Community Profiling

An algebra for Twitter Knowledge Graphs Management

Carlo La Viola 461959, Davinder Kumar 460825, gruppo: Kucha

July 16, 2017

Introduzione

L'obiettivo di questo progetto è di studiare le relazioni tra gli utenti nei social network attraverso operazioni algebriche.
In input si ha un file JSON che descrive i profili degli utenti su Twitter e questi profili sono rappresentati in termini di knowledge graph.
Sono state prodotte le operazioni unione, intersezione e differenza tra i knowledge graph, che permettono di confrontare i profili degli utenti per avere una conoscenza approfondita sulle loro relazioni.
I risultati delle operazioni sono stati visualizzati con il database Neo4j, che offre ampie opportunitá di analisi per i grafi prodotti.

Step eseguiti

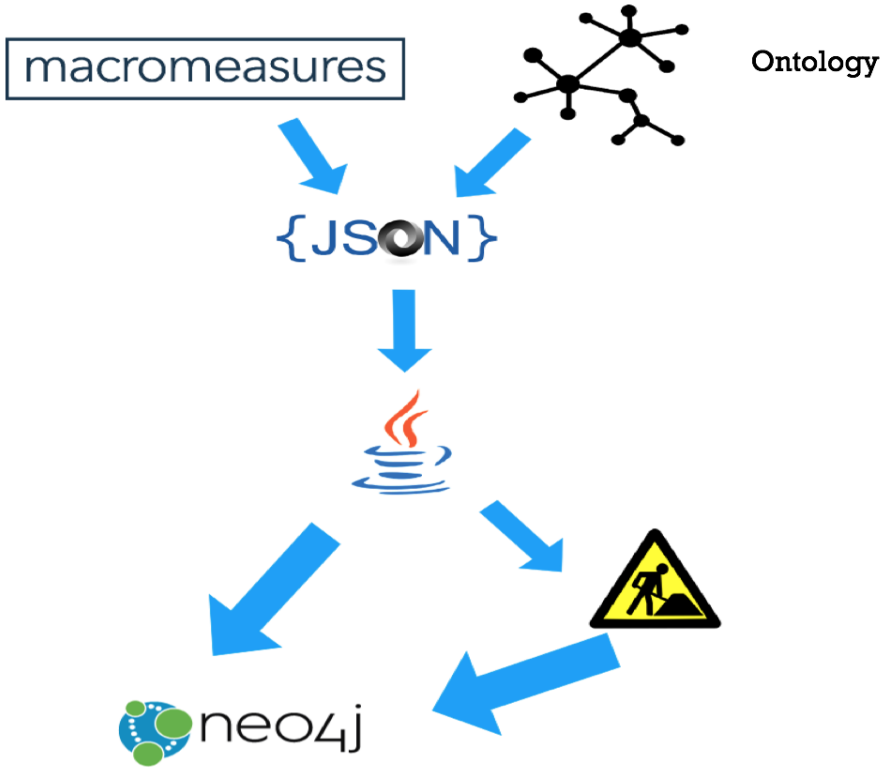

Figure 1: Step eseguiti

1 Estrazione da file JSON

Come accennato, questo progetto ha in input un file JSON. Questo file é creato a partire da un'ontologia, che attraverso delle regole, effettua l'estrazione. Cambiando ontologia o dominio di estrazione (in questo caso l'ontologia usata é fornita da *Macromeasures*, e il dominio di estrazione é *Twitter*), il file JSON prodotto avrebbe certamente un formato diverso. Dunque in questa prima parte l'obiettivo principale era effettuare una estrazione personalizzabile, che permettesse di effettuare le operazioni implementate su qualsiasi Knowledge Graph.
Per raggiungere questo scopo si é scelto di usare un file di configurazione, che rende l'estrazione personalizzabile. Esso si basa sul concetto che un Knowledge Graph é composto da **Entitá** e **Relazioni**; ogni entitá é un nodo del grafo con degli attributi, mentre le relazioni sono i label che vengono posti sugli archi del grafo.


```
{
  "_id": {
    "$oid": "5949550d3139b8042a7943c4"
  },
  "user": "1148580931",
  "info": {
    "valid": true,
    "type": {
      "updated": 1495148509,
      "personal": true,
      "confirmed": false
    },
    "interests": {
      "updated": 1495148509,
      "confirmed": false,
      "all": {
        "581261e892cffb71d2b9783b": {
          "useful": true,
          "score": 1,
          "parents": ["55afc38392cffb786d83f7b5"],
          "name": "Goodwood Festival of Speed",
          "level": "Low",
          "display": "Goodwood Festival of Speed (event)",
          "category": "event"
        },
        "581261b092cffb71d2b97835": {
          "useful": true,
          "score": 1,
          "parents": ["5457c811d4ac147c2bf5df5d"],
          "name": "Lotus Cars",
          "level": "Low",
          "display": "Lotus Cars (brand)",
          "category": "brand"
        },
        "57c9f4a092cffb3b2a0862ec": {
          "useful": true,
          "score": 1,
          "parents": ["55afc38792cffb786d83f92b", "5457c811d4ac147c2bf5df59"],
          "name": "Infiniti",
          "level": "Low",
          "display": "Infiniti (brand)",
          "category": "brand"
        },
```


Figure 2: Estratto del JSON in Input

Nella definizione del file di configurazione si sono utilizzate alcune convenzioni; vengono riportate di seguito per una maggiore chiarezza.

1. Campi annidati sono separato da '**.**'
2. Nel caso di piú elementi allo stesso livello di annidamento (una mappa), bisogna mettere '*****' per comunicare al parser di acquisire piú campi.
3. Si suppone che gli attributi siano sempre ad un livello di annidamento maggiore rispetto al campo che fa da ID; dunque per gli attributi viene indicato il path a partire dal campo chiave. Nel caso che non si vogliano estrarre attributi si inserisce **None**, mentre se si vogliono estrarre piú attributi per un elemento, essi vengono separati da '**-**'

```
1  REL:INTERESTED_TO KEY:user,ATTR:None VALUE:info.interests.all.*,ATTR:display
```

Figure 3: Configuration File

```
1  REL:IS_CHILD KEY:info.interests.all.*,ATTR:parents VALUE:info.valid,ATTR:None
```

Figure 4: Configuration File

In Figure 3 e Figure 4 si puó osservare la definizione delle due relazioni utilizzate per questo dominio. Si possono osservare 3 campi separati da spazi.

1. REL:[...] descrive il tipo di relazione tra i nodi;
2. KEY:[...] contiene il campo del JSON che fará da ID del nodo; ATTR[...] descrive gli attributi del nodo. In Figure 3 si puó notare che il livello di annidamento é 0, mentre in Figure 4 il livello di annidamento é 4
3. VALUE:[...] contiene gli ID dei nodi collegati al nodo descritto in *KEY:* ; possono essere, ovviamente, molti.

Note: I file JSON possono avere molte caratteristiche e particolaritá, a seconda di come viene effettuata l'estrazione. Ad esempio nella seconda relazione si nota che il campo chiave é composto da molti nodi, grazie all'uso di '*'. Va detto che per input troppo grandi, inserire una mappa nella chiave é fortemente sconsigliato. Per questo motivo si é implementata la possibilitá per l'utente di personalizzare la sua estrazione scrivendo semplici funzioni. In poche parole l'utente puó decidere di estrarre i dati in una forma non definitiva ma computazionalmente efficiente, per poi modificarli secondo le sue esigenze attraverso una UDF (User Defined Function).

Al termine dell'estrazione tutti i risultati sono stati scritti in una struttura dedicata, che conserva la relazione, i collegamenti e i nodi presenti, e sulla quale é possibile effettuare le operazioni implementate.

2 Le operazioni

Le operazioni algebriche implementate sono:

1. Unione
2. Intersezione
3. Differenza

Si é fornito un supporto guidato per eseguire le operazioni:

```
OPERAZIONI ALGEBRICHE TRA KNOWLEDGE GRAPH

Inserisci ID degli utenti separti dalla virgola
1148580931,728283781
Inserisci l'operazione da eseguire(Intersection, Difference, Union, No Operation)
Intersection
Ho inserito la relazione INTERESTED_TO
Inserisci la relazione su cui applicare UDF. Se non hai UDF scrivi None
None

INTERSECTION:
Operation terminated
INTERESTED_TO
1148580931 728283781 ,None:noattr 5457c811d4ac147c2bf5df55,display:Cars; 5457c811d4ac147c2bf5df59,display:Luxury vehicles;
```

Figure 5: Interazione con l'utente

La struttura dati utilizzata é:
Map<String,Map<GraphEntity,ArrayList<GraphEntity>>> relationsMap
RelationsMap é una mappa che ha come chiave la Relazione e come valore una mappa.
La mappa interna ha come chiave GraphEntity che sostanzialmente é una coppia di stringhe che rappresenta ID dello user e i suoi attributi e come valore si ha un ArrayList<GraphEntity>.
In ArrayList vengono riportati gli interessi; ogni elemento di questo insieme é un GraphEntity ovvero una coppia di stringhe che rappresentano ID dell'interesse e i suoi attributi.
Le operazioni algebriche fanno riferimento all'ID dell'interesse.
I risultati delle operazioni vengono visualizzati tramite il database Neo4J.

2.1 Unione

Il risultato di questa operazione saranno tutti gli interessi comuni e non comuni tra gli utenti. É possibile scegliere di fare l'operazione tra un insieme di utenti oppure tra tutti gli utenti presenti nel file. In caso si scelgono due o piú utenti su Neo4J si riporta un nodo centrale che ha un EntityID composto dalla concatenazione degli ID degli utenti passati in input.

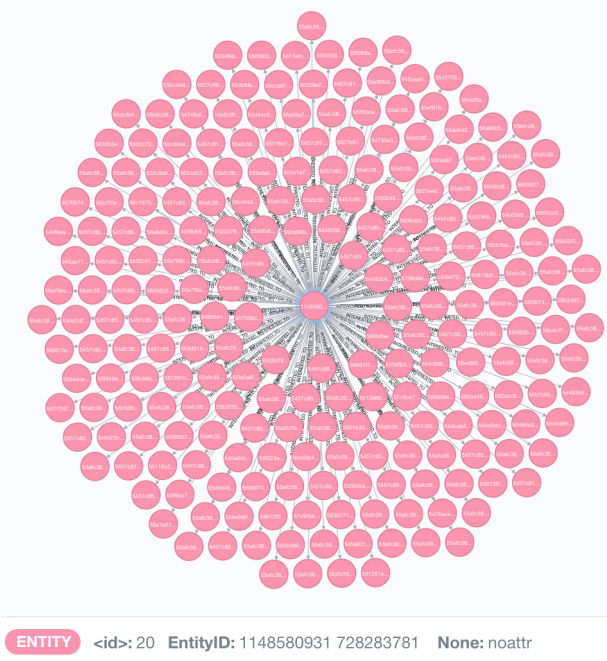

Figure 6: Unione tra gli utenti 1148580931 728283781

Se invece si sceglie l'opzione "all" alla richiesta degli utenti, il nodo centrale avrá EntityID pari a "all".

2.2 Intersezione

Quando si esegue l'operazione intersezione l'obiettivo é di ottenere tutti gli interessi in comune tra tutti gli utenti. Anche per intersezione é possibile passare in input gli utenti sui quali eseguire l'operazione. Si ha anche la possibilitá di lanciare l'operazione su tutti gli utenti presenti nel file di input ("all"). Per quanto riguarda il valore assegnato all'EntityID su Neo4J il ragionamento é analogo all'operazione unione.

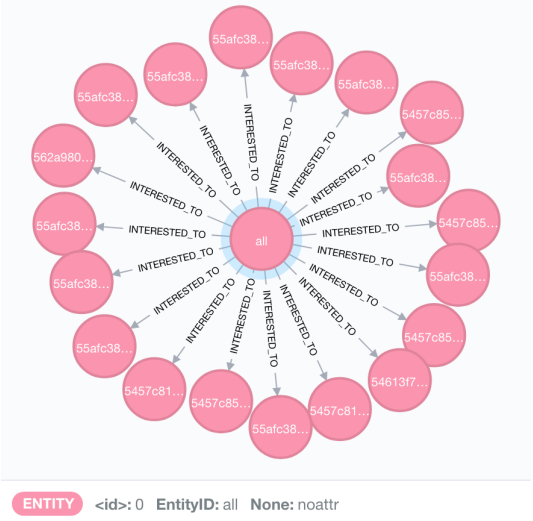

Figure 7: Intersezione tra 10 utenti

2.3 Differenza

L'operazione differenza produce tutti gli interessi che ha un utente e che non sono presenti in nessun altro utente. Alla richiesta degli ID degli utenti sui quali eseguire l'operazione si puó scegliere o due o piú utenti oppure se si desidera eseuire la differenza su tutti gli utenti é sufficiente l'ID dell'utente di interesse. Il nodo centrale su Neo4J riporterá in tutte e due i casi le informazioni dell'utente di interesse.

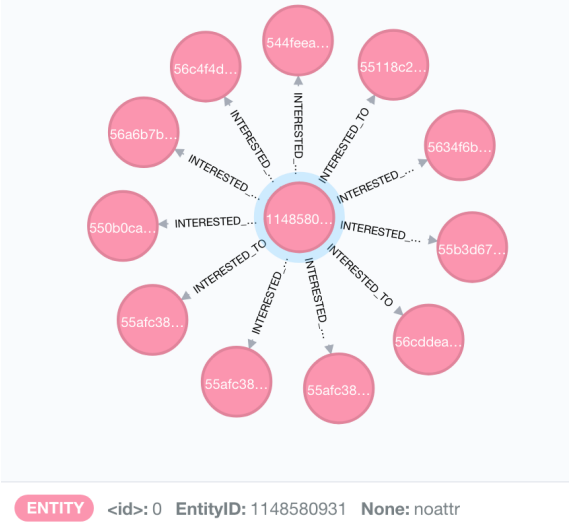

Figure 8: Differenza tra 100 utenti

No Operation

Inoltre é stata definita anche un operazione "No Operation" che serve sostanzialmente a collegare tra loro gli interessi; ovvero con le operazioni sopra riportate non riusciamo a capire il legame presente tra gli interessi e questa operazione serve a tale scopo. Di seguito si riporta un esempio di interazione che permette di capire il legame tra gli interessi di due utenti.

```
OPERAZIONI ALGEBRICHE TRA KNOWLEDGE GRAPH

Inserisci ID degli utenti separti dalla virgola
1148580931,728283781
Inserisci l'operazione da eseguire(Intersection, Difference, Union, No Operation)
No Operation
Ho inserito la relazione IS_CHILD
Inserisci la relazione su cui applicare UDF. Se non hai UDF scrivi None
IS_CHILD

No Operation:
IS_CHILD
55afc38492cffb786d83f822,None:noattr 55afc38492cffb786d83f811,None:noattr
56b22f3692cffb085ae9f1d8,None:noattr 55afc38892cffb786d83f96c,None:noattr
564b8fa192cffb22bc7eba1a,None:noattr 55afc38692cffb786d83f88a,None:noattr
```

Figure 9: No Operation

L'output di questa eseguzione viene riportato su Neo4j ed presente nella figura 10.

3 Scrittura su Neo4J

Come noto, *Neo4J* é un GraphDB. Dunque si é scelto di rappresentare i grafi prodotti dalle operazioni implementate proprio su questo DB. La scelta é stata guidata dal fatto di poter effettuare analisi e persistere i grafi prodotti. La potenza di Neo4J e del linguaggio **Cypher** permette di effettuare svariati tipi di analisi sui dati raccolti. Per i dettagli sulla configurazione del progetto, delle librerie, e dei driver, si rimanda alla documentazione presente su github (link alla fine della presentazione). Obiettivo di questo capitolo é descrivere la logica con la quale sono state effettuate le scritture. L'output delle operazioni é nello stesso formato della mappa prodotta a partire dal JSON. La classe che si occupa della scrittura prende la mappa <Nodo Chiave, Lista<Nodo Valore>> e per prima cosa crea il nodo chiave; dopodiché, per ogni nodo nella lista, lo crea e collega la chiave al nodo appena creato con una relazione; il label dell'operazione é acquisito direttamente dalla mappa ritornata dalle operazioni. In Figure 6, Figure 7 e Figure 8 si possono osservare alcuni grafi prodotti dalle operazioni.

Le figure successive mostrano come viene prodotto il grafo relativo agli interessi. Prima di tutto si genera il grafo sparso che descrive i collegamenti tra gli interessi; in questo caso si hanno piccole componenti sconnesse tra le quali vi sono nodi duplicati

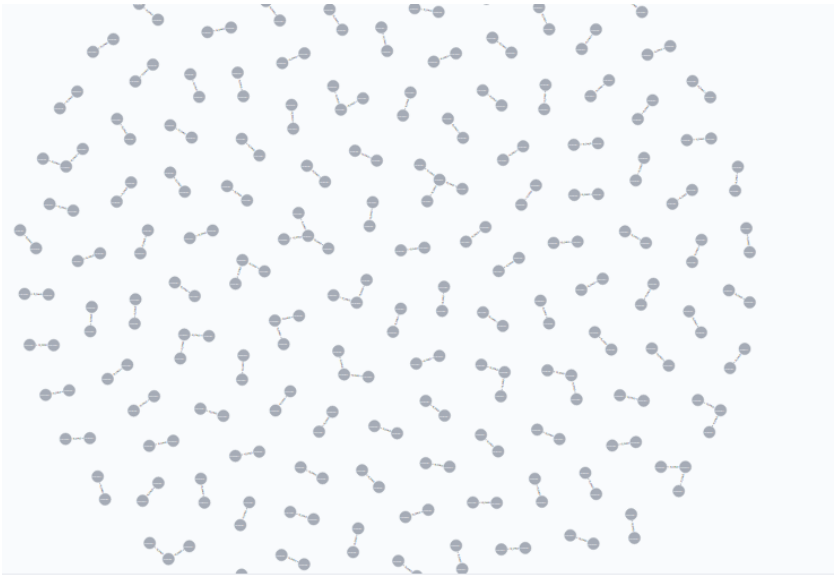

Figure 10: Grafo sparso di interessi

Poi, grazie ad una **Cypher Query**, i nodi con stesso ID vengono mergiati, formando cosi l'albero che descrive come sono collegati tra loro gli interessi.

Figure 11: Grafo Interessi Finale

```
MATCH (n:ENTITY)
WITH n.EntityID AS name, COLLECT(n) AS nodelist, COUNT(*) AS count
WHERE count > 1
CALL apoc.refactor.mergeNodes(nodelist) YIELD node
RETURN node
```

Figure 12: Query per mergiare nodi

Questa rappresentazione consente a chiunque voglia effettuare analisi o operazioni sui grafi di usare la potenza fornita da Neo4J e dal Cypher.
Note: La query proposta effettua correttamente il merging dei nodi, ma, inaspettatamente, modifica l'orientamento di alcuni archi. Si é cercata una soluzione al problema ma, data la sua natura casuale, ció risulta molto complicato.

4 Conclusioni e Sviluppi Futuri

In conclusione, il progetto sviluppato rispetta sicuramente le condizioni di adattabilitá a piú progetti e di analisi ed esplorazione dei grafi prodotti. Possibili sviluppi in questo senso riguardano sicuramente un supporto agli utenti maggiore, magari inserendo un'interfaccia grafica o semplificando ulteriormente il processo di configurazione. Ulteriori potenziamenti potrebbero essere fatti sulla query di merging per Neo4J e su eventuali configurazioni e automatizzazioni delle analisi da effettuare. Ovviamente un altro passo in avanti si potrebbe fare nell'implementazione di altre operazioni, senza dimenticare che le operazioni piú interessanti sono quelle i cui output sono utilizzabili per fare confronti o considerazioni sulla natura dei Knowledge Graph di diversi utenti, appartenenti o meno alla stessa comunitá.
Di seguito sono riportate le tabelle contenenti le misurazioni effettuate sui file forniti. Dai risultati raccolti si puó concludere che le componenti piú onerose sono la lettura del file e la scrittura sul DB, anche se con dataset discretamente grandi (15 mila utenti Twitter) le tempistiche di esecuzione risultano accettabili. I tempi di seguito riportati sono in secondi.

Table 1: Intersezione

	2utenti	1500utenti	15000utenti
Parsing	2.65	10.1	63.7
Operation	2.08	2.36	3.2
DB Writing	4.08	1.1	0.7

Table 2: Differenza

	2utenti	1500utenti	15000utenti
Parsing	2.65	12	63.7
Operation	1.9	1.2	3.3
DB Writing	5.3	0.9	0.7

Table 3: Unione

	2utenti	1500utenti	15000utenti
Parsing	2.65	9.5	63.7
Operation	0.9	2.1	2.6
DB Writing	5.3	15.74	16.18

Note:
Nella tabella di unione, le operazioni sono effettuate su pochi utenti, mentre nelle altre su tutti gli utenti forniti. Questa scelta per l'unione é dovuta allo scopo del progetto sviluppato. L'unione di due soli utenti produce un grafo molto denso e con molti nodi, difficile da esplorare. Effettuare l'unione di troppi utenti perde senso a livello di analisi e operazioni, poiché si produce un grafo in output con centinaia di migliaia di nodi.

Link Git

https://github.com/carloLV/KnowledgeGraph