Tesina Progetto Open Data

Introduzione

In questa tesina presentiamo il progetto di tecniche per la gestione degli open data.
L'elaborato si propone di trasformare il dataset degli hotel del comune di Milano da 3 a 5 stelle, facendo uso di un'ontologia e serializzando i dati in formato Turtle.
Inoltre proponiamo una valutazione della qualità di un hotel in funzione della distanza di questi da altre strutture cittadine quali pub, bar, ristoranti, ecc.

Capitolo 1: Discussione Dati

Per effettuare le nostre elaborazioni ci siamo muniti dei dataset riguardanti gli hotel presenti nel comune di Milano e dei dataset relativi alle altre strutture cittadine.
In particolare, abbiamo recuperato tre dataset riguardanti gli hotel, ovvero:

- "hotel_milano_2018.csv"
- "MilanHotel.geojson"
- "ds48_turismotempolibero_strutture-ricettive-alberghiere_2015.json"

Paragrafo: Discussione dataset hotel_milano_2018.csv

Il dataset è stato ricavato dal sito di dati aperti, "dati.lombardia.it", all'indirizzo: "*https://dati.lombardia.it/Turismo/hotel-milano-2018/2p47-g7cn*".
Questo dataset contiene l'elenco delle strutture alberghiere, extra-alberghiere e complementari presenti a Milano fino al 2018.
Il dataset è distribuito sotto licenza IODL.
Questa licenza prevede la possibilità di condividere, modificare, usare e riusare liberamente la banca dati. Allo stesso tempo impone di citare la fonte delle informazioni e di pubblicare e condividere gli eventuali lavori derivati con la stessa licenza o con una licenza compatibile. Utilizzando

questo dataset abbiamo riscontrato una buona coerenza, ma una parziale completezza dei dati.
Inoltre il dataset è stato aggiornato l'ultima volta il 14/01/18, quindi attuale, dato che la frequenza di aggiornamento è ogni quindici anni.
Dopo aver utilizzato i dati, abbiamo rilevato una buona accuratezza degli stessi. Il dataset riporta, tra i vari campi, la denominazione della struttura, l'indirizzo, il numero di stelle e le coordinate geografiche.

Paragrafo: Discussione dataset MilanHotel.geojson

Il secondo dataset sugli hotel è stato ricavato da una query eseguita su Overpass Turbo, basato sul database di OpenStreetMap.
In questo caso, il dataset è distribuito sotto licenza ODbL.
Questa licenza prevede la possibilità di condividere e modificare il dataset con il vincolo di attribuirne la paternità sotto licenza ODbL e di mantenere aperto il dataset.
Il dataset di OpenStreetMap presenta una buona accuratezza e una ragionevole coerenza.
Il dataset riporta diverse informazioni per diversi hotel, che andranno uniformate durante la fase di pulizia.
La query che abbiamo usato per ricavare il dataset dall'endpoint all'indirizzo "*http://overpass-turbo.eu*" è la seguente:

```
[out:json][timeout:25];
nwr(around:10000,45.4637901,9.1895511)["tourism"="hotel"];
out center;
```

Paragrafo: discussione dataset ds48_turismotempolibero_strutture-ricettive-alberghiere_2015.json

L'ultimo dataset utilizzato per gli hotel è stato reperito dal sito di dati aperti "dati.comune.milano.it", all'indirizzo:
"*http://dati.comune.milano.it/dataset/fd725151-da06-4973-b4a9-6c41717e7db0/resource/d5edc7e1-557e-42ee-9123-5a0b30da6387/download/ds48_turismotempolibero_strutture-ricettive-alberghiere_2015.json*".
Il dataset contiene i dati sulle strutture ricettive alberghiere presenti sul territorio del Comune di Milano al 31 dicembre 2015.
L'ultima data di modifica risale al 11-06-2018.

Il dataset viene distribuito sotto licenza CC-BY, in cui si prevede di indicare sempre l'autore della banca dati.
Il dataset presenta varie informazioni, tra le quali le più interessanti sono: l'insegna dell'hotel, il numero di stelle ed il numero di camere.

Paragrafo: discussione dataset BikeRental, CarSharing, Car Rental

Tutti e tre i dataset sono stati ricavati da query effettuate su OverpassTurbo.
Il dataset BikeRental riporta l'elenco delle posizioni geografiche, degli operatori e della capacità di una postazione di noleggio.
Analogo discorso vale per il dataset CarSharing.
Il dataset CarRental riporta l'elenco delle aziende e delle relative sedi riguardanti le attività di car rental a Milano.
Tutti e tre i dataset sono in formato geojson.
Le tre banche dati sono disomogenee per quanto riguarda le informazioni riportate su una certa struttura. Nonostante ciò, le informazioni sono abbastanza precise.

Paragrafo: discussione dataset Restaurant, FastFood, Pub, Bar

Anche questi dataset sono stati ricavati da query eseguite su Overpass Turbo. Come da nome, i dataset elencano le strutture ristorative presenti nella città di Milano.
Nonostante la disomogeneità delle informazioni presenti, nel dataset dei ristoranti vengono spesso riportati: nome, sito web e indirizzo;
nel dataset dei pub vengono spesso riportati: nome e indirizzo;
nel dataset dei fast food vengono riportati il nome e l'indirizzo e se il fast food effettua servizio a domicilio e serve cibo da asporto; per quanto riguarda i bar viene riportato il nome, l'indirizzo e il sito web.

Paragrafo: discussione dataset RailwayStation, SubwayStation e BusStation, Parking e Park

Il dataset Park è stato ricavato dal portale open data del comune di Milano.
L'url della pagina è il seguente:
"*http://dati.comune.milano.it/dataset/8920bbea-fe1a-4061-8c6e-0746d95316c1/resource/7cba11e7-c236-49a7-aef4-4db6329eec5d*".

Il dataset presenta licenza CC-BY e l'ultima modifica è stato apportata lo scorso mese. Il dataset è molto ben curato e preciso.
I dataset delle stazioni dei treni, delle fermate della metro e delle fermate degli autobus cittadini presentano un'ottima accuratezza.
Il dataset Parking presenta un elenco dei parcheggi nella città di Milano.
Anche questi dataset sono stati ricavati con query Overpass Turbo.
Di seguito riportiamo le query effettuate per ricavare alcuni dei dataset:

Dataset Fast food: *[out:json][timeout:25];*
nwr(around:10000,45.4637901,9.1895511)["amenity"="fast_food"];
out center;

Dataset Ristoranti: *[out:json][timeout:25];*
nwr(around:10000,45.4637901,9.1895511)["amenity"="restaurant"];
out center;

Dataset Noleggio Auto: *[out:json][timeout:25];*
nwr(around:10000,45.4637901,9.1895511)["amenity"="car_rental"];
out center;

Paragrafo: discussione dataset Beni_culturali_Bella_Lombardia.csv

Il dataset sui monumenti, reperibile dal sito di dati aperti "dati.lombardia.it" all'indirizzo: "*https://www.dati.lombardia.it/resource/jrbq-nyaa.csv*", contiene tutti i monumenti della regione Lombardia, che poi verrà ristretto al comune di Milano tramite una selezione delle righe aventi la colonna "COMUNE" = "Milano". Per ogni monumento abbiamo considerato i metadati: nome, autore, definizione, abstract, latitudine e longitudine.
Anche questo dataset è distribuito sotto licenza IODL.
Inoltre la data di ultimo aggiornamento è stata il 21/12/2018 e viene aggiornato con cadenza semestrale.

Capitolo 2: Elaborazione Dati

Durante la presentazione dei dataset, abbiamo menzionato il fatto di avere tre dataset riguardanti gli hotel di Milano. Quindi abbiamo avuto la necessitá di unirli.
Molti dei dataset presentano una struttura disuniforme, ma attraverso alcuni programmi in python ci occupiamo di risolvere tali problematiche.

Paragrafo: pipeline di elaborazione dati Hotel

Sulla base dei dati disponibili abbiamo scelto un pool di metadati da utilizzare per descrivere le informazioni di un hotel.
Il dataset nativo *“hotel_milano_2018.csv”* è in formato csv.
Trasformiamo questo file da csv a geojson tramite lo script python *“FromCsvToJson.py”* che prende a parametro i nomi dei file di input e output, rispettivamente.
Il file risultante è chiamato: *“HotelMilano2018.geojson”*.
Rimuoviamo i campi non presenti nel pool definito.
Rimuoviamo anche la struttura del file, in maniera tale da renderlo flat, simile ad una lista di insiemi di coppie chiave-valore.
Eseguiamo tale operazione con lo script:
“FromFlatHotelMilano2018ToFinalFlatHotelMilano2018.py”
Il file restituito è pronto per essere ulteriormente elaborato, ma non necessita di ulteriori pulizie. Tale file è denominato *“FinalHotelMilano2018.geojson”*.

Il dataset nativo *“ds48_turismotempolibero_strutture-ricettive-alberghiere_2015.json”* non presenta informazioni circa le coordinate geografiche degli hotel. Fortunatamente presenta l’indirizzo fisico.
Sfruttando il servizio di OpenStreetMap è stato possibile ricavare le coordinate di ogni hotel. Dopo questa operazione rimuoviamo/aggiungiamo i campi del pool definito.
Rimuoviamo anche la struttura del file, in maniera tale da renderlo flat, più simile ad una lista di insiemi di coppie chiave-valore.
Lo script che realizza tutto ciò è *“aggiustaDs48HotelFile.py”*.
Il file restituito è *“FinalDs48HotelFile.geojson”*.

Allo stesso modo, il dataset *“MilanHotel.json”* presenta una struttura che preferiamo rimuovere e dei metadati superflui.

Rendiamo tale file strutturalmente uguale ai precedenti tramite lo script *“FromFlatMilanHotelToFinalFlatMilanHotel.py”*.
Il file restituito è *“FinalMilanHotel.geojson”*.

Adesso uniamo i file tramite lo script *“createFinalHotel.py”*.
Nell’effettuare tale operazione bisogna riconoscere gli hotel in comune tra i dataset. Per rendere semplice tale processo abbiamo implementato la classe hotel ed abbiamo sovrascritto il metodo *“__eq__()”*.
Tale metodo si basa sulla coseno similarità della denominazione degli hotel e sulla distanza tra questi. Il file restituito dallo script è *“Hotel.geojson”* e viene posto nella directory *“DatiPerElaborazione”*.

Paragrafo: pipeline di elaborazione dati CarRental, CarSharing, BikeRental

Questi dataset presentano un elenco delle postazioni di noleggio dei mezzi di trasporto. Questi presentano una struttura disorganizzata.
Creiamo una nuova struttura composta da nome, operatore, indirizzo e coordinate.
Il nome presenta il seguente pattern: “*operatore, indirizzo*”.
Con operatore intendiamo il gestore del servizio.
In questo modo ogni elemento del dataset è individuato dall’operatore che si occupa del servizio e dalla sua posizione.
Il nome è stato modellato in questo modo per semplificare la creazione delle uri. Nel file nativo non era presente il campo indirizzo. È stato ricavato tramite il servizio di geocoding inverso fornito da OpenStreetMap.

Paragrafo: pipeline di elaborazione dati Monumenti

Il file nativo dei monumenti è in formato csv.
Trasformiamo tale file in un file geoJson, tramite lo script *“FromCsvToJson.py”*.
Quindi rimuoviamo i metadati non strettamente necessari per l’elaborazione e rendiamo flat il file risultante. Questo viene realizzato con lo script *“FromCleanedMonumentToFinalFlatMonument.py”*.

Paragrafo: pipeline di elaborazione dataset rimanenti

Eseguiamo elaborazioni di eliminazioni dei campi e “flattizzazione” sui restanti dataset. Ovviamente i dataset non possono avere le stesse informazioni se riguardanti strutture diverse.

Paragrafo: calcolo della valutazione di un hotel

La formula cerca di dare una valutazione della qualità di un hotel in funzione della distanza di questo dalle altre strutture cittadine che disponiamo nel nostro dataset.
Pesiamo le strutture cittadine in maniera diversa. Ad esempio daremo maggiore importanza alla vicinanza di un monumento che alla vicinanza di un parcheggio.
Pesiamo le strutture cittadine in funzione della loro distanza dal centro di Milano. L’idea è: un ristorante (ad esempio) vicino il duomo di Milano dovrebbe essere più importante di uno situato in periferia.
Per pesare le strutture cittadine in base alla loro natura (se un monumento o se un parco) utilizziamo dei pesi arbitrariamente scelti a somma unitaria.
Per pesare le strutture cittadine in base alla loro distanza dal centro usiamo una gaussiana con media nel centro di Milano.
La formula finale che aggiunge il contributo di una certa struttura è:
valueHotel += coeffStruc * gaussiana([struct.latitude, struct.longitude]) / geopy.distance.distance([dataHotel.latitude,dataHotel.longitude], [struct.latitude, struct.longitude]).m.
Questa formula verrà applicata per ogni hotel ad ogni struttura.
Il file risultante da questa elaborazione conterrà le coppie uriHotel,valoreHotel.
Tre campioni:

- hotel brunelleschi: 0.095108978189444408;
- grand hotel et de milan: 0.10675356425604907;
- hotel carlo goldoni: 0.062050154184620024.

In particolare nel file report.txt possiamo vedere le uri delle strutture che distano meno di 200 metri da ognuno di questi hotel. L’hotel “Grand hotel et de milan” è circondato da più ristoranti rispetto agli altri due hotel. Inoltre ha vicino diversi monumenti, mentre questo non si verifica per nessuno degli altri due hotel. Questi risultati sono abbastanza coerenti con il numero di stelle degli hotel:

- hotel brunelleschi: 4 stelle;
- grand hotel et de milan: 5 stelle;
- hotel carlo goldoni: 1 stella.

Capitolo 3: Ontologia

L’obbiettivo principale del nostro progetto è rappresentare, in una base di conoscenza, la classe degli Hotel.
Esistono diverse classi esistenti che definiscono il concetto di hotel, quindi abbiamo preferito optare verso il riutilizzo di tali classi, piuttosto che crearne una nuova.
Principalmente abbiamo analizzato le classi Hotel definite nell’ontologia di “*DBpedia”* e di “*schema.org*”. La classe Hotel di *DBpedia* è sottoclasse di ArchitecturalStructure, che rappresenta struttura architettonica, che potrebbe anche essere un monumento od una statua. Quindi abbiamo preferito evitare il riutilizzo, poco consistente con la nostra applicazione, di tale classe. Mentre la classe definita nell’ontologia di “*schema.org*” sembra rispondere meglio alle nostre esigenze. Infatti questa è sottoclasse di LocalBusiness. Questa, a sua volta, è sottoclasse di Place e Organization. Quindi, dato che noi intendiamo il concetto di Hotel da un punto di vista economico e quindi assimilabile al concetto di attività commerciale, abbiamo deciso di riutilizzare la classe di “schema.org”.
In particolare, dato che molte proprietà da noi definite hanno come dominio questa classe, abbiamo pensato di creare una nostra classe LocalBusiness, e di rendere quest’ultima equivalente a quella di schema.org. Abbiamo posto tale fatto negli alignments.

Le proprietà della classe hotel che abbiamo definito sono:

- category: proprietà che specifica il tipo di struttura ricettiva;
- hasValutation: proprietà che specifica il valore di un hotel calcolato sulla base della nostra formula;
- hasClose: proprietà che ci permette di descrivere il fatto che una certa struttura dista meno di cento metri dall’hotel.

Data la presenza dell'ultima proprietà, si è rivelato necessario rappresentare nella nostra base di conoscenza le altre classi che definiscono le strutture cittadine. Anche in questo caso dopo un'analisi della gerarchia di ereditarietà delle classi definite in "*DBpedia*" e delle classi definite in "*schema.org*" abbiamo preferito riutilizzare le classi di quest'ultima. La motivazione che ci ha spinto a questa decisione segue dalla stessa discussione fatta per la classe degli Hotel.
In "*schema.org*" è presente una classe aggregata che rappresenta sia il concetto di Bar sia il concetto di Pub. Abbiamo preferito costruire le nostre classi di Bar e Pub, separate l'una dall'altra.
La classe dei ristoranti e la classe dei fast food vengono riutilizzati da "*schema.org*". Nella base di conoscenza di "*schema.org*" non è presente la classe che rappresenta un parcheggio. Allora abbiamo definito tale classe, ponendolo come sottoclasse di LocalBusiness, dato che il nostro modo di interpretare tale classe è di tipo economico. Per rappresentare i monumenti abbiamo impiegato una classe di tipo più generale presente in "*schema.org*", ovvero "*TouristAttraction*". Anche la classe che rappresenta i parchi pubblici viene riutilizzata da "*schema.org*".
Le classi per rappresentare le stazioni metro, le stazioni ferroviarie e le fermate dei bus viene riutilizzata dall'ontologia di "*schema.org*". Nel particolare queste sono sottoclassi di "*CivicStructure*". Per definire i concetti di car sharing, car rental e bike rental abbiamo creato nuove classi. Nel nostro dominio, questi tre concetti rappresentano dei punti di noleggio di mezzi di trasporto. Inoltre questi rappresentano un certo tipo di business.
Per modulare la rappresentazione dei concetti, abbiamo pensato di costruire due classi, "*SharingTransportBusiness*" e "*RentalTransportBusiness*", sottoclassi di "*TransportBusiness*".
Per definire il concetto di azienda di noleggio auto abbiamo quindi definito la classe "*CarRental*", sottoclasse di "*RentalTransportBusiness*". Per definire il concetto di postazione di noleggio bici abbiamo definito la classe "*BikeRental*", sottoclasse di "*RentalTransportBusiness*".
Analogamente abbiamo definito la classe "*CarSharing*" per specificare una postazione di auto condivisa.
Quest'ultima è stata definita come sottoclasse di "*SharingTransportBusiness*".
Per tutte e tre le classi abbiamo creato le proprietà "*organizationOfTransportSystem*" e "*addressOfTransportSystemStation*". Queste due proprietà specificano il nome del gestore che si occupa del

servizio e l'indirizzo della postazione del servizio, rispettivamente. Il dominio di queste due proprietà comprende anche le classi "*SubwayStation*", "*TrainStation*" e "*BusStation*", riutilizzate da "*schema.org*".
Abbiamo necessità di specificare la posizione geografica di un qualsiasi concetto di tipo "*Place*". Per fare questo potremmo sfruttare la proprietà "*address*" presente nell'ontologia di "*schema.org*". Se così facessimo avremmo bisogno di utilizzare dei "*blank node*", che non possono essere individuati tramite uri. Quindi, per risolvere questo problema abbiamo pensato di creare delle nostre proprietà "*latitude*" e "*longitude*", con dominio "*Place*" e range stringa. Per specificare meglio la classe "*Parking*", abbiamo creato le proprietà "*parkingCapacity*", "*parkingCharge*" e "*parkingType*".
Per descrivere la classe dei monumenti abbiamo aggiunto le proprietà che descrivono l'autore, il tipo del monumento ed un piccolo abstract.
La proprietà "*lineOfPublicTransportSystem*" definisce la linea percorsa dal mezzo di trasporto pubblico.

Capitolo 4: Triple e Linking

Per ogni risorsa contenuta nel dataset abbiamo create le triple che descrivessero tale entità, sfruttando le proprietà definite.
Le risorse della nostra base di conoscenza sono state collegate con le risorse di DBpedia. Per collegare le nostre risorse abbiamo scaricato in locale il risultato delle seguenti query effettuate su un endpoint sparql di DBpedia :

Query sparql con proprietà location:

```
select distinct ?uri ?label
where {
        ?uri dbo:location dbr:Milan.
        ?uri rdfs:label ?label.
}
```

GeoQuery:

```
select distinct ?uri ?label
where {
```

```
dbr:Milan geo:geometry ?sourcegeo .
?uri geo:geometry ?location ;
      rdfs:label ?label .
FILTER( bif:st_intersects( ?location, ?sourcegeo, 20 ) ) .}
```

Prendendo visione dei risultati, abbiamo pensato di collegare stazioni metropolitane, stazioni ferroviarie, stazioni bus, monumenti, hotel.
Le altre risorse non sono presenti nella base di conoscenza di DBpedia.
Nella maggior parte dei casi abbiamo preferito usare la proprietà "*owl:seeAlso"* piuttosto che la proprietà "*owl:sameAs*", così da evitare collegamenti inconsistenti tra risorse. L'operazione di linking si basa sulla coseno similarità tra le label delle risorse.