters)):
        #prendo la casa contenuta nel cluster
        for j in range(0,len(self.clusters[i])):
            #ricostruisco l'indice del dataset originale
            indice = self.clusters[i][j]
            #prendo il valore dal datset della casa
            ↪   j-esima nel cluster i-esimo
            riga = self.anomalies.iloc[indice]
            #accumulo i valori su ogni campo
            self.centroids[i] =
            ↪   np.add(self.centroids[i],riga)
    #adesso divido ogni cluster per la sua size (secondo
    ↪   formula)
    for k in range(0,len(self.centroids)):
        #size
        size = len(self.clusters[k])
        #se un cluster è vuoto imposto size=1
        if(size==0):
            size=1
        #divisione
        self.centroids[k] =
        ↪   np.divide(self.centroids[k],size)
```

3.1.5 Fit dell'algoritmo

L'algoritmo ha un metodo *fit()* che esegue il processo di *fitting*:

```
def fit(self):
    """
    fit(self):
    Metodo che esegue il fitting.
    """
    for i in range(0,self.ITMAX):
       print("[KMEANS]: Iterazione num. ",i)
       self.clear_clusters()
       self.k_means_assign_points()
       self.k_means_recalc_centroid()
```

Il costo dell'algoritmo è di $\mathcal{O}(m \cdot itmax)$ con m numero di serie temporali da "clusterizzare" e $itmax$ numero massimo di iterazioni.

3.2. Output e plots

Se l'algoritmo viene eseguito $k = 2$, $itmax = 10$ si ottengono risultati leggermente diversi al variare della metrica di distanza utilizzata.

3.2.1 Distanza DTW

Usando come metrica di distanza il Dynamic time Warping:

$$DTW(X,Y) = \sqrt{\sum_{(i,j)\in\pi} \|X_i - Y_j\|^2}$$

Figure 3. Formula per lo score DTW.

Si ottiene il seguente clustering:

Tempo terminazione	13s
Cluster 0	125 case
Anomalie cluster 0	125
cluster 1	75 case
Anomalie cluster 1	25

I cui centroidi hanno la seguente forma:

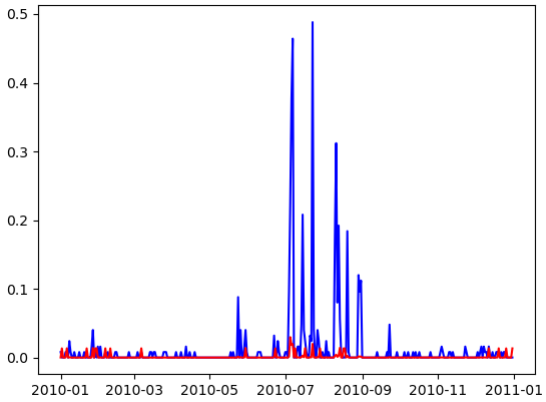

Figure 4. Plot dei centroidi calcolati con distanza "dtw", in blu il profilo "anomalo", in rosso il profilo "normale"

3.2.2 Distanza euclidea

Usando come metrica di distanza quella euclidea:

$$d(x,y) = \sqrt{\sum_{k=1}^{n}(x_k - y_k)^2}$$

Figure 5. Score euclideo

si ottengono i seguenti clusters:

Tempo terminazione	0.7s
Cluster 0	56 case
Anomalie cluster 0	56
cluster 1	144 case
Anomalie cluster 1	94

I cui centroidi hanno forma Figura 6

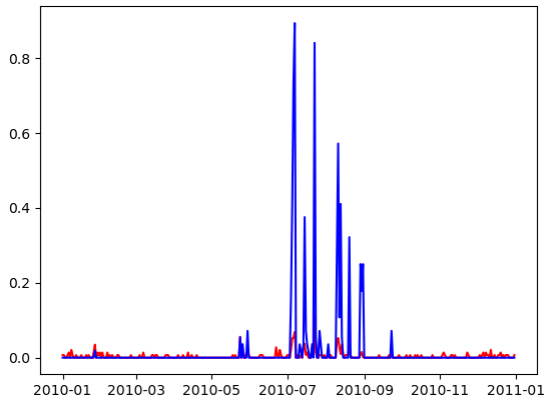

Figure 6. Plot dei centroidi calcolati con distanza ”euclid”, in blu il profilo ”anomalo”, in rosso il profilo ”normale”

3.3. Considerazioni

Tra le metriche di distanza utilizzate la D.T.W è quella che garantisce risultati più vicini a quelli dell'articolo[2] in cui si hanno due clusters da 127 e 73 sensori.

Questa distanza inoltre seleziona un profilo anomalo più definito rispetto alla distanza euclidea.

Possiamo notare una differenza nella velocità di convergenza tra i due metodi.

4. Links e riferimenti

GitHub: Link Github

Dataset: Link Dataset

References

[1] Varun Chandola, Arindam Banerjee, and Vipin Kumar. “Anomaly Detection: A Survey”. In: *ACM Comput. Surv.* 41 (July 2009). DOI: 10.1145/1541880.1541882.

[2] Darmawan Utomo and Pao-Ann Hsiung. “A Multi-tiered Solution for Anomaly Detection in Edge Computing for Smart Meters”. In: *Sensors* 20.18 (2020). ISSN: 1424-8220. DOI: 10.3390/s20185159. URL: https://www.mdpi.com/1424-8220/20/18/5159.

[3] Wikipedia. *68–95–99.7 rule — Wikipedia, The Free Encyclopedia*. 2021.