Progetto di Data Management

Traffico e meteo nella città di Milano: costruzione e analisi di un dataset

Gravina, Greta, mat. 881470
Puccinelli, Niccolò, mat. 881395
Scatassi, Marco, mat. 883823

Università degli studi di Milano-Bicocca
A.A. 2021/2022

Indice

1 Introduzione

L'oggetto di questo lavoro è la costruzione di un dataset concernente le informazioni relative al meteo e al traffico nella città di Milano. Nello specifico, i dati coprono le due settimane comprese tra il 28 dicembre 2021 e il 10 gennaio 2022 (compresi) e fanno riferimento alle due circonvallazioni più interne di Milano: la cerchia dei Navigli e la cerchia dei Bastioni.

Tramite apposite API, l'acquisizione dei dati è avvenuta in tempo reale tramite una macchina virtuale, con una chiamata al servizio ogni dieci minuti, i.e. dalle 00:00 del 28 dicembre 2021 alle 23:50 del 10 gennaio 2022. Successivamente, i dati sono stati opportunamente puliti e le informazioni relative al traffico sono state integrate coi dati del meteo. Il tutto è stato poi memorizzato in un DBMS *document-based*: **MongoDB**. Infine, è stata effettuata un'analisi di qualità, per lo più in termini di *completeness* e *currency*.

Di seguito presentiamo nel dettaglio gli obiettivi perseguiti e il processo di costruzione del dataset, articolato in *data acquisition*, *data integration*, *data storage* e *data quality*.

2 Obiettivi

L'obiettivo principale di questo progetto è la costruzione di un dataset in grado di rispondere a diverse domande di ricerca, relative al traffico e alle condizioni meteorologiche nelle due circonvallazioni più interne di Milano. Nello specifico:

- **Le condizioni meteorologiche influenzano il traffico milanese?**
- Come varia il traffico a seconda delle vie e delle circonvallazioni?
- Quanto influiscono le ZTL sulla circolazione dei veicoli?
- Come varia il traffico a seconda delle ore e dei giorni della settimana?

Poiché i dati più rilevanti sono quelli associati al traffico, abbiamo considerato quest'ultimi come primari, integrandoli successivamente con i dati del meteo. L'integrazione e la gestione dei dati sono avvenute sulla base delle domande di ricerca precedenti, i.e. la creazione, la manipolazione e l'eliminazione delle feature sono state effettuate tenendo conto degli obiettivi perseguiti.

3 Data acquisition

3.1 Metodologia

I dati sono stati acquisiti tramite API GET. Per traffico e meteo sono state rispettivamente impiegate le API di TomTom[1] e OpenWeather[2]. Per OpenWeather è stata sufficiente una chiave di accesso, mentre TomTom pone un limite di 2500 chiamate giornaliere al servizio, motivo per il quale abbiamo richiesto quattro chiavi di accesso, da usare alternatamente.

L'acquisizione dei dati è avvenuta in tempo reale, con una chiamata ai servizi API ogni 10 minuti, a partire dalle 00:00 del 28 dicembre 2021, fino alle 23:50 del 10 gennaio 2022 (compresi). Sono state considerate, in totale, 24 vie, 11 appartenenti alla cerchia dei Navigli e 13 alla cerchia dei Bastioni. Ogni 10 minuti, sono state richieste, per ciascuna via, le informazioni su traffico e meteo ai servizi sopracitati, tramite l'inserimento delle coordinate geografiche specifiche di ogni strada all'interno del metodo GET. Per TomTom in particolare, è stata eseguita una verifica delle coordinate. Inserendo quest'ultime all'interno della chiamata, TomTom restituisce, oltre alle informazioni del traffico, le reali coordinate a cui sono riferite le condizioni stradali, che sono state quindi inserite su *Google Maps* per verificare la correttezza della localizzazione.

Le richieste sono state gestite tramite il metodo *get* della libreria *requests*, mentre il task di acquisizione, da eseguire ogni 10 minuti, è stato implementato tramite *BlockingScheduler* della libreria *apscheduler*. I dati, ottenuti in formato JSON, sono stati scritti in file testuali suddivisi per fonte (TomTom e Openweather) e strada, tramite *append* delle nuove informazioni ogni 10 minuti.

Allo scopo di identificare temporalmente eventuali valori nulli durante la fase di data quality, è stato estrapolato, ad ogni chiamata, anche l'*header* della richiesta del traffico, contenente data e ora di risposta del servizio. Questi sono stati scritti subito dopo il *body* della relativa richiesta. Per il meteo, tale approccio sarebbe risultato superfluo, in quanto l'informazione temporale è già presente all'interno del campo **dt**, sebbene in Unix Time.

I dati del traffico sono tutti compresi nel campo **flowSegmentData**, il secondo campo, **@version** (i.e. versione del software utilizzata dal servizio), è stato invece rimosso in fase di integrazione. Considereremo dunque il solo campo **flowSegmentData** nella successiva descrizione delle features.

[1] https://developer.tomtom.com/traffic-api/api-explorer
[2] https://openweathermap.org/current

3.2 Descrizione dei campi

I dati grezzi presentano la seguente forma:

- Traffico (TomTom):
  - **frc**: Indica il tipo di strada (e.g. FRC3 = strada secondaria).
  - **currentSpeed**: Velocità media attuale nel tratto di strada richiesto.
  - **freeFlowSpeed**: Velocità nel tratto di strada richiesto, in condizioni ideali.
  - **currentTravelTime**: Tempo medio di percorrenza attuale nel tratto di strada richiesto.
  - **freeFlowTravelTime**: Tempo di percorrenza nel tratto di strada richiesto, in condizioni ideali.
  - **confidence**: Indica il livello di confidenza dei dati restituiti (tra 0 e 1).
  - **roadClosure**: Indica se la strada sia chiusa al traffico o meno.
  - **coordinates**: Coordinate del segmento di strada preso in considerazione.
- Meteo (OpenWeather):
  - **coord**: Coordinate geografiche associate alla richiesta.
  - **weather**: Indica la condizione meteorologica generale (e.g. clear = sereno).
  - **base**: Parametro interno.
  - **main**: Contiene informazioni meteorologiche quali temperatura e umidità.
  - **wind**: Contiene informazioni relative al vento.
  - **clouds**: Percentuale di nuvolosità.
  - **rain**: Volume di pioggia caduta nelle ultime ore.
  - **snow**: Volume di neve caduta nelle ultime ore.
  - **dt**: Data e ora di calcolo dei dati, in Unix Time.
  - **sys**: Contiene parametri interni e l'ora di alba e tramonto.
  - **timezone**: Indica la differenza in secondi rispetto a UTC.
  - **id**: ID della città.
  - **name**: Nome della città.
  - **code**: Parametro interno.

4 Data preparation

Per gestire i file testuali ottenuti nella fase di acquisizione e svolgere il processo di data preparation è stato utilizzato il linguaggio di programmazione Python, in particolare le librerie *json*, *pandas*, *numpy*, *re* e *datetime*. Nelle sottosezioni seguenti vengono descritte nel dettaglio le fasi di data cleaning e data integration.

4.1 Data cleaning

Traffic Data Una volta acquisiti, i file contenenti i dati relativi al traffico si presentavano come nell'immagine seguente.

Immagine 1: Preview dati del traffico relativi a viale Papiniano

I file di testo sono stati letti creando due dizionari (uno per ogni circonvallazione). Ogni elemento del dizionario ha come chiave il nome della singola via a cui è riferito e come attributo una lista. Ciascun elemento di quest'ultima è formato da una stringa contenente due JSON, uno relativo al *body* della chiamata API (i.e. le informazioni relative al traffico), l'altro corrispondente all'*header* della stessa (contenente data e orario della risposta). Dal momento in cui, per varie ragioni (e.g. errori interni al server), la risposta a una chiamata API può non essere valida, abbiamo verificato se ciò fosse successo nei dati raccolti. Inoltre, come osservabile in figura 1, le chiavi **date** e l'attributo ad esse associato, a differenza degli altri elementi, sono compresi tra apici singoli (''), rendendo la stringa racchiusa da graffe

({}) che le contiene non leggibile come file JSON. Si è quindi proceduto alla sostituzione con doppie virgolette (""). A questo punto è stato possibile leggere i singoli JSON. Nello specifico, sono stati creati due dizionari per ogni circonvallazione. Ogni elemento del primo dizionario ha come chiavi i nomi delle vie e come valori le liste contenenti i file JSON con le informazioni relative al traffico. Il secondo dizionario ha come chiavi i nomi delle vie e come valori le liste contenenti i JSON con le informazioni relative al tempo di risposta delle singole chiamate API.

Dopo aver integrato i JSON dei due dizionari costruiti (vedi sezione 4.2) sono state convertite le date da UTC a CET, è stato eliminato il campo **@version** dai JSON (inutile ai fini dell'analisi) ed è stato rimosso il campo **coordinates**, sostituito in seguito dal nome della via. Si è poi proceduto alle seguenti modifiche dei JSON:

- Aggiunta del campo **ringRoad** contenente il nome della circonvallazione di appartenenza della via (Bastioni o Navigli);
- Sono stati rinominati il campo **frc** (ora **roadType**) e i corrispondenti attributi:
  - $\mathrm{FRC3}$ con Strada secondaria;
  - $\mathrm{FRC4}$ con Strada di collegamento locale.

  per aumentarne la comprensibilità;
- Aggiunta del campo **LTZ** con valore:
  - True se nell'orario associato al campo **date** del JSON è attiva la zona a traffico limitato;
  - False se nell'orario associato al campo **date** del JSON non è attiva la zona a traffico limitato;

  al fine di introdurre un ulteriore elemento di analisi rilevante;
- Aggiunta del campo **flowConditions** con valore:
  - Flowing, se $0\% \leq \delta < 25\%$;
  - Moderate, se $25\% \leq \delta < 50\%$;
  - Congested,se $50\% \leq \delta < 75\%$;
  - Blocked, se $75\% \leq \delta \leq 100\%$;

dove:

$$\delta = \frac{freeFlowSpeed - currentSpeed}{freeFlowSpeed} \cdot 100 \tag{1}$$

Weather Data Per quanto riguarda i dati relativi al meteo, i file acquisiti si presentavano nella forma seguente:

Immagine 2: Preview dati del meteo relativi a viale Papiniano

La lettura del file è stata eseguita in maniera analoga ai dati di traffico, così come il controllo della presenza di eventuali richieste invalide. In particolare, di quest'ultime ne sono state individuate due, poi eliminate. Le altre modifiche effettuate ai JSON sono state le seguenti:

- Sostituzione del campo **coor** con il campo **street** contenente il nome della via;
- Conversione dei campi **sunrise**, **sunset** e **dt** nello stesso formato di data usato per il traffico;
- Aggiunta del campo **ringRoad** contenente il nome della circonvallazione di appartenenza della via (Bastioni o Navigli);
- Eliminazione dei campi non rilevanti ai fini dell'analisi o ridondanti e riorganizzazione dei JSON annidati.

4.2 Data integration

Dopo aver letto i file testuali in formato JSON, come già descritto, si sono ottenuti tre dizionari per ogni circonvallazione. Il primo contenente i dati sul traffico eccetto l'orario (contenuto nel secondo dizionario) e il terzo contenente i dati relativi al meteo, comprensivi di data. Il primo passo è stato quello di integrare i primi due dizionari, associando all'i-esimo JSON del primo dizionario l'i-esimo JSON del secondo. Questo perché, per come è avvenuta la raccolta dati, nei file testuali dopo ogni "traffic json" era scritto il relativo "time json" e questa corrispondenza è stata mantenuta al momento della creazione dei dizionari. A questo punto i dizionari restanti erano due per ogni circonvallazione (traffico e meteo).

Abbiamo quindi unito i dizionari delle due circonvallazioni, ottenendo una collezione per il traffico e una per il meteo, entrambe contenenti tutte e 24 le vie. Ciò nonostante, l'informazione relativa alla circonvallazione di appartenenza non è andata persa, poiché è stata aggiunta come campo di ciascun record JSON.

A questo punto, è stata eseguita l'integrazione sincronizzata del traffico con il meteo, cercando di preservare il più possibile l'ordine temporale di entrambi i dataset. A tale scopo, abbiamo ricercato, per ogni record del traffico, il corrispettivo record del meteo che avesse l'aggiornamento della stazione meteorologica più vicino al campo **date** della viabilità, dal punto di vista temporale.

Poiché sono mancanti diversi documenti, dovuti a errori nella raccolta, i dizionari con i JSON di traffico e meteo relativi alla stessa via non hanno la stessa lunghezza. Può accadere, ad esempio, che i dati del meteo di una via siano in minor numero rispetto ai dati del traffico della stessa (o viceversa). Per questo motivo, la sincronizzazione tra i due dataset è di fondamentale importanza.

E' stata dunque effettuata una ricerca *greedy*:

- Abbiamo verificato che, per ogni via, la differenza tra la lunghezza del JSON del traffico e quello del meteo fosse sotto una certa soglia, in modo tale da ricercare il record 'ottimale' del meteo entro un determinato intorno.

- Successivamente, iterando sui dati del traffico, abbiamo individuato il corrispondente dato del meteo in un intorno di 16 record rispetto a quello preso in considerazione durante il ciclo (la differenza massima tra le lunghezze dei JSON di traffico e meteo vale al massimo 8).

- Una volta identificato il record del meteo 'ottimale', il dataset del traffico è stato integrato con il campo **weather**, contenente le informazioni sul meteo precedentemente uniformate e pre-processate.

Infine, per agevolare le interrogazioni sui dati e migliorare l'interpretabilità temporale del dataset, è stato aggiunto, per ogni record, un campo **dateTime** contenente le seguenti informazioni su data e ora:

- **trafficDate**: Data di misurazione (traffico).
- **weatherDate**: Data di misurazione (meteo).
- **time**: Ora teorica di misurazione.
- **effectiveTrafficTime**: Ora effettiva di misurazione dei dati relativi al traffico.
- **effectiveWeatherTime**: Ora effettiva di misurazione dei dati relativi al meteo.

5 Data storage

Dopo la fase di data integration ci siamo focalizzati sul processo di data storage. Una volta integrati i dati del meteo a quelli del traffico, sono stati creati dei singoli file JSON contenenti le informazioni relative alle condizioni meteorologiche e alla viabilità delle strade prese in esame.

Abbiamo quindi scelto di memorizzare ed interrogare i dati utilizzando un modello documentale: di conseguenza la scelta del DBMS è ricaduta su **MongoDB**. La decisione di utilizzare un modello ad albero deriva da diversi fattori:

- I dati restituiti dalle API erano in formato JSON;
- La ricerca di una struttura più complessa rispetto al modello relazionale, i.e. che ci permettesse di interrogare i dati innestati senza ricorrere ad un numero troppo elevato di *join*;
- Nonostante la ricerca di una maggiore complessità, abbiamo comunque escluso il modello a grafo, poiché esso mette in evidenza le relazioni che intercorrono tra i nodi.

Il nostro obiettivo dunque è stato quello di creare un db che potesse essere interrogato formulando *query* complesse, a diversi livelli di dettaglio: ad esempio, a partire da un orario preciso è possibile avere informazioni sulla viabilità di una determinata strada.

Entrando più nel dettaglio di questa fase, è stata utilizzata *PyMongo*, libreria di Python che permette di manipolare file in formato JSON ed è il driver Python ufficiale per **MongoDB**. Sull'editor di testo è stata creata una connessione che ha permesso di collegare Python a **MongoDB Compass**, utilizzato in locale. Successivamente, è stato effettuato il richiamo al database e alla collezione in cui si intendeva memorizzare i dati. I file JSON sono stati letti e caricati attraverso un ciclo, andando ad implementare la *collection* 'DataMan' del database 'DataMan_database'. Abbiamo ottenuto una *collection* composta da 47511 documenti, ognuno avente la medesima struttura e costituito dai seguenti campi principali:

- **_id**: Codice identificativo.
- **street**: Nome della via.
- **ringRoad**: Circonvallazione di appartenenza.
- **roadType**: Tipo di strada (e.g. strada secondaria).
- **LTZ**: Attività della Zona a Traffico Limitata.
- **traffic**: Informazioni sul traffico.
- **weather**: Informazioni sul meteo.
- **dateTime**: Informazioni temporali.

6 Data quality

Per la fase di data quality abbiamo deciso di analizzare principalmente la *completeness*, a causa della assenza di diversi documenti, e la *currency*, per via della natura 'temporale' del dataset. Inoltre, aggiungiamo un commento riguardante la *consistency*.

6.1 Consistency

Al fine di garantire la consistenza del dataset, i valori (specialmente quelli concernenti la dimensione temporale) sono stati standardizzati tramite lo stesso formato, e.g. le date hanno tutte lo schema 'yyyy-mm-dd' e gli orari 'hh:mm:ss'. Inoltre, per quanto riguarda il traffico, l'unità di misura delle velocità è espressa in km/h, mentre il tempo è indicato in secondi. Infine, le ZTL risultano consistenti con gli orari indicati nei campi **date** e **time** di **dateTime**.

6.2 Completeness

Un'analisi relativa alla completezza è necessaria, poiché non tutti i documenti che la macchina utilizzata avrebbe dovuto raccogliere sono stati effettivamente scaricati. La completezza è stata quindi valutata sia in riferimento agli orari, sia in riferimento alle vie, al fine di riuscire ad avere un'idea di quali di questi abbiano un maggior numero di documenti mancanti (2016 è il numero di documenti che idealmente il nostro database dovrebbe avere per ogni via, mentre 336 il numero per ogni orario). In particolare, è stato interrogato il database creato su **MongoDB** al fine di aggregare i dati, per ora e via, e contare quanti fossero presenti.

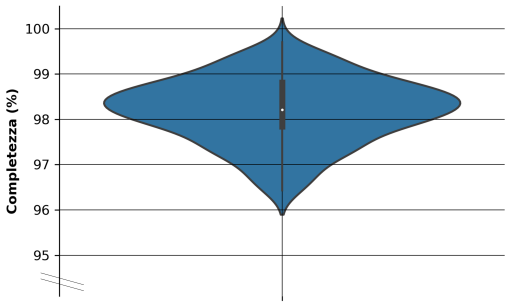
Per quanto riguarda la completezza oraria, ogni orario raccoglie tra i 324 ed i 335 record, inoltre:

- le $14{:}50{:}00$ hanno la maggior percentuale di valori non mancanti, pari al 99.7%;
- le $20{:}50{:}00$ hanno la minor percentuale di valori non mancanti, pari al 96.43%.

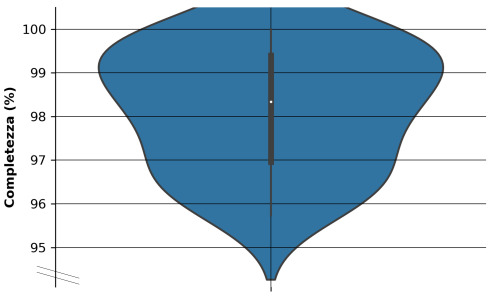
Per quanto riguarda la completezza per vie, ogni via contiene tra i 1930 ed i 2016 record, inoltre:

- Via De Amicis ha la maggior percentuale di valori non mancanti pari al 100%;
- Bastioni Porta Volta ha la minor percentuale di valori non mancanti pari al 95.73%.

Più in generale, le distribuzioni associate sono mostrate dai seguenti *violinplot*:



(a) Orario



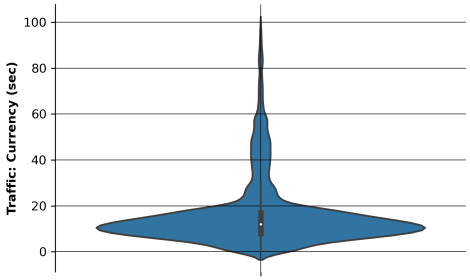
(b) Vie

Immagine 3: Completeness (%)

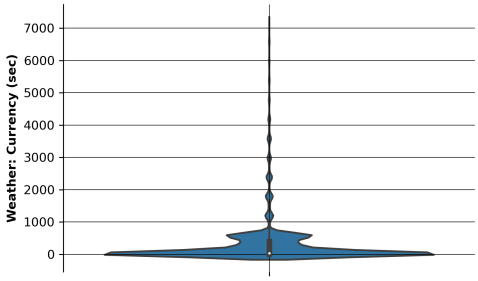
Complessivamente si ha una completezza pari al 98.2%, quindi una percentuale di valori mancanti pari a 1.8%.

6.3 Currency

Per quanto riguarda la *currency* relativa al traffico, è stata calcolata la differenza (in secondi) tra il tempo ideale di misurazione (i.e. **time** di **dateTime**) e il tempo effettivo di arrivo della risposta da TomTom (i.e. **effectiveTrafficTime** di **dateTime**). Riguardo al meteo invece, la differenza (sempre in secondi) è stata calcolata tra **time** e **effectiveWeatherTime**. Riportiamo i risultati ottenuti nei seguenti *violinplot*.



(a) Traffico



(b) Meteo

Immagine 4: Currency (sec)

La maggior parte dei dati relativi al traffico si concentra nella fascia 0-20 secondi. Considerando che le informazioni riguardano la viabilità e sono acquisite ogni dieci minuti, riteniamo tale intervallo più che accettabile. Per quanto riguarda il meteo invece, seppure la maggior parte dei punti si concentri in un intorno vicino allo 0, notiamo uno sviluppo più verticale del *violinplot*, con alcuni dati in ritardo di una o due ore. Questo è probabilmente dovuto alla variabilità del tempo di aggiornamento delle stazioni meteorologiche, che spesso non riportano informazioni per un lasso di tempo considerevole, spesso durante le ore notturne. Ciò nonostante, tali misurazioni sono in numero minoritario rispetto a quelle 'puntuali' e la notte vi è generalmente un minore traffico sulle vie prese in considerazione.

7 Conclusioni e sviluppi futuri

In conclusione, sono stati acquisiti dati in formato JSON sulla viabilità e il meteo relativi alle due circonvallazioni più interne di Milano (la cerchia dei Navigli e la cerchia dei Bastioni), durante le due settimane tra il 28 dicembre 2021 e il 10 gennaio 2022 (estremi compresi). Ciò è stato possibile grazie ai servizi API forniti da TomTom e Openweather. Le informazioni sono state acquisite tramite richiesta GET, effettuata ogni 10 minuti grazie all'ausilio di una macchina virtuale. Successivamente, i dati ottenuti sono stati puliti e rielaborati in un unico dataset, contenente, per ogni via, le informazioni relative a traffico e meteo ogni 10 minuti. Inoltre, per migliorare l'interpretabilità, sono stati aggiunti diversi campi (e.g. **LTZ** e **flowConditions**) e le informazioni temporali, contenenti il tempo 'ideale' e il tempo effettivo di misurazione, sia per il traffico che per il meteo. I singoli JSON (uno per ogni misurazione di ogni via) sono stati dunque memorizzati sul DBMS *document-based* **MongoDB**, tramite la libreria Python *PyMongo*. Infine, abbiamo analizzato la qualità del dataset, basandoci principalmente su *completeness* e *currency*.

Questo lavoro si presta a diverse analisi, in primis la verifica di un'eventuale correlazione tra le condizioni meteo e la viabilità delle circonvallazioni. E' possibile, inoltre, lavorare su tale dataset per scoprire come varia il traffico a seconda della circonvallazione, della singola via, del giorno della settimana, dell'orario o dell'attività di una ZTL. In futuro, questi dati potrebbero essere usati anche per effettuare delle predizioni sulla viabilità e sull'efficacia, ad esempio, delle misure predisposte dal Comune di Milano per agevolare la circolazione.