Politecnico di Milano

Scuola di Ingegneria Industriale e dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica

Estrazione di regole di associazione da dati RDF

Luca Putelli

Matricola: 823310

Relatore: Alessandro Campi

Correlatore: Daniele M. Braga

Anno accademico 2015/2016

Indice

ABSTRACT

Contesto: la struttura a tripla soggetto-predicato-oggetto del linguaggio RDF per la rappresentazione dei dati permette di rappresentare una base di conoscenza, anche molto complessa, sottoforma di grafo. Questa tecnologia ha permesso la creazione di dataset, anche di notevoli dimensioni, connessi tra loro, disponibili e leggibili anche in modo automatico: i Linked Open Data (LOD). Una knowledge base di questo tipo può contenere informazione utile non solo nelle singole risorse o nei predicati, ma anche in come si relazionano tra loro, in particolari sottografi e configurazioni. L'estrazione di regole di associazione da dataset RDF presenta quindi problematiche diverse da quelle di transazioni lineari, come il riconoscimento di configurazioni, la trasformazione delle triple in una forma utilizzabile dagli algoritmi per l'estrazione e l'interpretazione delle regole.
Obiettivo: l'obiettivo di questa tesi è di presentare un framework e una metodologia per la ricerca di regole di associazione complesse tra strutture eterogenee di risorse e predicati. La metodologia è suddivisa in fasi, caratterizzata da trasformazioni tipiche e utilizza anche algoritmi e software noti di data mining. Questo approccio è in parte derivato da alcuni metodi per la gestione di strutture dati ad albero, ma si differenzia da questi per la presenza di problematiche nuove, come la possibilità di accedere a più dataset sfruttandone i collegamenti, la presenza di sottografi frequenti e l'introduzione di template di query in SPARQL, il linguaggio di interrogazione per RDF.
Risultati: il framework proposto opera a partire dalle knowledge base nella loro interezza, le semplifica e ne modifica la struttura attraverso opportune trasformazione, definendo e riconoscendo concetti complessi e integrandoli in una forma standard, che possa essere tradotta, con un algoritmo generale, nell'input per le procedure di estrazione delle regole di associazione. Queste vengono infine tradotta nuovamente in RDF, in modo tale da poter collegare ogni risorsa e concetto coinvolto alle sue relazioni nel dataset, senza perdita di informazione. La metodologia proposta viene applicata a diversi esempi, seguendo le fasi e le trasformazioni, al fine di dimostrare la validità degli algoritmi e dei concetti definiti.

1. Introduzione

1.1 Regole di associazione ed RDF

La definizione degli algoritmi per l'estrazione di regole di associazione, come Apriori ed FPGrowth, parte dal concetto di **itemset**: un insieme di oggetti, più o meno simili, che possono comparire nel dataset che si vuole analizzare. Il secondo concetto fondamentale è quello di **transazione**, ovvero di un sottoinsieme dell'itemset di elementi che hanno caratteristiche comuni, rappresentato come un vettore binario della lunghezza dell'itemset, in cui il valore 1 in posizione i sta a significare che l'elemento i-esimo è presente nella transazione. L'esempio classico di basket data analysis, vede come itemset tutta la serie di possibili prodotti che si possono acquistare in un supermercato, le transazioni invece sono gli effettivi acquisti dei consumatori, registrati sugli scontrini.
Le regole di associazione, in forma $\mathbf{A} \rightarrow \mathbf{B}$, mettono in relazione due sottoinsiemi disgiunti dell'itemset, A e B: la presenza all'interno delle transazioni di A, implica anche quella di B.
Queste definizioni sono nate e sono particolarmente significative nel modello relazionale, in cui di fatto l'item è il semplice valore di un campo di una tabella, rappresentato da un numero, un codice, o da una stringa di testo. Nel caso di dati espressi in **RDF** (Resource Description Framework), avendo un modello più ricco e maggiori possibilità, l'item non è più necessariamente un valore atomico: può essere un oggetto complesso, formato da diverse risorse, proprietà e di come queste interagiscono tra loro.
L'estrazione di regole in basi di dati espresse in RDF, e rappresentabili visivamente come un **grafo**, quindi, potrebbe non essere più la ricerca di uno o più valori che compaiano nelle stesse transazioni, ma un procedimento più complesso, in cui ogni item può essere definito come un sottografo e il collegamento tra i due sottinsiemi A e B può essere dato da una serie di proprietà.
Il World Wide Web Consortium (W3C), utilizza per il suo progetto Semantic Web e per la pubblicazione di dati collegati (i Linked Open Data, o LOD), le tecnologie RDF. Esistono numerosi **dataset**, anche di grandi dimensioni, con dati che spaziano dalla geografia (GeoNames), al contenuto di enciclopedie intere (DBPedia, o Wikidata), cinematografia (LinkedMDB), medicina e genetica (LinkedCT, Bio2RDF) e molti altri campi. Le linee guida del progetto fanno sì che i dataset abbiano proprietà comuni, in modo da poter passare da uno all'altro e poter relazionare dati di un dominio semantico con caratteristiche di un altro (per esempio, dati medici con la struttura geografica). Dove

spesso i database relazionali si occupano di una singola porzione di realtà, i LOD possono dar vita a modelli che integrano diverse discipline e domini di informazione, su cui trovare regole di associazione può essere interessante.

1.2 Natura del progetto

Il progetto consiste nella realizzazione di un framework e di una metodologia per l'estrazione di regole di associazione da un numero variabile di basi di dati RDF collegate tra loro. L'estrazione vera e propria viene svolta da algoritmi standard, forniti dai software di data mining, per cui vengono svolte alcune trasformazioni intermedie per adattare l'input dalla forma generale RDF a quella richiesta dai tool. Le fasi della metodologia sono le seguenti:

- l'**identificazione** delle fonti, locali o remote, ovvero dei dataset e dei loro cammini di join;
- la **riduzione** dei dataset, selezionando solo le parti interessanti, realizzazione di modelli RDF intermedi, costruiti dal **join** dei dataset scelti e dalla selezione di proprietà e risorse possibilmente interessanti, attraverso il linguaggio SPARQL;
- l'esecuzione di query e trasformazioni tipiche, per il **riconoscimento di configurazioni notevoli**, che possano essere processate per la ricerca di regole. Durante questa fase, i concetti complessi vengono ridotti in oggetti più semplici; per preservare tutta l'informazione presente nel dataset originale viene tracciata la provenienza, attraverso l'opportuna **apposizione di etichette**;
- la **realizzazione di una forma normale**, non più in RDF ma sottoforma di matrice **binaria**, che faccia da base per la costruzione delle transazioni, cioè dell'input per i diversi software di data mining;
- l'esecuzione, con opportuni parametri di supporto e confidenza, di algoritmi quali Apriori e FPGrowth richiamandoli direttamente da tool come Weka, Apache Spark, ecc., per la produzione di regole;
- la **ritrasposizione** degli item che formano le regole **nel formato RDF** e la **riassegnazione delle etichette**.

La base del progetto, necessaria per l'interrogazione e la manipolazione dei dataset, è il linguaggio SPARQL (acronimo ricorsivo di SPARQL Protocol and RDF Query Language), che permette sia di eseguire interrogazioni (anche con funzioni matematiche, controlli di stringhe, come in SQL per i database relazionali), sia di rimodellare documenti in RDF producendone di nuovi, attraverso la clausola CONSTRUCT. Le fasi di riduzione e di riconoscimento delle configurazioni notevoli sono composte da una serie di query e construct, operano a partire da dataset molto grandi e grafi molto complessi e producono modelli sempre più semplici, affrontando anche problematiche tipiche della

preparazione dei dati.
Il procedimento è generalizzato, con template di trasformazioni che dipendono sia dalla tipologia delle regole che si intende ottenere, di cui è proposta una classificazione (dalla più banale valore implica valore a quella che coinvolge due grafi generici), sia dai sottografi notevoli che modellano i concetti interessanti. Da queste caratteristiche, derivano inoltre il numero e il tipo delle etichette da apporre per l'interpretazione.
L'implementazione è realizzata in Java, utilizzando le librerie Apache Jena per l'esecuzione di SPARQL, sia da fonti remote tramite gli endpoint messi a disposizione dal proprietario dei dati, sia da modelli locali. Anche i tool di data mining, dove possibile, vengono utilizzati importandone le librerie per Java, e richiamando gli algoritmi all'interno del programma.

1.3 Struttura della tesi

Questo documento consiste, oltre a questa introduzione, di cinque capitoli, con il compito di illustrare i concetti pre-esistenti e la metodologia, sia da un punto di vista teorico che da quello più strettamente pratico:

- **2. Stato dell'arte**, dove vengono illustrate origini, caratteristiche e ruolo dell'RDF nel web semantico e viene introdotto SPARQL con alcuni esempi di query; ci si occupa delle regole di associazione, sia spiegandone i concetti chiave che illustrando gli algoritmi (con opportuni esempi) e i principali software; vengono presentate le principali tecniche per l'estrazione di regole di associazione per strutture dati ad albero e le relazioni tra l'RDF e i grafi, con alcune metodologie da cui è stato preso spunto o la cui integrazione potrebbe rappresentare uno sviluppo futuro;
- **3. Estrazione delle regole**, dove viene definito con maggiore profondità il problema e la sua risoluzione, con una descrizione sintetica framework e procedura, vedendone anche un esempio su due dataset online, a cui applicata tutta la metodologia;
- **4. Paradigmi di regole su grafi**, in cui viene introdotta una definizione rigorosa dei concetti, come le diverse tipologie di regole, le configurazioni intermedie che assumono i dati trattati e i diversi sottografi notevoli che si potrebbero riconoscere tra risorse e proprietà, mostrandone anche le condizioni SPARQL;
- **5. Estrazione delle regole: implementazione**: dove viene spiegata nel dettaglio la metodologia, mostrando anche il codice di query e algoritmi, nelle prime operazioni di base, nell'utilizzo dei concetti teorici durante le trasformazioni, nella costruzione dell'input per i tool di data mining e nel post-processing dell'output;

- **6. Conclusioni e sviluppi futuri**, in cui si riprendono i concetti dell'introduzione, mostrando risultati e difficoltà dell'applicazione della metodologia, oltre che fornire alcuni spunti per sviluppi futuri nello stesso ambito.

2. Stato dell'arte

2.1 RDF

2.1.1 L'idea e il linguaggio

L'idea che sta alla base del linguaggio RDF (Resource Description Framework) è molto semplice: un dominio di conoscenza può essere rappresentato attraverso una serie di piccole strutture chiamate triple.
Una tripla è composta da un soggetto, una proprietà e un oggetto e ponendo soggetto e oggetto come due nodi e le proprietà come dei lati, otteniamo un grafo come questo:

Figure 2.1: Struttura a tripla

Ogni soggetto è quindi una risorsa, identificata da un IRI (International Resource Identifier, una stringa Unicode univoca), su cui si intende specificare un particolare aspetto o caratteristica, mettendolo in relazione con qualcos'altro, che può essere sia un'altra risorsa oppure un letterale, cioè una stringa o un valore numerico.
Gli oggetti, a loro volta, possono diventare soggetti di altre triple, componendo grafi più complessi, formando cicli, sottografi ad albero, relazioni bidirezionali ecc. come si può vedere nella Figura 2.2.
Un documento RDF può essere quindi essere scritto come una lista di triple, separate da un punto, ad esempio:

<http://www.w3.org/#me> <http://www.w3.org/#fullName>"Eric Miller" .
<http://www.w3.org/#me> <http://www.w3.org/property/#mail> e.miller@example .
<http://www.w3.org/#me> <http://www.w3.org/property/#personalTitle> "Dr." .
<http://www.w3.org/#me> <http://www.w3.org/rdfType > <http://www.w3.org/#Person>.

tuttavia esistono altre sintassi, che possono essere più compatte, leggibili e simili ad altri linguaggi. Una di queste è Turtle (Terse RDF Triple Language), in cui si utilizzano prefissi per rappresentare le parti comuni degli IRI e si può

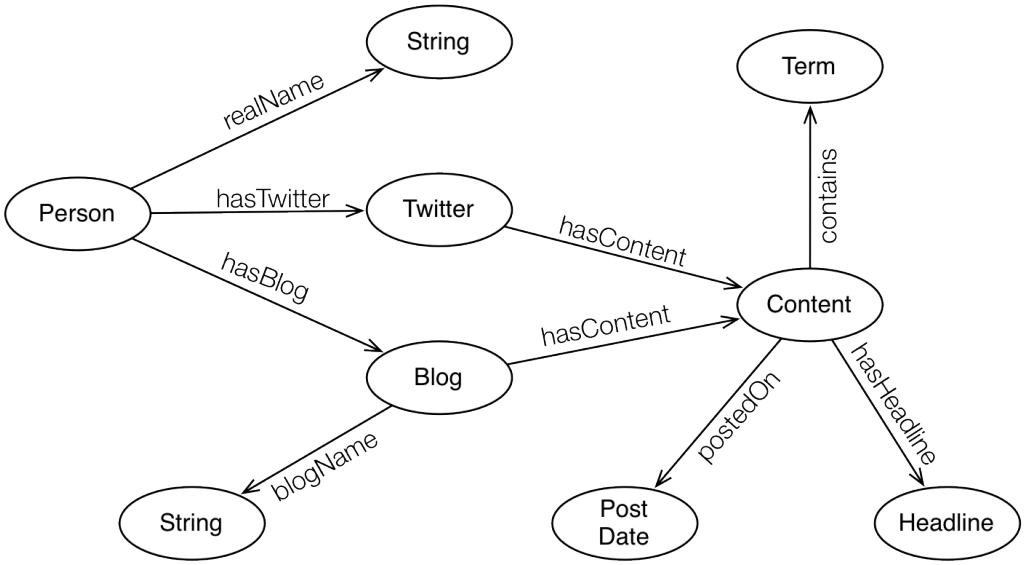

Figure 2.2: Grafo RDF

utilizzare il carattere ';' per elencare diverse relazioni dello stesso soggetto. Le stesse informazioni contenute nell'esempio precedente potrebbero quindi essere scritte in questa maniera:

```
@prefix eric: <http://www.w3.org/resource/#>.
@prefix prop: <http://www.w3.org/property/#>.
@prefix rdf: <http://www.w3.org/rdfProperty/#>.

eric:me contact:fullName "Eric Miller" ;
contact:mailbox <e.miller@example>.
eric:me contact:personalTitle "Dr." .
eric:me rdf:type contact:Person .
```

Un'altra sintassi molto utilizzata prevede la rappresentazione delle triple in XML, ereditandone la struttura e le regole, ma con alcune caratteristiche in più. Anche questa prevede l'utilizzo di prefissi (rdf, my, dbp, rdfs):

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:my="http://myvocabulary.com/"
    xmlns:dbp="http://dbpedia.org/ontology/"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" >
```

Le proprietà della risorsa vengono descritte prima definendo di quale soggetto si parla (il film "A day of the races", che ha la sua pagina su DBPedia), e poi,

come elementi figli, le proprietà (my:starring, equivalente a <http://myvocabulary.com/starring>o dbp:director, equivalente a <http://dbpedia.org/ontology/director>) con il rispettivo valore, nell'attributo rdf:resource oppure all'interno del tag, come nel caso di rdfs:label che si collega con la stringa del titolo.

```
<rdf:Description rdf:about="http://dbpedia.org/resource/A_Day_at_the_Races_(film)">
    <my:starring rdf:resource="http://dbpedia.org/resource/Groucho_Marx"/>
    <my:starring rdf:resource="http://dbpedia.org/resource/Chico_Marx"/>
    <dbp:director rdf:resource="http://dbpedia.org/resource/Sam_Wood"/>
    <rdfs:label>A Day at the Races</rdfs:label>
</rdf:Description>
```

In questo documento, si utilizzeranno tutte e tre queste sintassi, a seconda della convenienza del caso.

2.1.2 L'utilizzo di RDF e i Linked Open Data

Nel web tradizionale, moltissime informazioni vengono rappresentate tramite il linguaggio naturale, che con le sue ambiguità sintattiche e semantiche (pensiamo ai diversi significati di un vocabolo, ai modi di dire ecc.) è molto difficile da processare e comprendere a livello automatico. E' stato quindi necessario creare una serie di nuovi strumenti e linguaggi, in un progetto del W3C chiamato Semantic Web, affinchè i dati contenuti in un documento possano essere raccolti, selezionati e utilizzati per trovare nuova informazione, attraverso l'inferenza logica, in modo automatico da delle macchine.
RDF è uno dei cardini di questo progetto, fornendo, grazie alla sua struttura rigida ma anche adattabile ai diversi contesti, la giusta sintassi per la rappresentazione dei dati e l'aggiunta di un livello semantico, con i linguaggi RDF-Schema e OWL, per definire classi, proprietà e le relazioni tra di esse.
Ogni dataset RDF, tuttavia, può avere le proprie descrizioni e caratteristiche interne, che, come nel caso dei database relazionali, può rappresentare un limite a questi intenti di scambio ed elaborazione automatica. Pensiamo a due dataset separati: uno che contenga le informazioni riguardante film (LinkedMDB, per esempio), un altro che invece presenti i dati biografici di personaggi pubblici (DBPedia ne raccoglie molti): un'idea potrebbe essere di combinare i due argomenti, trovando caratteristiche, quali la provenienza geografica o l'età, di attori e registi di alcuni film particolari. Per fare ciò, è necessario che il film abbia, tra le sue proprietà, il riferimento alla risorsa dell'attore o del regista dell'altro dataset, in modo da poter fare un "join". Se il riferimento invece è solamente interno, con un URI ad uso e consumo dello stesso dataset dei fim, l'intersezione non è più possibile (se non utilizzando i rischiosi nome e cognome).
Per evitare questo genere di problema, esistono i Linked Open Data. Il nome è già esplicativo di per sè, sono dati resi pubblici da organizzazioni (istituzioni, aziende, agenzie ecc.), seguendo linee guida in modo tali da renderli collega-

bili e sia possibile fare delle analisi che occupino diversi dataset e argomenti. Per esempio, le nazioni della Banca Mondiale sono tutte oggetti della classe "Country" definita su DBPedia, e contengono un riferimento alla stessa risorsa su GeoNames, enorme raccolta di dati geografici con città, comuni, regioni ecc. Oppure, i film su LinkedMDB hanno il riferimento alla rispettiva pagina su Freebase, progetto enciclopedico di Google confluito poi in Wikidata. Tuttavia, cambiamenti e spostamenti in un dataset possono far sì che il collegamento non sia più valido o che non sia immediatamente fattibile, riuscendo a ricostruire il join attraverso operazioni più complesse, manipolando nomi ed indici.
I dataset sono resi pubblici per il download, di archivi di dimensioni molto spesso notevoli. Per facilitarne l'utilizzo in locale quindi, è stato creato un formato binario per RDF, chiamato HDT (Header, Dictionary, Triples) molto più compatto della normale notazione, e quindi capace di ridurre la dimensione del dataset anche di 15 volte. HDT permette inoltre di essere indicizzato, facilitando quindi anche l'esecuzione di query e riducendone il tempo di calcolo. Questo formato non è ancora uno standard W3C, tuttavia esistono diversi dataset (come GeoNames, Wikidata, Wikictionary) disponibili anche in questa versione ridotta.
I LOD possono essere accessibili anche tramite endpoint, cioè attraverso un servizio rivolto all'esterno che riceve query SPARQL, formulate dagli utenti, e ritorna i risultati in diversi formati (HTML, XML, JSON, lo stesso RDF ecc.). Esistono diverse tecnologie per implementare un web server che faccia da endpoint, tra cui OpenLink Virtuoso, utilizzato tra gli altri per DBPedia e Bio2RDF (che contiene dati genetici e biologici) e D2RQ, che è quello ad esempio di DBLP, raccolta delle pubblicazioni accademiche sull'informatica.

2.1.3 SPARQL

In questa sezione, entriamo maggiormente nel dettaglio del linguaggio di interrogazione SPARQL (arrivato alla versione 1.1) e di come estrarre informazioni da dataset RDF. Analogamente a SQL per le basi di dati relazionali, troviamo parole chiave come SELECT, FROM, WHERE e simili, anche se a volte con significati e utilizzi diversi.
La più grande differenza tra i due linguaggi riflette quella tra i due modelli di rappresentazione: quello a tripla per il primo, quello relazionale per il secondo. Non esiste quindi nessuna forma di tabella: la ricerca va fatta selezionando e manipolando triple, e questa struttura sta alla base di tutte le query possibili. Per brevità, come nella sintassi Turtle, anche una query SPARQL può essere preceduta da prefissi, per indicare la parte fissa di URI di proprietà e risorse. Partiamo quindi con un esempio molto semplice, su questi dati che rappresentano due persone (le risorse a e b) con i loro nomi, legati attraverso la proprietà <http://xmlns.com/foaf/0.1/name>:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/>.

a foaf:name "Alice" .
b foaf:name "Bob" .
```

La query SPARQL per estrarre i nomi delle risorse riflette la struttura della tripla soggetto-proprietà nome-stringa del nome:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/ >
SELECT ?x ?name
WHERE { ?x foaf:name ?name }
```

La parte WHERE contiene le condizioni della query, dove le variabili vengono precedute dal carattere "?". ?x prenderà dunque i valori di tutti i soggetti che possiedono la proprietà foaf:name, per cui (nel documento di esempio) le risorse a e b, mentre ?name saranno i corrispondenti oggetti. Il risultato sarà quindi:

a	Alice
b	Bob

La selezione avviene a livello di triple, per cui in caso di una risorsa con più nomi, la stessa query produrrà come risultato una riga risorsa-stringa per ogni oggetto della proprietà name.
Mostriamo un esempio più complesso:

```
SELECT DISTINCT ?distributor
WHERE {
        ?film a dbo:Film ;
        dbo:distributor ?distributor ;
        dbo:director ?director .
        ?director dbo:birthPlace <http://dbpedia.org/resource/Manhattan>.
        ?distributor dbo:founded ?date .
        FILTER(year(?date) >1920)
}
```

Anche in SPARQL, come in Turtle, è possibile evitare la ripetizione del soggetto, nel caso serva specificarne diverse proprietà, attraverso il ";", per cui la variabile ?film avrà le proprietà a, dbo:distributor e dbo:director. Mentre seconda a terza sono abbastanza intuitive (il responsabile della distribuzione del film e il regista), la prima è una parola chiave di SPARQL, corrispondente alla proprietà rdf:type e indica l'appartenenza della risorsa a un tipo definito da un'ontologia, o a una classe (in questo caso quella generica dei film definita su DBPedia).
Vengono però testate anche condizioni relative a regista (che deve essere nato a Manhattan), e al distributore, di cui ricaviamo la data di fondazione e successivamente testiamo, con la clausola FILTER, se l'anno è superiore al 1920.

Il grafo che rispetta questi vincoli è quello in Figura 2.3.
Analogamente al modello relazionale, anche in SPARQL è possibile calcolare

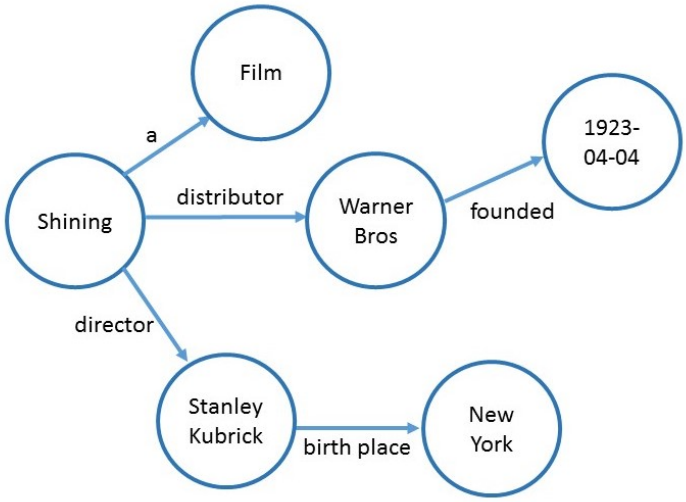

Figure 2.3: Risorsa e proprietà di Shining

medie, somme, conteggi e altre forme di aggregati, con le solite parole chiave, potendo anche raggruppare i risultati (GROUP BY) e filtrandone anche i risultati (HAVING), con una sintassi che, a prescindere dalle condizioni a tripla, è del tutto uguale a quella di SQL.
Con SPARQL, è possibile anche fare domande dirette al dataset, con risposta vero/falso, attraverso la clausola ASK. Il pattern è molto simile a quello delle query:

ASK WHERE { conditions }

Più interessante invece la possibilità di rimodellare i risultati in nuovi modelli RDF, con la clausola CONSTRUCT. Questo permette di creare nuove risorse, nuove proprietà, spostare elementi in base alle proprie esigenze.

```
CONSTRUCT {

    ?province <http://myvocabulary.com/leaderName>?name ;
    <http://myvocabulary.com/party>?party ;
    <http://myvocabulary.com/capital>?capital ;
    <http://myvocabulary.com/mayorParty>?mayorParty ;
    <http://myvocabular.com/popTotal>?tot ;
    <http://myvocabulary.com/over>?over .
} WHERE {{
    ?province dbpprop:settlementType <
    http://dbpedia.org/resource/Provinces_of_Italy>;
    dbpprop:leaderName ?name ;
    dbpedia-owl:populationTotal ?tot;
```

```
        dbpprop:seat ?capital .
        VALUES ?over { 1 }
        OPTIONAL{ ?province dbpprop:leaderParty ?party} .
        OPTIONAL{ ?capital dbpedia-owl:leaderParty ?mayorParty}
        FILTER(?tot >= 300000)
} UNION {
        ?province dbpprop:settlementType <
        http://dbpedia.org/resource/Provinces_of_Italy>;
        dbpprop:leaderName ?name ;
        dbpedia-owl:populationTotal ?tot;
        dbpprop:seat ?capital .
        VALUES ?over { 0 }
        OPTIONAL{ ?province dbpprop:leaderParty ?party} .
        OPTIONAL{ ?capital dbpedia-owl:leaderParty ?mayorParty}
        FILTER(?tot <300000)}
}
```

In questa construct, per esempio, vengono rimodellate le province italiane, mettendo un flag (il valore della variabile ?over), assegnato con la clausola VALUES ad 1 se la provincia ha più di 300000 abitanti (per le query che quindi vengono selezionate nella prima parte della where), 0 altrimenti (nella parte successiva, dopo la UNION).
Viene poi posizionato il partito del sindaco del capoluogo direttamente come proprietà della provincia: questa particolare trasformazione di avvicinamento, banale ma anche molto utile, servirà poi nel processo di conversione dal modello a grafo generico di RDF a uno più piatto e più semplice da trattare.

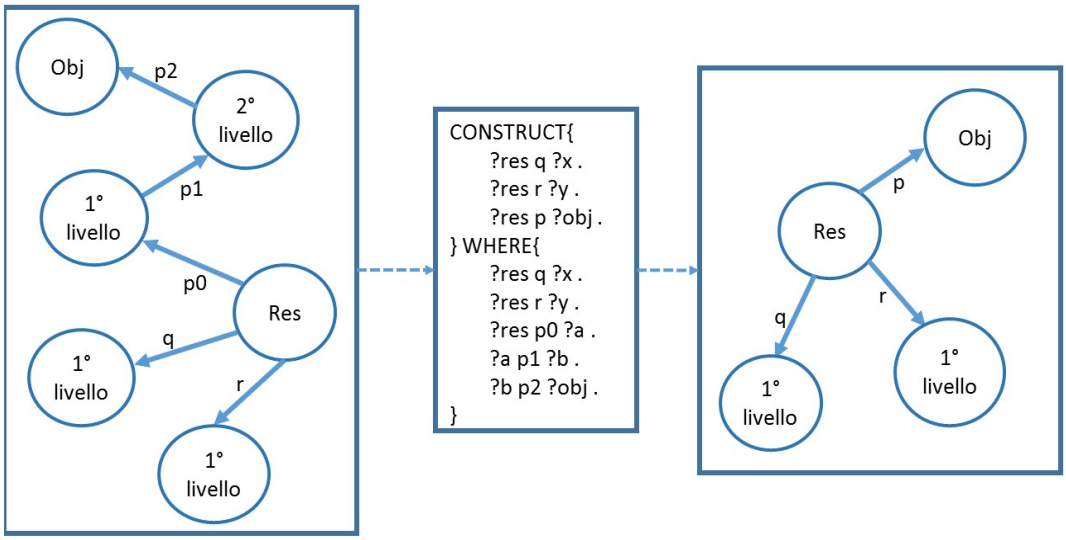

Figure 2.4: Livelli e avvicinamento

2.1.4 Come interrogare: librerie e servizi

Il modo più semplice per interrogare un dataset RDF online è attraverso l'endpoint. Specialmente nei casi di Virtuoso e D2R, questo comprende anche una pagina web con una form in cui inserire la propria query e selezionare il formato del risultato.
In caso di database locali o di progetti più complessi dell'eseguire query, che comprendono magari una rielaborazione complessa del risultato, SPARQL ha diverse implementazioni, API e query engines.
Molto utilizzate in questo progetto, le librerie Apache Jena che permettono di gestire dataset RDF in Java a tutti i livelli: importando modelli da file (in tutte le sintassi, HDT compreso), eseguendo query in locale o attraverso l'URL dell'endpoint, traducendo i risultati, siano essi tabelle o formati da altre triple nuovamente interrogabili. Jena fornisce inoltre la possibilità di creare un server SPARQL, Fuseki (giunto alla seconda versione). Alternativo a Jena, sempre per il trattamento di dati RDF in Java, c'è Sesame che fornisce anche un proprio linguaggio di interrogazione, SeRQL.
Analogamente, SPARQL può essere integrato in software scritti anche con altri linguaggi, grazie a Rasqal per C, SPARQL Python Wrapper, RAP e EasyRDF per PHP. Anche molti software di reasoning, che operano su RDF e OWL, hanno integrato un motore SPARQL. Sono senz'altro da citare AllegroGraph, Protégé e Pellet.

2.2 Regole di associazione

Le regole di associazione fanno parte di quella serie di informazioni elaborate e utili che si possono ricavare da grandi basi di dati, grazie agli algoritmi e alle tecniche che compongono questa branca del data mining. Sono state introdotte nel 1993 da Rakesh Agrawal, Tomasz Imielinski e Arun Swami, con una delle applicazioni più citate: l'analisi degli acquisti all'interno dei supermercati, trovando regole per cui se il cliente compra alcuni prodotti, automaticamente avrà nel carrello anche degli altri. Attualmente, i campi di applicazione sono molteplici e comprendono diagnosi mediche, censimenti, utilizzo del web, relazione con i clienti ecc.

2.2.1 Concetti fondamentali

I dati da cui partire per estrarre questo genere di informazioni hanno una struttura fissa e prendono il nome di transazioni. Queste sono semplicemente delle liste di elementi, sottoinsiemi dell'insieme generico I, composto da tutti gli oggetti che si possono trovare nell'analisi.
Dato un database composto da un gran numero di transazioni, si intende trovare una regola A =>B, tale per cui la presenza di A (un itemset, letteralmente un insieme di oggetti) in alcune transazioni, implichi, con una certa

sicurezza, che nelle stesse sia presente anche gli elementi di B.
Non tutte le regole però sono uguali e utili allo stesso modo: l'insieme A potrebbe essere presente solo in un numero esiguo di transazioni, oppure la regola non è sufficientemente sicura da essere applicabile (pensiamo alle diagnosi mediche, dove la certezza è fondamentale). Per questo motivo, fin dall'originale definizione del problema delle regole di associazione, sono state introdotte due metriche per misurare l'utilità del risultato (entrambe, come si può facilmente intuire, sono numeri compresi tra 0 e 1):

- il supporto, ovvero il rapporto tra le transazioni che contengono A e il totale;
- la confidenza, cioè il rapporto tra le transazioni che contengono sia A e che B e quelle che contengono A.

Una buona regola, deve avere un buon supporto e una buona confidenza, con esigenze che possono dipendere dal caso in analisi. Non esiste una misura sicura, alzando molto il supporto si potrebbero perdere alcuni casi significativi di nicchia (per esempio, nell'esempio classico di basket-case analysis, potrebbe tralasciare gli articoli molto costosi, poco venduti ma redditizi), ma tenendolo troppo basso si potrebbero produrre molte regole inutili, con costi significativi poi nell'analisi. Una confidenza troppo alta, per esempio del 100%, potrebbe far tralasciare alcuni casi importanti, e potrebbe non avere la tolleranza necessaria a errori nei dati, valori mancanti ecc.
L'estrazione di regole necessita di due fasi distinte:

- la ricerca dei frequent itemset, cioè di quegli elementi che compaiono insieme con supporto sufficiente;
- la produzione di regole che abbiano una confidenza che superi la soglia prefissata.

La prima parte, specialmente con dataset molto pesanti e transazioni composte da un numero elevato di elementi, può essere molto dispendiosa in termini di complessità. Il metodo più semplice, quello di generare tutti i frequent itemset con il metodo di forza bruta, è assolutamente proibitivo, per cui esistono diversi algoritmi più sofisticati.

2.2.2 Apriori

Lo stesso Rakesh Agrawal, insieme a Ramakrishnan Srikant, ha introdotto l'algoritmo più famoso per la generazione dei frequent itemset. Apriori sfrutta una proprietà del supporto: dato un itemset X, il supporto delle transazioni che contengono Y sovrainsieme di X è sicuramente minore o uguale a quello di X.
Per i frequent itemset, la tecnica è quella di unire la generazione combinatoria

all'eliminazione dovuta al supporto non sufficiente. A partire dagli itemset di un solo elemento (cioè di fatto tutti gli elementi possibili, 1-itemset), si eliminano quelli che hanno supporto inferiore alla soglia fissata. Si generano poi i k+1-itemset candidati da quelli rimasti di livello k, sfruttando appunto la proprietà anti-monotona del supporto, controllando poi quali superano la soglia e così via.
I frequent itemset da estrarre saranno quindi i candidati, più o meno numerosi, che non hanno un sovrainsieme frequente.
Molto simile ad Apriori, che parte però da una rappresentazione diversa delle

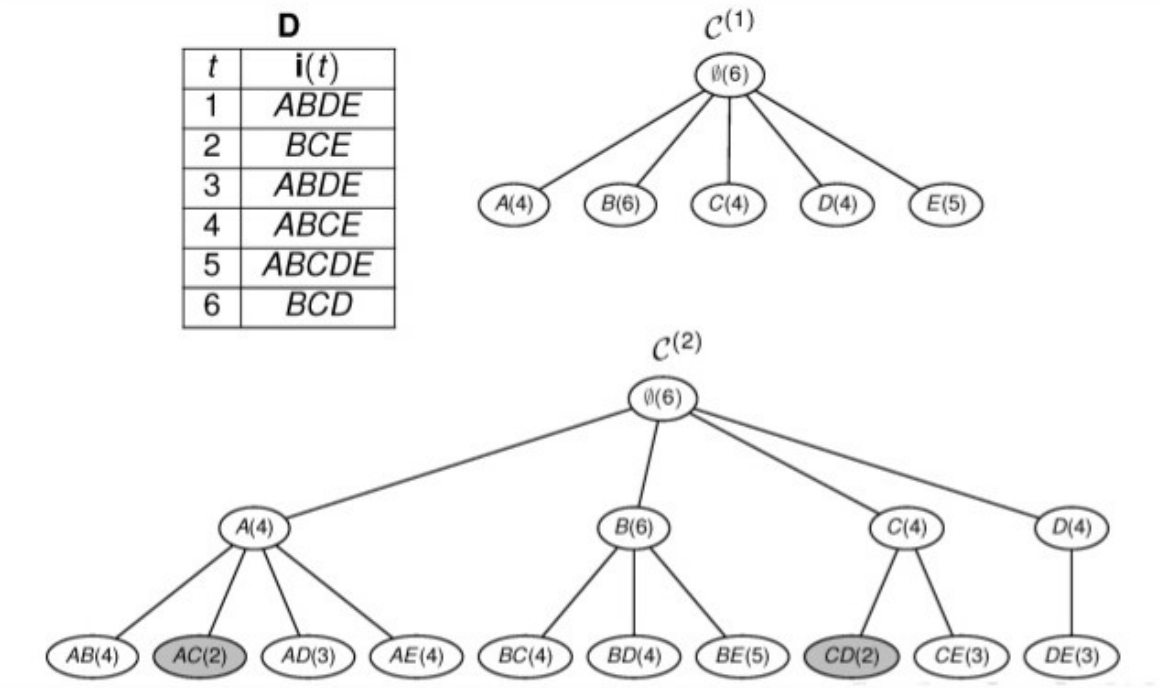

Figure 2.5: Esempio di esecuzione di due livelli di Apriori

transazioni, è l'algoritmo di Eclat, creato da Mohammed J. Zaki.
Le principali problematiche di questo metodo sono la generazione dei candidati, molto meno dispendiosa del metodo di forza bruta ma comunque pesante, e i continui controlli nel database per calcolare il supporto di ogni itemset. Per evitare questo genere di problema, è stato introdotto FPGrowth.

2.2.3 FPGrowth

Questo algoritmo, creato da Jiawei Han, Jian Pei, e Yiwen Yin, necessita di una struttura dati apposita, l'FPTree, da cui poi si estraggono direttamente i frequent itemsets, senza la generazione dei candidati.
Per la creazione di questo albero è necessitario riordinare le transazioni in ordine decrescente, a partire dall'elemento più frequente. A partire da una radice, rappresentante l'insieme vuoto, vengono poi lette le transazioni. Ognuna di queste sarà un cammino dell'albero: il primo elemento sarà figlio della radice,

il secondo figlio del primo, il terzo del secondo e così via. Se ci sono più transazioni che condividono un cammino, se ne tiene semplicemente un conteggio: saranno queste parti comuni dell'albero i frequent itemsets.

La proprietà migliore dell'FPTree è che percorrendolo si trovano le stesse

t	i(t)
1	ABDE
2	BCE
3	ABDE
4	ABCE
5	ABCDE
6	BCD

t	i(t)
1	BEAD
2	BEC
3	BEAD
4	BEAC
5	BEACD
6	BCD

I(t)	Frequenza
B	6
E	5
A	4
C	4
D	4

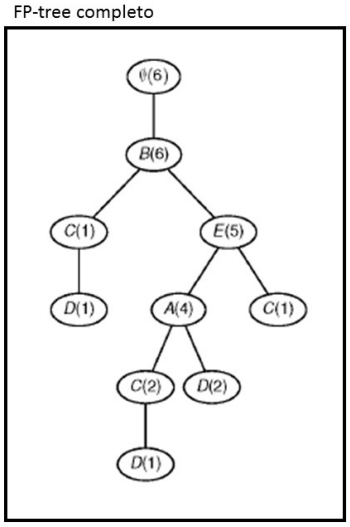

Figure 2.6: FPTree

informazioni delle transazioni, ma poste in maniera più efficiente: gli elementi con supporto minore infatti saranno le foglie, e spesso non sarà necessario, per l'estrazione dei frequent itemsets, nemmeno leggerle. Inoltre, questa struttura dati, una volta creata, può essere facilmente aggiornata con una nuova transazione, e non richiede ulteriori accessi al database, presentando al suo interno anche il supporto senza bisogno di conteggi.
Questo approccio è molto efficiente, in termini di scalabilità sia per quanto riguarda il supporto che per il numero di transazioni.

2.2.4 Generazione delle regole

Una volta ottenuti i frequent itemset, il passo successivo è utilizzarli per la generazione delle regole di associazione. Il metodo standard è quello di, per ogni itemset trovato X, di provare, per tutti i sottoinsiemi propri e diversi dall'insieme vuoto Y, le regole $\{Y\} => \{X \setminus Y\}$, di calcolarne la confidenza (come rapporto tra le transazioni che contengono X e quelle che contengono solamente Y) e vedere quali soddisfano la condizione, eventualmente fermandosi prima di provarle tutte, visto che anche la confidenza delle regole ricavate dallo stesso itemset godono della proprietà anti-monotona.
Le problematiche dipendenti dalla scelta delle soglie di supporto e confidenza portano comunque, in alcuni casi, a una computazione lenta e dispendiosa. Un'alternativa, implementata anche in diversi tool di data mining, è l'algoritmo

TopKRules, introdotto da Philippe Fournier-Viger, Cheng-Wei Wu e Vincent S. Tseng, che estrae solo un numero k di regole del massimo supporto possibile e con confidenza superiore alla soglia minima prefissata.

2.2.5 Tool e tecnologie

Essendo una branca molto importante del data mining, l'estrazione di regole di associazione è presente in moltissimi tool per l'analisi dei dati, accanto alla classificazione, al clustering ecc.
E' senz'altro da citare Weka, software in Java e che fornisce anche API da incorporare in altri progetti, che dedica un pacchetto all'argomento contenente anche le implementazioni di Apriori e FPGrowth. Una particolarità di questo tool è il formato in cui richiede i dati, anche per altre forme di analisi, l'Attribute-Relation File Format (ARFF) che descrive le transazioni in termini di attributi e valore. Questa formattazione è diventata abbastanza popolare e, come vedremo in seguito, è accettata anche da altri software.
Un progetto molto attivo è quello di Apache Spark (scritto in Java, Python, R e Scala), grazie anche alla qualità di processare grandi quantità di dati in modo distribuito. All'interno della sua Machine Learning Library, anche questo possiede la sua implementazione di FPGrowth e anche di una struttura trasparente e utilizzabile dell'FPTree, che richiede dati semplicemente posti in un file tab-delimited. Essendo un progetto open source, è presente su Maven e può essere anche questo incorporato nelle applicazioni degli utenti.
Il software R, noto più che altro per la sua utilità e importanza in campo statistico, possiede anch'esso un pacchetto dedicato alle regole di associazione. Sviluppato in C e Fortran, R possiede anche l'omonimo linguaggio di scripting, con cui è possibile costruire strutture dati, effettuare calcoli, definire funzioni ecc. Proprio per questo, il software è quasi sempre utilizzato da solo e, nonostante sia presente un'interfaccia tra Java e il linguaggio R (rJava) che permette di inviare comandi, non è stato utilizzato negli esempi di questa tesi.
Il pacchetto di librerie per Java SPMF, realizzato dal canadese Philippe Fournier-Viger e in continuo aggiornamento, è specializzato nell'estrazione di frequent itemset, regole (oltre ai pattern sequenziali) implementando moltissimi algoritmi anche meno famosi, come appunto il TopKRules citato in precedenza. Per questo motivo, unita alla compatibilità con il formato ARFF e l'ampia documentazione disponibile, è stato utilizzato in alcuni esempi.

2.3 Regole di associazione per alberi

La possibilità di codificare informazione attraverso strutture ad albero (e quindi delle forme particolari di grafo) ha richiesto lo sviluppo tecniche di data mining ad hoc per la classificazione, il clustering e le regole di associazione. Tipicamente, la rappresentazione di dati di questo tipo avviene in XML e, di conseguenza, l'utilizzo di XQuery per interrogazioni e trasformazioni.

2.3.1 Concetti di base

Per la scoperta di regole di associazione in XML, è necessario identificare i tipici componenti (item, transazione ecc.) nella nuova forma, per poi reimplementare gli algoritmi già esistenti, o svilupparne di nuovi.

Una particolare importanza assume il concetto di contesto, ovvero quelle

Figure 2.7: Transazioni e item in XML

caratteristiche comuni di tutti gli item. Supponiamo di avere un documento di pubblicazioni, e di essere interessati solamente a ciò che è stato pubblicato dai membri dello staff dopo il 2006: queste caratteristiche fanno parte del contesto. In Figura 2.8 possiamo vedere un esempio di regola di associazione: i membri dello staff che hanno pubblicato in una certa rivista (attributo publi_where) avranno anche un progetto di un dominio specifico (attributo domain).

Il contesto è importante anche per i suoi effetti sul supporto e la confidenza: prendendo in esame i membri dello staff di tutti i dipartimenti, una regola avrà un supporto assoluto maggiore rispetto a quanto avrebbe selezionando solo le pubblicazioni del dipartimento di Fisica, tuttavia la confidenza nel caso generale potrebbe essere non sufficiente, e la regola potrebbe essere valida solo in alcuni casi particolari.

2.3.2 Transazioni in XML

Un metodo semplice per trovare regole di associazione per XML è stato proposto nel 2004 da Jacky W. W. Wan e Gillian Dobbie.

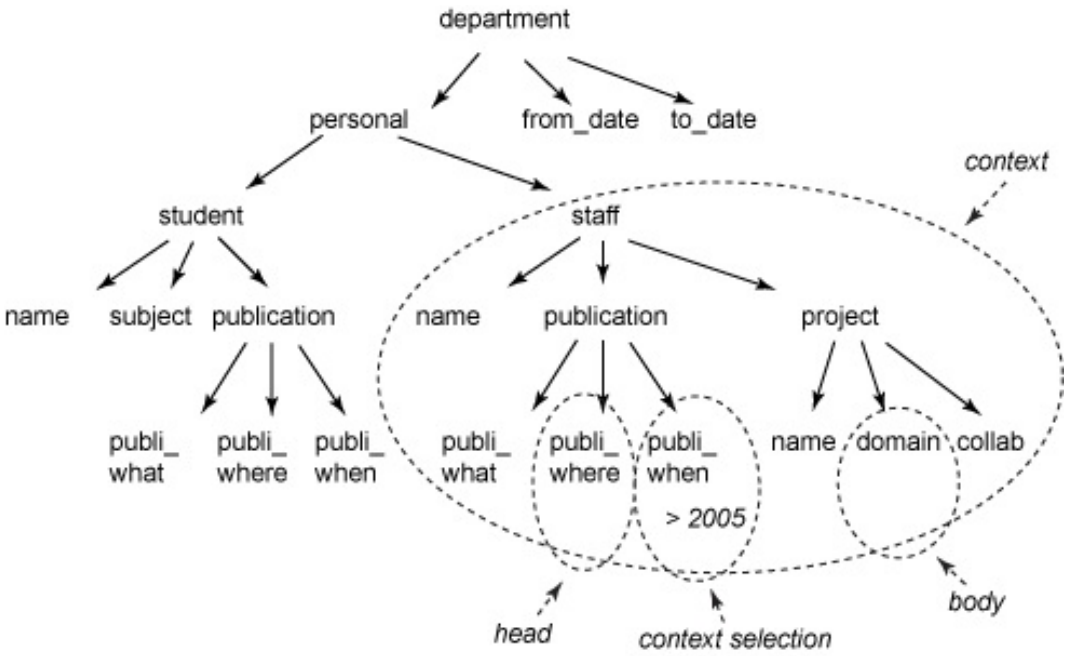

Figure 2.8: Regola di associazione e contesto

La procedura si divide in tre fasi:

- il preprocessing, ovvero la traduzione del generico documento in una forma standard, che espliciti transazioni e item (Figura 2.9), utilizzando le proprietà di selezione e trasformazione di XQuery, che permette di creare qualsiasi struttura XML;

- l'implementazione di un algoritmo per l'estrazione delle regole (ad esempio Apriori), sempre con XQuery utilizzandone le proprietà di linguaggio di programmazione vero e proprio;

- la creazione di una struttura XML per la rappresentazione dell'output, con nodi di tipo rule, attributi di supporto e confidenza, e gli item della parte antecedente e conseguente.

2.3.3 Tree-Based Association Rules

L'esempio della Figura 2.8 vede head e body della regola di associazione come singoli attributi. L'approccio con l'adattamento in transazioni prende in considerazione solamente oggetti semplici, come gli item a, b, c e d dell'esempio in figura 2.9, allo stesso modo di quanto farebbe Apriori per un database relazionale. L'estrazione di regole di associazione all'interno di strutture ad albero richiede tuttavia un'analisi più profonda, che comprenda anche il caso di interi sottoalberi che ne implichino altri.

Una Tree-Based Association Rule (TAR) è quindi una regola di associazione tra due sottoalberi. Nel primo esempio di implicazione della Figura 2.10, ad

```
<transactions>
        <transaction id="1">
                <items>
                        <item> a </item>
                        <item> d </item>
                        <item> e </item>
                </items>
        </transaction>
        <transaction id="2">
                <items>
                        <item> b </item>
                        <item> c </item>
                        <item> d </item>
                </items>
        </transaction>
        <transaction id="3">
                <items>
                        <item> a </item>
                        <item> c </item>
                </items>
        </transaction>
        <transaction id="4">
                <items>
                        <item> b </item>
                        <item> c </item>
                        <item> d </item>
                </items>
        </transaction>
        <transaction id="5">
                <items>
                        <item> a </item>
                        <item> b </item>
                </items>
        </transaction>
</transactions>
```

Figure 2.9:

esempio si verifica che ogni nodo A abbia un figlio di tipo B.
Esistono due tipi di TAR:

- iTAR(instance TAR), ovvero regole di associazione che tengano conto sia della struttura che del contenuto (PCDATA) dei nodi come nell'ultimo esempio della Figura 2.10;
- sTAR(structure TAR), in cui si analizza solamente la struttura.

L'estrazione di TAR richiede l'estrazione dei sottoalberi frequenti, con algoritmi quali TreeMiner e FREQT (FREQuent Tree miner), e la verifica della confidenza delle regole, in modo del tutto analogo a quanto visto nella sezione 2.4.

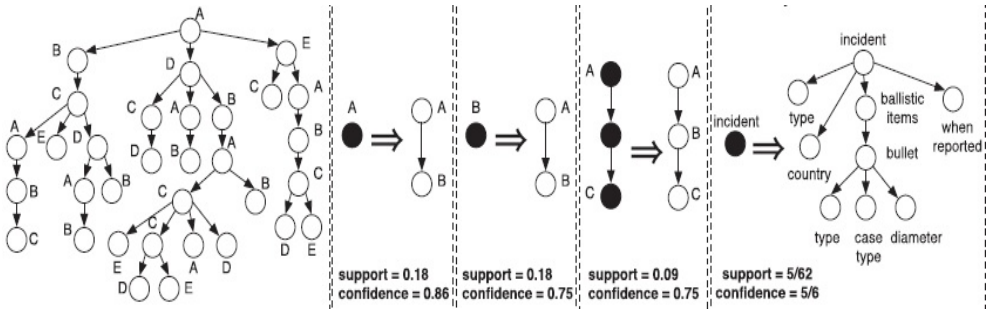

Figure 2.10: Esempi di regole di associazione tra sottoalberi

2.4 RDF e regole di associazione

In questa sezione, verranno esplorati alcuni legami già esistenti tra il linguaggio RDF e le regole di associazione.

2.4.1 RuleML

RuleML è un linguaggio basato su XML, e ne rispetta tutte le regole base, per esprimere le regole di associazione in una forma più strutturata rispetto a quella classica {head} =>{body}. La regola, per esempio, "se in un film compare Harpo Marx, compare anche Groucho Marx", può essere espressa in questa maniera, in cui head è la parte di ipotesi, body è la tesi e ogni atom è un elemento, in cui è possibile trovare una proprietà e un valore.

```
<Implies>
    <head>
        <Atom>
            <Rel>http://dbpedia.org/ontology/starring</Rel>
            <Var>http://dbpedia.org/resource/Harpo_Marx</Var>
        </Atom>
    </head>
    <body>
        <Atom>
            <Rel>http://dbpedia.org/ontology/starring</Rel>
            <Var>http://dbpedia.org/resource/Groucho_Marx</Var>
        </Atom>
    </body>
</Implies>
```

RuleML è perfettamente interoperabile a RDF per cui le regole di associazione possono essere espresse attraverso un modello RDF.

2.4.2 Semantic Web mining

Il problema di trovare informazione utile dal web semantico è oggetto di studio già da qualche anno, spesso mettendone in luce pregi, difficoltà e sfide future. Nonostante la possibilità di avere dati con una struttura uniforme, a differenza di database scollegati e da integrare, le problematiche sno molteplici, come l'adattamento degli algoritmi tradizioni di data mining e la complessità della struttura dati (in cui ogni tripla definisce un possibile fatto interessante). I campi di applicazione del semantic web mining riguardano anche il riconoscimento di un contesto di informazione di alto livello dati semplici ed eterogenei, su cui si possano effettuare ragionamenti logici, oppure possono avere applicazioni nei motori di ricerca, adattando maggiormente le query poste dagli utenti ai risultati forniti.
Direttamente centrato sulle regole di associazione ed RDF lo studio di Nebot e Berlanga, che propone inoltre un'estensione di SPARQL per l'estrazione diretta. I concetti principali, ripresi anche in un articolo di Ziawash Abedjan e Felix Naumann, sono quelli di target, cioè la risorsa sotto analisi, predict ovvero la proprietà implicata e di contesto dal quale la regola viene estratta. Nell'esempio dell'articolo, si cerca per i pazienti (target), la malattia (predict) in base ai dati relativi agli esami (contesto).

2.4.3 Grafi

Considerando l'RDF come una rappresentazione dei dati attraverso un grafo, è possibile trovare alcuni algoritmi che possono estrarre sottografi frequenti. Uno di questi, presentato nel saggio, si basa proprio su Apriori o FPGrowth e sull'isomorfismo tra grafi.
Nel caso più specifico di sottoalberi frequenti, anche l'algoritmo TreeMiner genera i candidati attraverso una variante di Apriori, mentre il supporto viene calcolato con tecniche di clustering.
Mentre in questi casi (e nel modello RDF), il grafo è la struttura da cui partire nell'estrazione di conoscenza utile, in SemGrAM di John F. Roddick e Peter Fule, il grafo è un modo di rappresentare la distanza concettuale tra gli oggetti dell'analisi, rilevando sinonimi perfetti, concetti simili e integrando queste informazioni nella struttura FPTree da cui ricavare le regole di associazione con FPGrowth. I componenti degli insiemi coinvolti dalla regola non sono più solo semplici valori, ma diventano gruppi di concetti simili (fino ad una certa soglia), che quindi potranno implicare altri gruppi. Questi sono rappresentati come sottografi, in cui i lati hanno costo pari alla distanza concettuale.

3. Estrazione delle regole

In questa sezione vengono enunciati i problemi connessi all'estrazione di regole di associazione da dataset RDF e si presenta la soluzione ideata. Alcuni concetti, qui illustrati in maniera solamente intuitiva, troveranno una specifica maggiore nel capitolo dedicato alla classificazione delle regole estratte. Analogamente, gli algoritmi e le query che costituiscono la metodologia adottata saranno analizzati più in dettaglio successivamente.

3.1 Definizione del problema

I dati di ingresso presentano alcune difficoltà di trattamento, per via di fattori quali:

- le grandi dimensioni delle fonti e la presenza di sottografi inutili all'analisi, che rendono il trattamento dei dati meno efficiente e molto più dispendioso in termini di tempo e memoria;
- la presenza di diverse fonti specializzate in un particolare argomento, e la necessità di incrociare dati da diversi dataset, con i limiti pratici della pubblicazione dei Linked Open Data;
- il formato, che rende impossibile l'utilizzo diretto degli algoritmi classici per l'estrazione diretta delle regole;
- la semplicità delle relazioni soggetto-predicato-oggetto, e quindi l'utilizzo di diverse triple per definire un concetto potenzialmente utile nel data mining;
- l'eterogeneità di grafi e modelli, che rendono difficile la progettazione di algoritmi generali.

Dati di questo tipo vanno quindi necessariamente ridotti, trasformati, rimodellati, al fine di poter riconoscere in essi gli elementi chiave della ricerca di regole di associazione, quali transazioni, itemset ecc. Queste operazioni, svolte nelle prime cinque fasi della metodologia, devono avere una validità quanto più generale possibile, in modo da essere adattabili a diversi casi, e per questo motivo è necessario trovare alcuni grafi notevoli, caratteristiche comuni e vincoli standard da porre nelle query SPARQL, da applicare poi ai singoli esempi con le risorse e proprietà specifiche.

Avendo concetti complessi, per un'analisi completa si rende anche necessario un metodo per isolare e riconoscere i sottografi che li rappresentano, in modo tale da analizzarli come un elemento singolo e senza perdita di informazione.
Il problema quindi consiste nel realizzare un sistema che permetta il passaggio dai vasti ed eterogenei dataset RDF a una forma ben strutturata, che possa fare da input per i tool di data mining (fasi 6 e 7), direttamente o tramite piccole modifiche, e ottenere le regole di associazione (fase 8), in una forma leggibile e a cui si possa attribuire un significato, prodotta nelle ultime due fasi.

3.2 Estrazione delle regole

La proposta di soluzione consiste nello sviluppo di una metodologia che modifichi, con una sequenza di fasi e configurazioni intermedie ben definite, il grafo del dataset iniziale, possa riconoscere e renda utilizzabili i concetti formati da più risorse e proprietà e venga poi tradotto nell'input (file tab-delimited, ARFF ecc.) necessario al tool di data mining. Le operazioni sul grafo sono svolte in SPARQL, in special modo con le query construct che permettono non solo di selezionare alcune parti del grafo ma anche di rimodellarlo a proprio piacimento, mentre la traduzione nei dati d'ingresso per gli algoritmi di estrazione richiedono un linguaggio di programmazione vero e proprio, in questo caso Java, che possa leggere i risultati delle query e riorganizzarli nel modo più opportuno
Come illustrato nella Figura 3.1, il progetto consiste in diversi passaggi (illustrati nel dettaglio di query e codice nel capitolo 5):

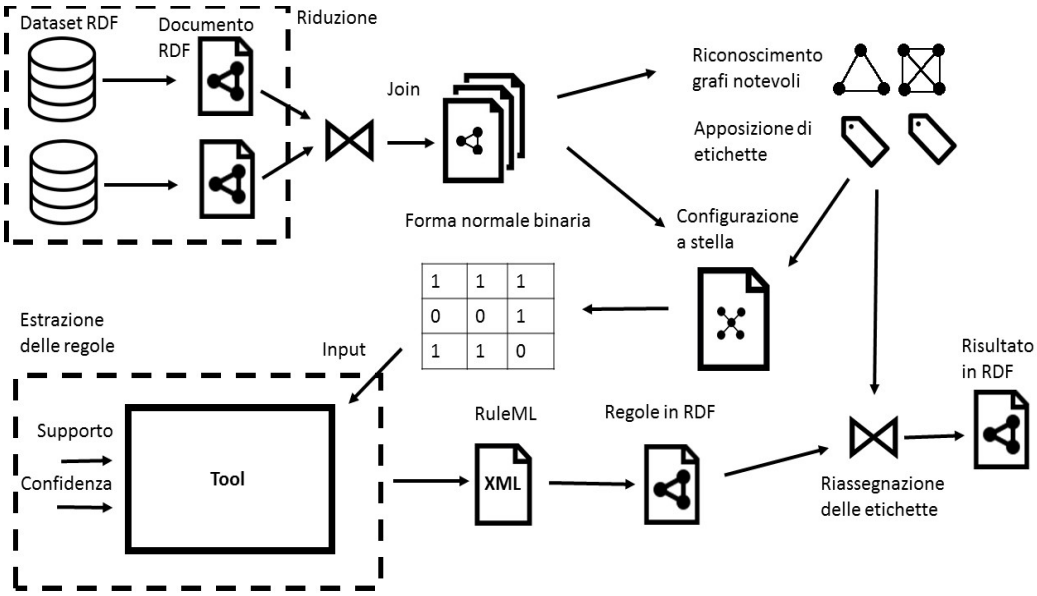

Figure 3.1: Schema della metodologia

1. **Riduzione**: questa fase di pre-processing crea sostanzialmente una copia del dataset contenente solamente le informazioni interessanti, scartando proprietà inutili, interi sottografi che trattano di altri argomenti, risorse che non corrispondono ai requisiti. Per distinguere ciò che può essere utile all'analisi da ciò che non lo è, è necessaria una preliminare esplorazione dei dati, ma in questo modo è possibile passare da un'intera base di dati, la cui lettura e selezione potrebbero essere molto dispendiose, a un documento più contenuto, che tuttavia presenterà ugualmente ciò che interessa all'utente senza nessuna modifica. Se i dati sono presenti in rete, accessibili tramite un endpoint ben performante, questa operazione non è strettamente necessaria;

2. **Join**: nel caso in cui si abbia a che fare con più dataset, la riduzione viene svolta su ognuno di questi, unendo poi le informazioni attraverso una construct SPARQL tra le diverse fonti. Il modello risultante presenta quindi un grafo completo di tutto ciò che l'utente ritiene utile all'analisi, nella forma derivata dalla configurazione iniziale;

3. **Riconoscimento di configurazioni notevoli**: questa fase permette di isolare sottografi, formati da più risorse e proprietà ma che esprimono un concetto unitario, che si ritengono utili all'analisi; questa parte è ancora svolta con una construct, che rilevi nella parte where la presenza del grafo e lo riassuma nella parte di redifinizione in una risorsa singola, da legare poi al resto del modello. Nel capitolo 4 sono presenti alcuni di essi (come la clique, grafo in cui tutti gli elementi sono il relazione l'uno con l'altro), ma il procedimento è libero e adattabile in base alle esigenze del caso in esame;

4. **Apposizione delle etichette**: vengono assegnate **etichette**, sotto forma di proprietà RDF, alle risorse che riassumono i sottografi notevoli riconosciuti nella fase precedente, in modo tale da poter integrare informazioni che non possono essere rappresentate solamente con un IRI e che potrebbero essere utili alla comprensione delle regole;

5. **Configurazione a stella**: viene svolta un'ulteriore trasformazione che porti il grafo dalla forma qualsiasi alla configurazione a stella, in cui la risorsa centrale (che può essere anche l'IRI di un sottografo precedentemente riassunto) è collegata con altre solo direttamente, con una sola proprietà e senza passare attraverso nodi intermedi. La realizzazione di una forma di questo tipo tipo richiede sia la gestione delle proprietà interessanti raggiungibili con un cammino composto da più di un lato, sia la risoluzione di questioni legate alle caratteristiche dimensionali (ad esempio quella geografica di comune, provincia, regione, nazione o continente);

6. **Realizzazione della forma normale binaria**: la configurazione a stella viene tradotta, con un algoritmo generalizzato, in una matrice binaria, in cui ogni colonna rappresenta una coppia proprietà-valore, mentre

le righe sono le singole stelle; il valore 1 si assegna se nella stella è compresa la coppia, 0 invece se è assente. Per questa traduzione non è più necessario solamente SPARQL, comunque utilizzato per la lettura dei valori, ma anche il trattamento di stringhe e strutture dati. La forma normale binaria presenta tutte le informazioni contenute nella configurazione a stella, che a sua volta è stato realizzato curando che tutto ciò che è potenzialmente utile a fini di data mining, ma senza la scomodità della struttura a grafo.

7. **Creazione dell'input**: la forma normale binaria, o una sua versione modificata in forma di transazioni, costituisce l'input per i software che forniscono gli algoritmi Apriori e FPGrowth;

8. **Estrazione delle regole**: l'algoritmo scelto viene eseguito, producendo in output le regole di associazione nel formato specifico del software (file testuale, lista di oggetti, insieme di stringhe);

9. **Traduzione in RuleML**: le regole di associazione vengono tradotte in una forma standard in XML, secondo le specifiche di RuleML;

10. **Trasposizione in RDF e riassegnazione delle etichette**: per rendere più agevole un'analisi automatica, alla luce delle informazioni precedentemente selezionate nel dataset e inserite nelle etichette, il modello in RuleML viene tradotto in RDF, producendo quindi un documento del tutto analogo a quello di input e che potrebbe essere riprocessato da questo stesso sistema.

3.3 Un esempio completo

Illustriamo subito la metodologia attraverso un esempio al tempo stesso piccolo e completo, in modo tale da coprire le diverse fasi, e alcune insidie comuni in questo genere di operazioni, riportando il codice SPARQL delle trasformazioni e l'evoluzione dei modelli (specialmente in forma grafica) dalle prime interrogazioni verso le fonti tramite endpoint al file RDF finale.
Come si può immaginare, figure ben riconoscibili nei dataset possono essere quelle familiari. Se pensiamo al classico nucleo familiare, con padre, madre e un figlio è facile capire come può essere rappresentato in triple: 3 risorse, due di queste unite dalla proprietà matrimonio e l'altra con i vincoli genitoriali verso le altre. Le famiglie storicamente meglio documentate sono quelle reali: questioni come la linea di successione al trono di un paese necessitano di avere tutta una serie di informazioni riguardanti matrimoni, figli, fratelli e simili.
L'esempio in questione riguarda proprio questa situazione e prende le triple riguardanti il Regno di Francia, tra l'anno 1000 e il 1789. Si sono succedute tre diverse dinastie: i Capetingi, i Valois e i Borbone ed è logico pensare che ogni re abbia pensato alla sua successione, sposandosi e avendo un figlio maschio che possa prendere il suo posto, formando quindi un proprio "triangolo familiare" che possa essere isolato all'interno del dataset.

Un altro aspetto spesso documentato nella vita dei sovrani in genere è quello di data e luogo di nascita e di morte. Se la prima però è evidentemente unica per ognuno di essi, potrebbero esserci correlazioni tra i luoghi, vista la presenza di residenze di corte capaci di durare diverse generazioni: un re potrebbe essere nato nello stesso posto del suo predecessore, o altre relazioni simili.

3.3.1 Estrazione da DBPedia e Wikidata

Una buona fonte per questo genere di informazioni è DBPedia, anche per via del fatto che rappresenta i regnanti francesi con una classe apposita: KingsOfFrance, dell'ontologia Yago. Questa presenta le opportune proprietà di predecessore e successore di un re, la situazione familiare (la madre, il padre, con relativi sposi), le date di nascita e di morte e spesso anche i luoghi. Questi tuttavia non sono sempre presenti nella forma desiderata, nonostante siano conosciuti:a volte sono semplici stringhe, a volte si trovano nel testo di qualche altro attributo, a volte sono delle risorse create ad hoc, senza quindi le tipiche proprietà di regione, nazione ecc.
Per comprendere anche queste informazioni in maniera più strutturata, si ricorre a Wikidata, che è sì incompleta per quanto riguarda le proprietà familiari, ma fornisce una rappresentazione più standard dei dati geografici, collegandoli anche con GeoNames.

Avendo due dataset molto pesanti, e viste le difficoltà pratiche delle query

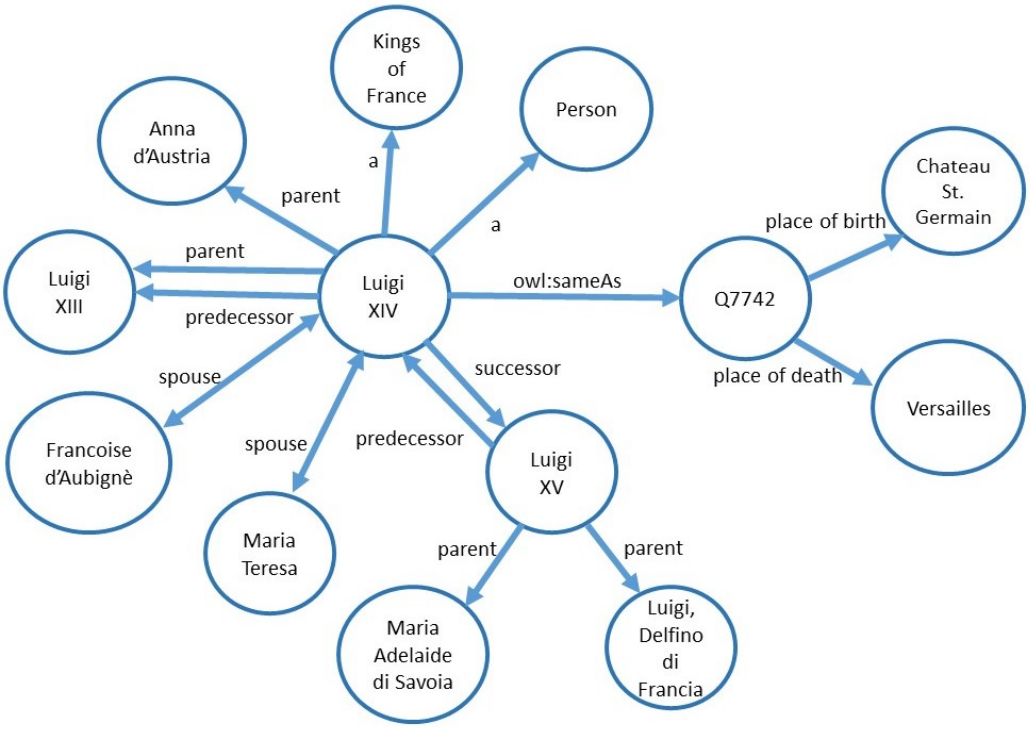

Figure 3.2: Collegamento tra DBPedia e Wikidata

attraverso due endpoint, è possibile recuperare tutte queste informazioni per

gradi. Innanzitutto, serve produrre un piccolo modello che contenga solamente le triple che servono di entrambe le fonti.
Partendo da DBPedia, queste sono le condizioni che servono per la formazione del triangolo familiare: la risorsa ?king deve possedere una moglie, un figlio da questa moglie di cui è sia padre che predecessore.

```
?king a <http://dbpedia.org/class/yago/KingsOfFrance>.
?king <http://dbpedia.org/ontology/deathDate>?date .
?king <http://dbpedia.org/property/spouse>?spouse .
?son <http://dbpedia.org/ontology/parent>?king .
?son <http://dbpedia.org/ontology/predecessor>?king .
?son <http://dbpedia.org/property/mother>?spouse .
```

Comprendiamo anche le informazioni riguardanti il re precedente e la data di morte, che poniamo compresa tra il 1000 e il 1789.

```
?king <http://dbpedia.org/ontology/predecessor>?predecessor .
?predecessor a <http://dbpedia.org/class/yago/KingsOfFrance>.
FILTER(year(?date) <1789)
FILTER(year(?date) >1000)
```

Infine, selezioniamo la proprietà owl:sameAs, che fornisce un collegamento alla stessa risorsa su altri dataset, fornendone un identificatore. DBPedia è connessa a moltissime fonti esterne, ma attraverso la clausola filter e le funzioni sulle stringhe possiamo selezionare solamente il collegamento a ciò che ci interessa, ponendo che l'IRI contenga la sottostringa "wikidata".

```
?king <http://www.w3.org/2002/07/owl#sameAs>?sameAs.
FILTER(contains(str(?sameAs), 'wikidata'))
```

L'altro modello ridotto comprenderà le informazioni relative ai luoghi di nascita e di morte di ogni sovrano. Su Wikidata esiste la proprietà "position held" (P39) tra la risorsa del re e quella che invece rappresenta la carica "re di Francia" (Q3439798), ma non è presente per tutti quelli estratti da DBPedia. Per evitare questo problema, creiamo un modello leggermente ridondante, comprendente anche tutti i nobili della casata dei Valois (Q182135), dei Capetingi (Q179544) e dei Borbone (Q3642350), con relativo luogo di nascita (P19) e di morte (P20).

```
CONSTRUCT{
    ?king a <http://myvocabulary.com/FrenchNoble>;
    <http://www.wikidata.org/prop/direct/P19>?birthPlace ;
    <http://www.wikidata.org/prop/direct/P20>?deathPlace .
} WHERE{
    FILTER(EXISTS {?king <http://www.wikidata.org/prop/direct/P39>
    <http://www.wikidata.org/entity/Q3439798>}
```

```
    OR EXISTS{?king <http://www.wikidata.org/prop/direct/P53>
    OR http://www.wikidata.org/entity/Q182135>}
    OR EXISTS{?king <http://www.wikidata.org/prop/direct/P53>
    <http://www.wikidata.org/entity/Q179544>}
    OR EXISTS{?king <http://www.wikidata.org/prop/direct/P53>
    <http://www.wikidata.org/entity/Q3642350>})
    ?king <http://www.wikidata.org/prop/direct/P19>?birthPlace .
    ?king <http://www.wikidata.org/prop/direct/P20>?deathPlace .
}
```

Entrambe le query vengono eseguite sui rispettivi endpoint, producendo due modelli che possono essere nuovamente interrogati, rielaborati, ma che prima di tutto vengono scritti su due file: replicaDBpedia.rdf e replicaWikidata.rdf.

3.3.2 Join e produzione del modello completo

Per unire le informazioni provenienti dai due dataset, serve un'ulteriore construct, fatta sui due documenti. Ripetiamo quindi le informazioni dei due modelli attraverso la ripetizione della tripla generica ?s ?p ?o sia nella parte where che nella parte construct, che riporta qualsiasi cosa presente in entrambi anche in quello appena creato, legando però anche i luoghi al rispettivo re.

```
CONSTRUCT {
    ?s ?p ?o .
    ?king <http://dbpedia.org/property/birthPlace>?b .
    ?king <http://dbpedia.org/property/deathPlace>?d .
}
FROM <replicaDBpedia.rdf>
FROM <replicaWikidata.rdf>
WHERE{
    ?s ?p ?o .
    ?king a <http://dbpedia.org/class/yago/KingsOfFrance>.
    ?king <http://www.w3.org/2002/07/owl#sameAs>?u .
    ?king <http://dbpedia.org/ontology/predecessor>?predecessor .
    ?predecessor a <http://dbpedia.org/class/yago/KingsOfFrance>.
    BIND(replace(str(?u),'http://wikidata.org/entity/','http://www.wikidata.org/entity/')
    AS ?url)
    BIND(iri(?url) AS ?link)
    ?link a <http://myvocabulary.com/FrenchNoble>.
    ?link <http://www.wikidata.org/prop/direct/P19>?b ;
    <http://www.wikidata.org/prop/direct/P20>?d .
}
```

Per effettuare il join, viene trovato il collegamento attraverso la proprietà owl:sameAs. In teoria, questo potrebbe essere utilizzato direttamente come

IRI della risorsa su Wikidata, ma in questo caso serve rimediare a un piccolo errore di inserimento. Le stringhe "http://wikidata.org/entity/" devono essere completate con il www in modo da ottenere il prefisso desiderato: "http://www.wikidata.org/entity/". Per poi trasformare la stringa in identificatore di una risorsa, si richiama la funzione iri. La variabile ?link risultante sarà quindi l'IRI del re nel documento replica di Wikidata, da cui si possono trovare tutte le informazioni del modello. I luoghi vengono quindi riassegnati direttamente alla risorsa DBPedia, creando un grafo completo.

3.3.3 Riconoscimento dei grafi notevoli

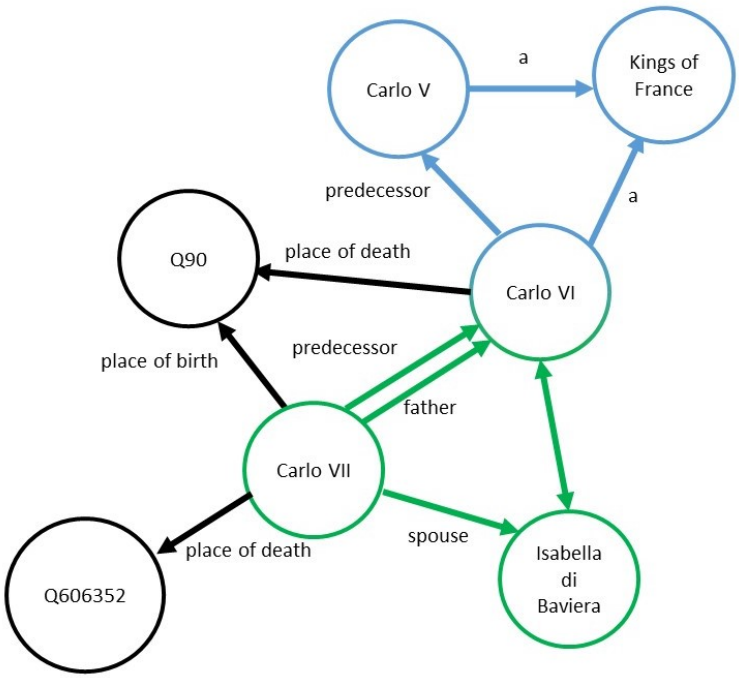

Figure 3.3: Triangolo familiare e di successione

Nella figura 3.3 è subito riconoscibile il triangolo familiare, ma è possibile trovarne anche un altro, il triangolo di successione, composto da un re, il suo predecessore e l'entità di tipo che accomuna entrambi, che può essere utile per le eventuali correlazioni tra luoghi tra una generazione e l'altra. Per isolare questi concetti, è necessaria un'ulteriore rielaborazione del modello RDF, con la creazione di risorse fittizie che rappresentino le figure.
Definiamo quindi un nuovo grafo, in cui a ogni triangolo di successione (del re attuale K2 e del precedente K1) possiamo legare il futuro triangolo familiare (con K2 e il futuro K3), secondo lo schema presente in Figura 3.4.
Il riconoscimento del triangolo di successione, valido per tutti i re in esame, è molto semplice:

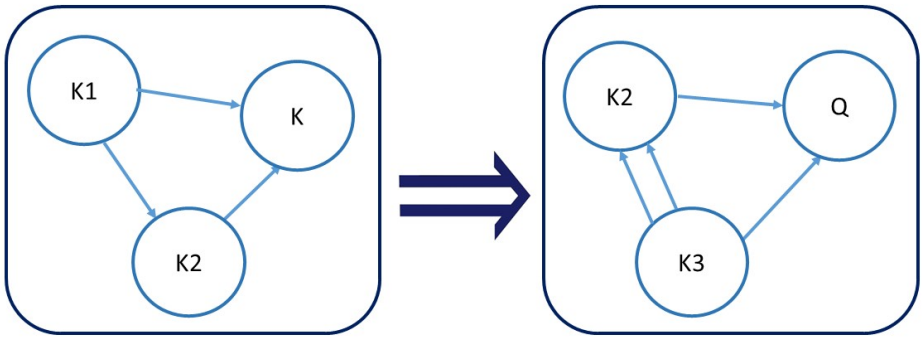

Figure 3.4: Regola grafo implica grafo desiderata

```
?king a <http://dbpedia.org/class/yago/KingsOfFrance>.
?king <http://dbpedia.org/ontology/predecessor>?predecessor .
?predecessor a <http://dbpedia.org/class/yago/KingsOfFrance>.
```

A queste condizioni devono corrispondere però le procedure per costruire la risorsa fittizia:

```
BIND(replace(str(?king),'http://dbpedia.org/resource/',"") AS ?i)
BIND(replace(str(?predecessor),'http://dbpedia.org/resource/',"") AS ?j)
BIND(concat("http://myvocabulary.com/succession",?i4, ?i1,"") AS ?s)
BIND(iri(?s) AS ?center)
```

Se, come in figura, si utilizzano le risorse http://dbpedia.org/resource/Charles_V_of_France e Charles_VI_of_France, l'IRI risultate è

```
http://myvocabulary.com/successionCharles_V_of_FranceCharles_VI_of_France
```

pronto per essere utilizzato nella parte construct della trasformazione. Può essere assegnata anche una prima etichetta, quella relativa ai componenti del triangolo:

```
CONSTRUCT{
    ?center a <http://myvocabulary.com/SuccessionTriangle>;
    <http://myvocabulary.com/comprende><
    http://dbpedia.org/class/yago/KingsOfFrance>;
    <http://myvocabulary.com/comprende>?king ;
```

<http://myvocabulary.com/comprende>?predecessor .

}

Il triangolo familiare è invece opzionale, per cui la construct dovrà necessariamente dividersi in due parti: una che riconosca i triangoli, e una che si occupi dei re che non li possiedono.
Nella prima avremo quindi:

?king <http://dbpedia.org/property/spouse>?spouse .
?son <http://dbpedia.org/ontology/parent>?king .
?son <http://dbpedia.org/ontology/predecessor>?king .
?son <http://dbpedia.org/property/mother>?spouse .

Con relativa costruzione della risorsa triangolo:

BIND(replace(str(?king),'http://dbpedia.org/resource/',"") AS ?i1)
BIND(replace(str(?spouse),'http://dbpedia.org/resource/',"") AS ?i2)
BIND(replace(str(?son),'http://dbpedia.org/resource/',"") AS ?i3)
BIND(concat("http://myvocabulary.com/",?i1, ?i2, ?i3,"") AS ?id)
BIND(iri(?id) AS ?iri)

Avendo a che fare con Carlo VI, Carlo VII e la regina Isabella di Baviera, avremo l'IRI Charles_VI_of_FranceIsabeau_of_BavariaCharles_VII_of_France, a cui verranno assegnate le relative etichette:

?iri a <http://myvocabulary.com/FamilyTriangle>.
?iri <http://myvocabulary.com/comprende>?son ;
<http://myvocabulary.com/comprende>?king ;
<http://myvocabulary.com/comprende>?spouse .

Ai due tipi di triangoli vengono anche assegnate la proprietà di omogeneità (no per quello di successione, essendoci una risorsa di tipo diverso, sì invece per il secondo che è composto solamente da persone) e il flag se tutte le proprietà sono bidirezionali.

<http://myvocabulary.com/FamilyTriangle><http://myvocabulary.com/omogeneo>1.
<http://myvocabulary.com/FamilyTriangle><http://myvocabulary.com/bidirezionale>0.
<http://myvocabulary.com/FamilyTriangle>a <http://myvocabulary.com/Triangle>.

Per le risorse invece che non hanno un triangolo familiare, occorre la seconda parte della construct, unita dalla clausola union, dove si esclude la presenza di mogli o figli che erediteranno il trono, del tutto identica a quella utilizzata nella trasformazione di riduzione.
Il risultato del riconoscimento di una serie di risorse di tipo SuccessionTriangle, alcune delle quali in relazione con alcuni triangoli familiari.
Le proprietà dei luoghi, siano essi del re o del predecessore, vengono diretta-

mente assegnate ad ogni ?center.

```
?center <http://dbpedia.org/property/birthPlaceKing>?b .
?center <http://dbpedia.org/property/deathPlaceKing>?d .
?center <http://dbpedia.org/property/birthPlacePredecessor>?b1 .
?center <http://dbpedia.org/property/deathPlacePredecessor>?d2 .
```

Dove le variabili che fanno da oggetto corrispondono a queste nella parte where, con clausola optional, dato che non sono sempre tutte presenti:

```
OPTIONAL {?king <http://dbpedia.org/property/birthPlace>?b }
OPTIONAL {?king <http://dbpedia.org/property/deathPlace>?d }
OPTIONAL {?predecessor <http://dbpedia.org/property/birthPlace>?b1 }
OPTIONAL {?predecessor <http://dbpedia.org/property/deathPlace>?d1 }
```

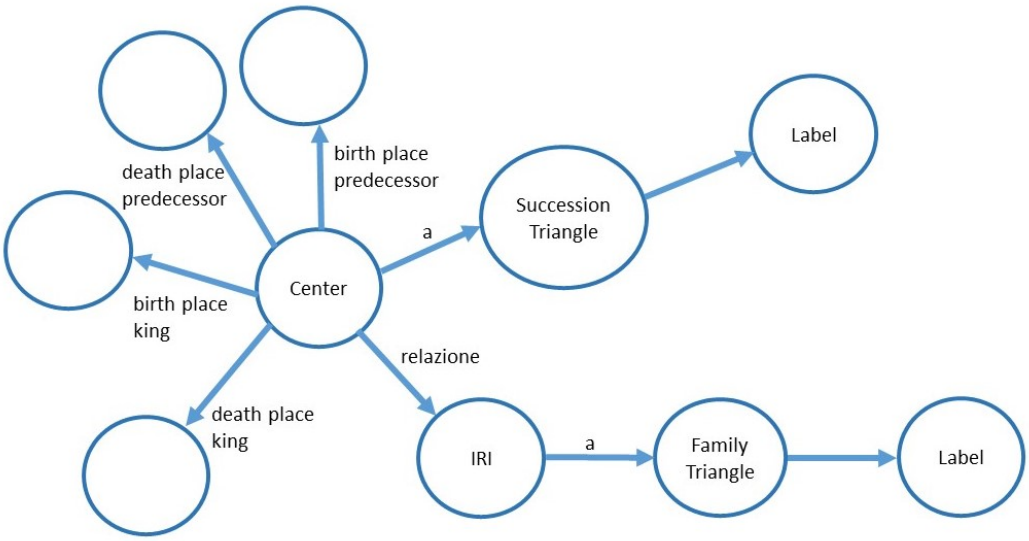

Figure 3.5: Grafo di una risorsa prima dell'avvicinamento

3.3.4 Avvicinamento ed elaborazione

Il modello risultante dopo il riconoscimento dei grafi notevoli ha una forma già abbastanza definita, ma non è ancora pronto per il data mining. I triangoli di successione sono infatti collegati con le risorse dei luoghi, senza che si possa ancora trovare una correlazione per cui un re sia nato nello stesso posto del suo predecessore, e con l'IRI del triangolo familiare, unico per ogni linea di successione.
Per algoritmi come Apriori o FPGrowth che fanno della frequenza degli elementi il concetto chiave, una proprietà che varia per ogni transazione non può chiaramente essere utile. Ciò che invece conta, per la ricerca delle regole di associazione, è che ogni risorsa fittizia abbia un tipo comune, e quindi si possa

trovare una regola che dato un generico elemento SuccessionTriangle questo sia connesso con uno FamilyTriangle, con un certo supporto e una confidenza. La proprietà rdf:type di ogni triangolo familiare viene quindi messa direttamente in relazione alla risorsa centrale, con una trasformazione di avvicinamento.

```
CONSTRUCT{
    ?center a <http://myvocabulary.com/SuccessionTriangle>.
    ?center <http://myvocabulary.com/tipoRelazione>?tipo .
    ?center ?prop ?val .}
WHERE{
{SELECT * WHERE{{
    ?center ?prop ?val .
    ?center a <http://myvocabulary.com/SuccessionTriangle>.
    ?center <http://myvocabulary.com/relation>?iri .
    ?iri a ?tipo . }
    UNION{
    ?center ?prop ?val .
    ?center a <http://myvocabulary.com/SuccessionTriangle>.
    FILTER(NOT EXISTS{?center <http://myvocabulary.com/relation>?iri})
    VALUES ?tipo {<http://myvocabulary.com/NoTriangle>}
}}} }
```

Assegnamo quindi al centro, oltre a quelle precedenti, la proprietà tipoRelazione, con oggetto la proprietà rdf:type dell'IRI con cui è già posto in relazione. Per quelle successioni che non hanno il corrispondente triangolo familiare, creiamo un tipo NoTriangle.

Una volta effettuata questa trasformazione, il risultato è già configurato a

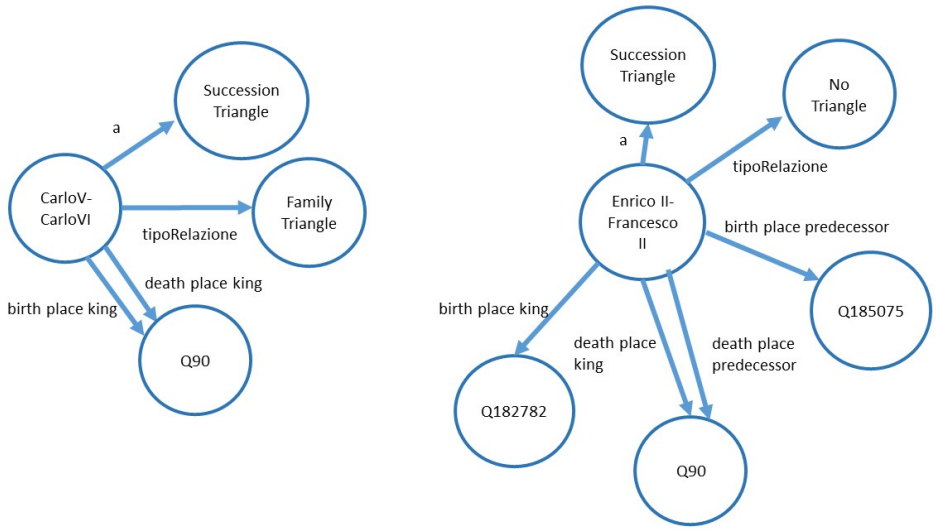

Figure 3.6: Due risorse dopo la trasformazione di avvicinamento

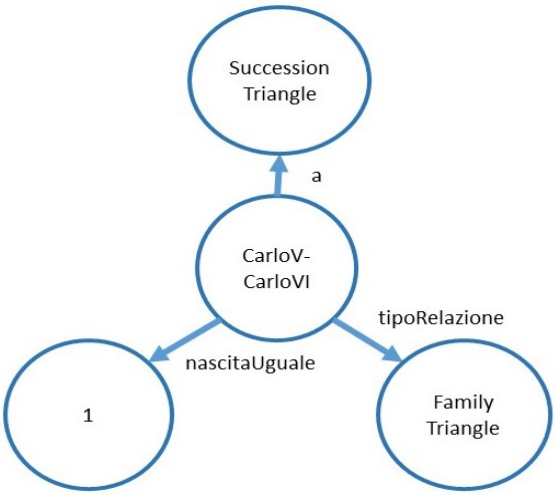

Figure 3.7: Risorsa dopo l'elaborazione

stella, ma va ancora perfezionato: è ancora presente l'IRI del triangolo familiare, informazione inutile, e non si è ancora fatto il confronto tra luoghi tra re e predecessore. E' necessaria quindi un'ulteriore construct:

```
CONSTRUCT{
    ?center a ?type .
    ?center <http://myvocabulary.com/tipoRelazione>?rel .
    ?center <http://myvocabulary.com/nascitaUguale>?val .}
WHERE{SELECT * WHERE{{
    ?center a ?type .
    ?center <http://myvocabulary.com/tipoRelazione>?rel .
    ?center <http://dbpedia.org/property/birthPlaceKing>?birth .
    ?center <http://dbpedia.org/property/birthPlacePredecessor>?b .
    FILTER(?birth = ?b)
    VALUES ?val {1}}
    UNION{
    ?center a ?type .
    ?center <http://myvocabulary.com/tipoRelazione>?rel .}
}}
```

Si assegna quindi un flag binario alla nuova proprietà nascitaUguale nel caso in cui re e predecessore, proprietà già assegnate direttamente al centro in una delle trasformazioni viste precedentemente, siano nati nello stesso posto. Per comprendere anche tutte le triple che non soddisfano questa condizione, vengono incorporate anche tutte quelle generiche dalla clausola union. Il nuovo modello avrà solo tre proprietà: il tipo, il tipo della relazione e il nuovo flag, senza informazioni ridondanti o inutili. E' chiaramente possibile, avendo anche il luogo della morte della maggior parte dei sovrani, provare altre coppie definendo altre proprietà.

3.3.5 Traduzione e regole di associazione

Dopo tutta questa fase, avremo finalmente una configurazione a stella che contiene tutte le informazioni desiderate. A questo punto, serve tradurlo in una forma che sia leggibile per i tool di data mining. Questa è la forma normale binaria: una matrice in cui sulle righe avremo ogni risorsa-triangolo di successione, sulle colonne invece le coppie proprietà-oggetto di ogni stella, che nel nostro caso saranno rdf:type-SuccessionTriangle, tipoRelazione-FamilyTriangle, tipoRelazione-NoTriangle, nascitaUguale-1. Nella matrice, viene messo valore 1 se nella stella formata dalla risorsa centrale e le sue proprietà è compresa anche la coppia, altrimenti 0. La traduzione, da RDF a una struttura dati

IRI	type=SuccessionTriangle	tipoRel=FamilyTriangle	tipoRel=NoTriangle	nascitaUguale=1
PhilipII-LouisVII	1	1	0	1

Figure 3.8: Riga della matrice corrispondente a Filippo II e Luigi VII

completamente diversa, non può più essere svolta semplicemente in SPARQL, ma necessita di un algoritmo sviluppato in Java e di cui saranno mostrati i dettagli successivamente.
La matrice risultante, se realizzata con l'ausilio delle librerie di Weka, può essere direttamente utilizzata per eseguire Apriori o FPGrowth sullo stesso tool e ottenere quindi le regole di associazione. Una di queste prova la correlazione tra triangoli di successione e quelli familiari, con confidenza del 74% (dei 23 casi asaminati, 17 sono favorevoli):

[rdf:type=SuccessionTriangle]=>[tipoRelazione=FamilyTriangle]

Un ulteriore algoritmo deve essere sviluppato per passare da questa forma, oggetti AssociationRule di Weka, allo standard RuleML in XML:

```
<Implies>
    <head>
        <Atom>
            <Rel>http://www.w3.org/1999/02/22-rdf-syntax-ns#type</Rel>
            <Var>http://myvocabulary.com/SuccessionTriangle</Var>
        </Atom>
    </head>
    <body>
        <Atom>
            <Rel>http://myvocabulary.com/tipoRelazione /Rel>
            <Var>http://myvocabulary.com/FamilyTriangle</Var>
        </Atom>
    </body>
</Implies>
```

Questo può essere ritradotto, creando risorse rappresentanti le diverse parti del documento XML (la regola, l'head, gli atom ecc.) e collegandole opportunamente, in un modello RDF, che può essere ricollegato con le etichette.

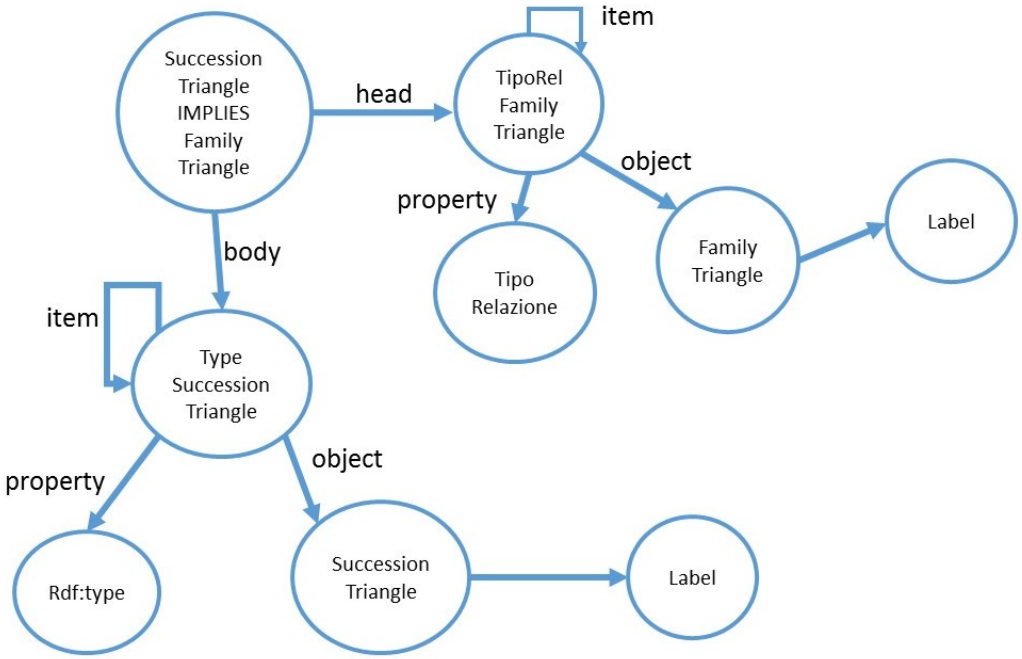

Figure 3.9: Modello RDF della regola di associazione

4. Paradigmi di regole su grafi

Questa sezione propone una classificazione delle possibili regole che si possono estrarre da un dataset RDF, delle trasformazioni da effettuare, dei sottografi notevoli che si possono riconoscere, presentando anche degli esempi pratici. A ciascuna tipologia corrisponde un template di query o un insieme di condizioni SPARQL, da inserire o completare in caso di applicazione.

4.1 Tipologie di regole

Prima di iniziare la ricerca di regole di associazione, è opportuno definire cosa si potrebbe trovare. Infatti, se nel classico modello con le transazioni si cercheranno occorrenze frequenti di valori semplici, per RDF è possibile anche trattare sottografi strutturati, che potranno implicare valori o altri grafi. La differenza tra le regole non sarà quindi solo nel numero o negli elementi che conterranno, ma anche in base alla topologia dei grafi coinvolti. Prima di entrare nel dettaglio dei singoli casi, viene qui proposta una classificazione generale.

4.1.1 Metriche di classificazione delle regole

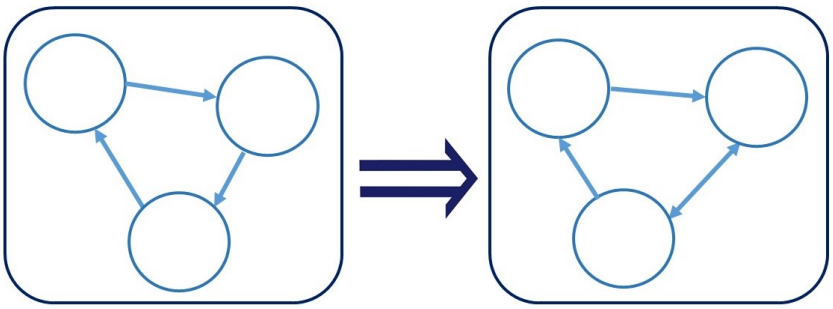

Figure 4.1: Esempio di regola triangolo implica triangolo

Utilizzando anche la teoria dei grafi, è possibile definire alcune metriche che coinvolgano le figure della parte antecedente e conseguente di una regola di associazione.
La prima di queste è l'isomorfismo: una regola è isomorfa quando coinvolge solamente grafi isomorfi, ovvero ci possa essere una funzione bijettiva dai nodi

di uno a quelli dell'altro, mantenendo gli stessi lati di collegamento. Nel nostro caso, una regola è isomorfa quando coinvolge lo stesso numero di risorse in tutti i grafi, e la stessa struttura di proprietà che le collegano tra loro.
La seconda metrica è l'omogeneità: una regola si dice composta da grafi omogenei quando, a prescindere dalla struttura, le risorse che compongono il singolo grafo sono tutte dello stesso tipo. Il tipo è definito dalla proprietà rdf:type, fondamentale sia nell'RDF che in SPARQL (tanto da avere una parola chiave "a" per abbreviarlo), e ha come oggetto una classe, definita dall'ontologia. In caso di presenza di diversi tipi (ad esempio, la risorsa "Isaac Newton" su DBPedia è sia una Persona che uno Scienziato), basta l'uguaglianza tra uno di questi.
Ponendo il caso più semplice, con gli insiemi antecedente e conseguente della

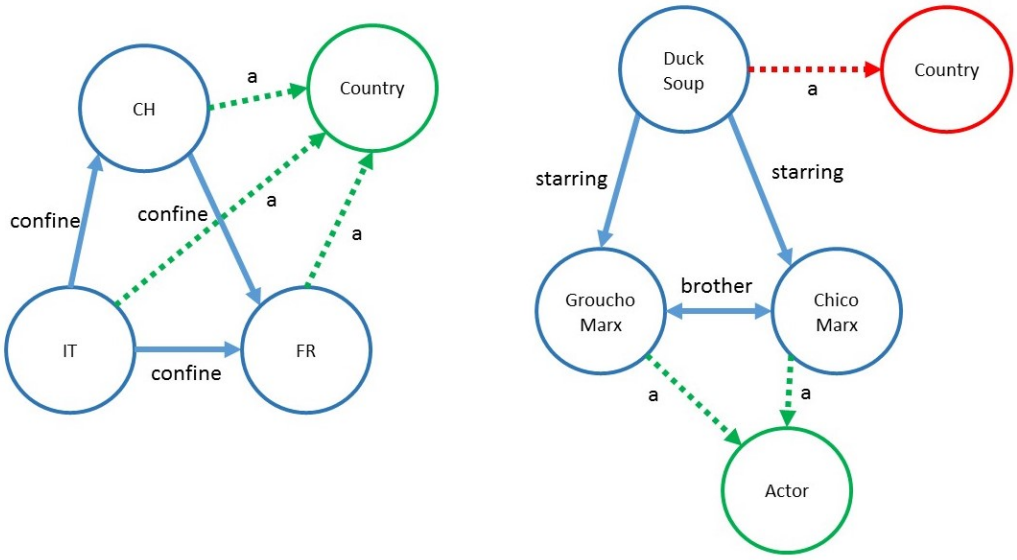

Figure 4.2: Grafo omogeneo e non omogeneo

regola formati da un solo elemento, possiamo quindi definire quattro combinazioni per la classificazione, con G e H due grafi generici:

Omogeneità / isomorfismo	OK	NO
OK	G →G omogenea	G →H omogenea
NO	G →G eterogenea	G →H eterogenea

4.1.2 Valore → valore

Una regola valore implica valore è la tipologia più semplice possibile: gli insiemi A e B di A → B sono composti da sole coppie proprietà-valore, con quest'ultimo che potrà essere una risorsa o un letterale. Si tratta di un caso particolare delle regole isomorfe, dato che i grafi hanno un solo elemento e quindi necessariamente la stessa struttura. Tuttavia, una regola di questo tipo

può essere omogenea o meno: una regola "Stan Laurel → Oliver Hardy" sarà omogenea, dato che entrambe le risorse sono Attori, una invece "Oliver Hardy → Commedia" no.
Di fatto, l'estrazione di questo tipo di regole è del tutto analogo a quanto viene fatto con le transazioni classiche e il modello relazionale, non essendoci nessun riconoscimento di qualche sottografo notevole. La procedura comprende la selezione delle risorse interessanti, cioè che abbiano alcune caratteristiche comuni (es. devono essere tutte di tipo "scontrino", devono essere delll'anno corrente ecc.) e delle proprietà che potrebbero produrre risultati.
In genere, è buona norma escludere quelle proprietà che abbiano un codominio molto esteso rispetto alle risorse, in quanto sarà improbabile che un valore superi la soglia minima di supporto. A questo proposito, è possibile interrogare direttamente il dataset attraverso un ASK che controlli il rapporto tra il dominio e codominio della proprietà:

```
ASK{
    SELECT AVG(?dominio) WHERE {
        {SELECT ?val (COUNT(DISTINCT ?res) AS ?dominio ) WHERE
        { ?res <property>?val . }
        GROUP BY ?val }
    } HAVING(AVG(?dominio)>SOGLIA)
}
```

La clausola ASK ritorna vero se esiste una tripla che soddisfi la seguente condizione: si contano, per ogni valore del codominio, il numero di risorse in relazione con esso; viene poi eseguita una media tra i conteggi, che deve superare una certa soglia. In questo modo, si verifica che ci siano più risorse collegate con un certo valore attraverso la proprietà, e che quindi il codominio sia inferiore al dominio. Ad esempio, studiando un dataset di album musicali, il genere è sicuramente una proprietà interessante.

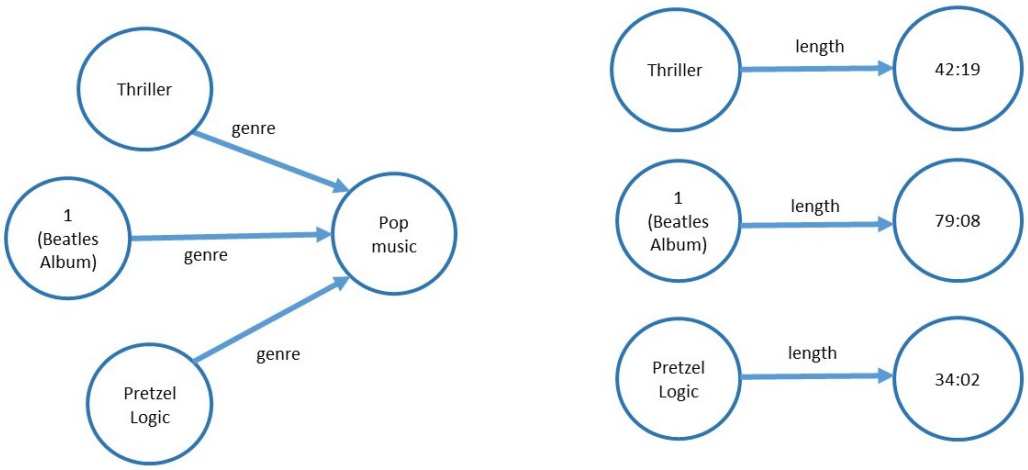

Figure 4.3: Proprietà aggreganti e non aggreganti

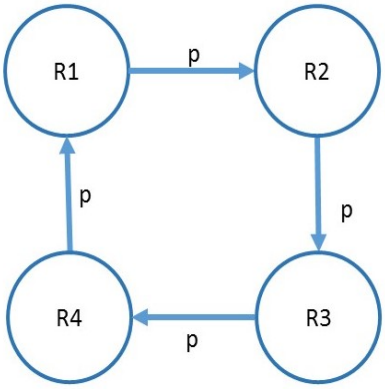

Figure 4.4: Quadrato

4.1.3 Grafo → grafo

Diversamente dal caso semplice, la generazione di una regola di questo tipo richiede una fase più complessa di pre-processing, mirata al riconoscimento di sottografi notevoli che potrebbero essere frequenti nel dataset. Queste figure saranno trovare attraverso l'inserimento nella query SPARQL delle condizioni che le costituiscano e costruendo risorse fittizie che le rappresentino. Ad esempio, nel riconoscimento di una figura a quadrato (come quella in Figura 4.4), formata da 4 risorse collegate dalla stessa proprietà, servirà inserire:

```
?r1 p ?r2 .
?r2 p ?r3 .
?r3 p ?r4 .
?r4 p ?r1 .
```

Manipolando gli identificatori delle risorse che compongono la figura, sarà possibile costruire una stringa che faccia da IRI per il sottografo notevole. Una maggiore specifica invece delle proprietà della figura sarà attribuita attraverso una serie di label, anche queste espresse in RDF. Prendendo per esempio un triangolo formato da tre stati confinanti (ognuno dei quali confina con gli altri due), come quello tra Italia (su GeoNames http://sws.geonames.org/3175395), Svizzera (2658434) e Francia (3017382), si può costruire un identificatore un IRI http://sws.geonames.org/2658434-3017382-3175395 unico e che possa rappresentare, in un nuovo modello RDF, l'entità triangolo, specificata come tipo della risorsa.
Una regola quindi di tipo grafo implica grafo, da un punto di vista esclusivamente pratico, sarà equivalente a una tra due valori, in cui però le risorse coinvolte saranno create ad arte e collegate con delle etichette che facciano ricostruire i grafi, dando quindi alla regola il suo pieno significato.

4.1.4 Grafo → risultato

Una regola grafo implica risultato è un altro caso particolare della regola grafo implica grafo (eterogenea ed eteromorfa), in cui però la parte conseguente è un valore derivato da operazioni riguardanti le proprietà delle risorse che costituiscono il grafo.
Poniamo il caso sempre di 3 stati confinanti: potrebbe essere ragionevole supporre che abbiano all'incirca le stesse condizioni economiche. Una regola per cui la presenza del triangolo implica la parità di queste condizioni è una regola grafo implica risultato. La proprietà è infatti il prodotto di una serie di operazioni (di confronto in questo caso, ma potrebbero essere matematiche e operando sul PIL per esempio) fatte su quelle singole delle risorse.
Anche qui, il grafo nella parte antecedente della regola sarà rappresentato da una risorsa fittizia, collegata con il valore riassuntivo attraverso una proprietà.

4.2 Configurazione a stella

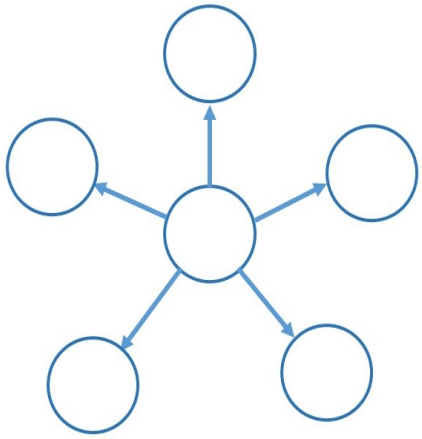

Figure 4.5: Configurazione a stella

La stella è composta semplicemente da una risorsa centrale, da cui si irradiano diverse proprietà. Queste possono avere un valore singolo (ad esempio, la data di nascita di una persona), così come uno multiplo (come gli attori di un film). I valori possono essere sia altre risorse, che potrebbero essere a loro volta il centro di altre stelle, oppure stringhe, date o valori numerici, che non possono avere proprietà.
Questa figura è la più semplice, infatti le proprietà si diramano direttamente dalla risorsa, senza nodi intermedi, esattamente come farebbero in un diagramma E-R dal rettangolo rappresentante l'entità. Proprio per questa somiglianza, è possibile facilmente passare dal modello a stella alla rappresentazione relazione: il centro è sicuramente un IRI e quindi un identificare univoco all'interno del dataset, mentre le proprietà sono gli altri attributi che compongono la tupla, con valori che possono essere di diverso tipo ed eventualmente, se sono altri identificatori, collegarsi con altre "tabelle".

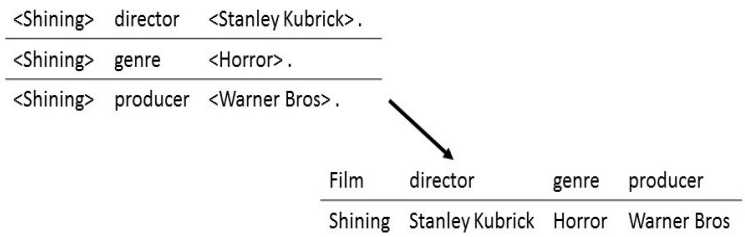

Figure 4.6: Dalle triple al modello relazionale

Le connessioni del modello a stella sono un sottoinsieme di tutte le possibili proprietà della risorsa centrale, e vengono dirette di primo livello, non avendo nessuna risorsa intermedia tra il centro e il valore finale.
Non sempre però, nell'interesse di chi analizza i dati, le proprietà interessanti si trovano subito al primo livello. Per esempio, cercando una correlazione tra la nazionalità di un campionato di calcio e quella dei suoi allenatori, capita che la risorsa relativa ad una squadra su Wikidata contenga solo il riferimento all'allenatore, che poi possederà la propria cittadinanza.
Nell'esempio, la risorsa "Netherlands" è di secondo livello, infatti è proprietà

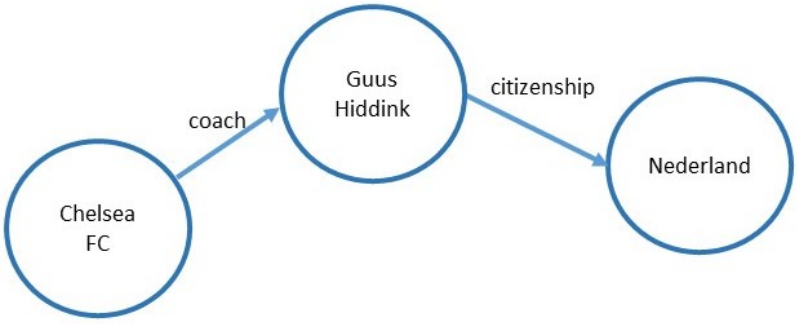

Figure 4.7: Squadra, allenatore e cittadinanza

di una di primo livello. Quella di terzo sarà connessa a uno di secondo, e così via. Proprio per la comodità del modello a stella, si esegue la trasformazione che porta al primo livello anche quelle che si trovano indirettamente, con una CONSTRUCT in SPARQL. Riportandoci quindi in questa forma, possiamo ottenere di nuovo facilmente una tupla tradizionale.
Il processo di trovare le regole di associazione quindi non è altro che un'analisi delle stelle: dato un insieme di risorse, con certe caratteristiche (per esempio, gli articoli riguardanti l'informatica), vengono trovare quali coppie proprietà-valore ne implicano altre. Per la composizione delle stelle quindi occorre:

- selezionare le risorse su cui si intende fare data mining;
- selezionare le proprietà interessanti;

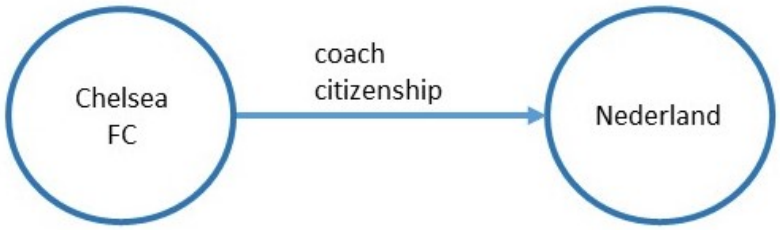

Figure 4.8: Squadra e cittadinanza dell'allenatore

- effettuare la trasformazione delle proprietà di livello superiore al primo, connettendole direttamente alla risorsa centrale.

4.2.1 Stelle e risorse

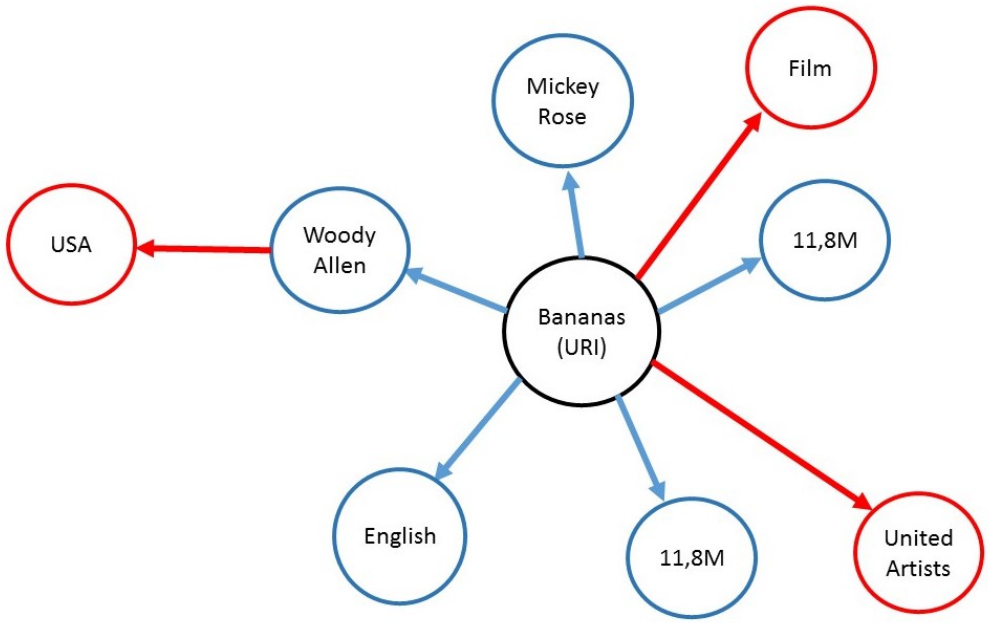

Figure 4.9: Proprietà comuni e interessanti

Nel linguaggio RDF, tuttavia, le proprietà sono tutte descritte allo stesso modo, nella stessa struttura a tripla. La selezione delle risorse in base a determinate caratteristiche quindi non è altro che imporre che alcune proprietà delle risorse prima di tutto esistano, e che abbiano un valore prefissato. Se vogliamo estrarre, come nella figura di esempio, tutti i film distribuiti dalla United Artists e che abbiano un regista statunitense, occorre imporre direttamente in SPARQL questa condizione, fissando le triple:

?film wiki:instanceOf <Film>.
?film wiki:distributor <United Artists>.
?film wiki:director ?director.
?director wiki:citizenship <United States>.

Queste condizioni, segnate in rosso nella Figura 4.9, non faranno parte della stella, in quanto non saranno di alcuna utilità per il data mining, essendo infatti comuni per tutte le risorse non aggiungeranno informazione utile. Quelle invece segnate in blu, come l'incasso, la lingua, l'URI del regista e del compositore della colonna sonora, sono quelle selezionate per trovare qualche correlazione inaspettata.
Un caso particolare di questa differenza tra proprietà comuni e quelle interessanti è la gerarchia, specialmente quella geografica. Prendiamo per esempio le caratteristiche dei cittadini, la proprietà di residenza non è semplicemente un URI di una singola risorsa, ma, utilizzando i Linked Open Data, il riferimento a un sistema amministrativo di suddivisione: il comune (poniamo il caso italiano) apparterrà a una provincia, che sarà sua volta contenuta in una regione, compresa in uno stato situato in un continente. Chiaramente, tutto dipende dall'organizzazione interna del dataset, se questo ha solamente una sua rappresentazione interna dei comuni, non sarà possibile delineare la gerarchia che porta fino al continente, se invece utilizzerà i riferimenti a GeoNames si potranno studiare le caratteristiche dei cittadini a diversi livelli geografici.
La gerarchia si può arrestare a seconda delle preferenze: se si vorrà studiare

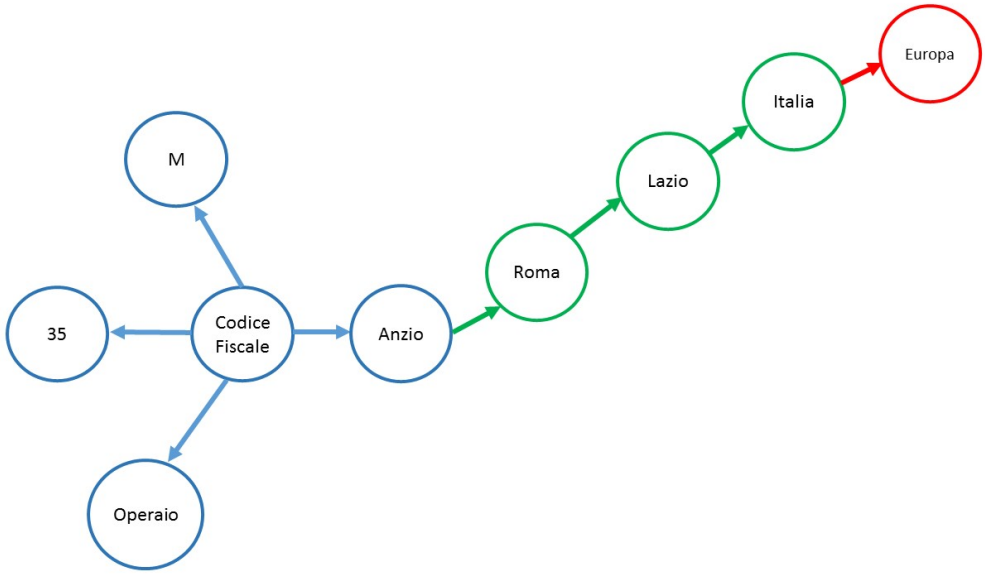

Figure 4.10: Gerarchia geografica

le caratteristiche dei cittadini del Lazio, non sarà necessario proseguire fino al continente, tenendo invece come elemento della stella il comune e la provincia.

4.2.2 Join

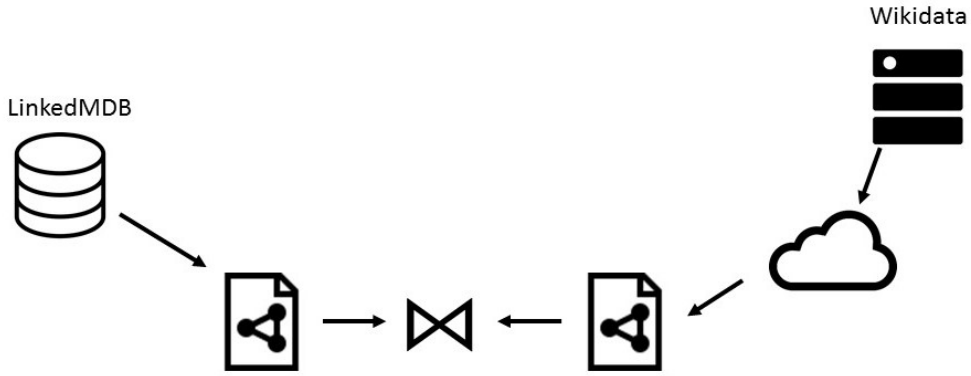

Figure 4.11: Join tra fonte locale e remota

La selezione delle risorse, la composizione delle stelle, spesso può richiedere l'accesso a diversi dataset. Nello spirito di RDF e dei LOD è improbabile che ogni knowledge base implementi ogni aspetto del dominio di conoscenza, bensì richiamerà al suo interno elementi di ontologie definite. Ad esempio, il dataset Muninn dedicato alla storia militare e deriva la propria struttura per le persone, i conflitti direttamente da DBPedia, e dove ci siano risorse in comune (quelle chiaramente più importanti) ne riporterà il riferimento.
La proprietà tipica per il cambio di knowledge base è owl:sameAs, che indica, per una risorsa soggetto, l'equivalenza con l'oggetto.
Per esempio, la risorsa del progetto Muninn dedicato al National War Memorial (Figura 4.12), monumento commemorativo canadese, è equivalente a quella di DBPedia, che a sua volta collega, con la proprietà owl:sameAs la pagina di Wikidata, di GeoNames (essendo comunque un luogo fisico), dell'ontologia Yago ecc.
Con ogni dataset che si può focalizzare su un aspetto diverso, definiamo come join lo strumento per integrare proprietà di ambiti diversi sulla stessa risorsa, in modelli intermedi su cui puoi eseguire altre trasformazioni.
Non sempre il riferimento è diretto: a volte per effettuare un join occorre manipolare proprietà comuni, lavorando a livello di stringa. Ad esempio, il database cinematografico LinkedMDB presenta le location in cui il film è girato in una propria soluzione interna, senza nessun riferimento alla gerarchia geografica, presente invece su Wikidata. Il join tra i due dataset non avviene attraverso la semplice proprietà owl:sameAs, ma sfruttando i riferimenti di entrambi al popolarissimo portale Internet Movie DataBase.
In SPARQL, questa operazione di join prende nomi e significati diversi a seconda se viene fatta in locale oppure attraverso endpoint. Con due dataset presenti sul disco, infatti l'operazione non è altro che una query normale, fatta su due file diversi, specificati attraverso la clausola FROM, che vengono interrogati come fossero uno singolo. Sta a chi interroga quindi fare in modo che l'intersezione logica funzioni. Per evitare di lavorare con enormi quantità di

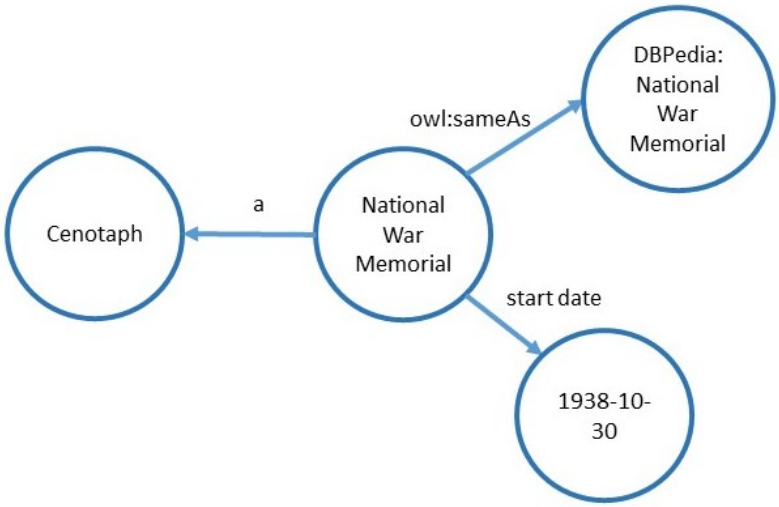

Figure 4.12: Muninn e collegamento a Wikidata

dati, spesso è utile fare prima una riduzione, trasferendo in locale i dati che servono dall'endpoint, o restringendo dump di diversi GB.

```
CONSTRUCT {
    ?film a <http://data.linkedmdb.org/movie/film>.
    ?film <http://data.linkedmdb.org/resource/movie/actor>?actor .
    rdfs:label ?label ;
    wikidata:location ?location ;
    wikidata:mainSubject ?sub .
}
FROM <linkedMDBInfo.rdf>
FROM <dbPediaInfo.rdf>
WHERE{
    ?film a <http://data.linkedmdb.org/movie/film>;
    <http://xmlns.com/foaf/0.1/page>?page ;
    rdfs:label ?label .
    <http://data.linkedmdb.org/movie/director>?director ;
    <http://data.linkedmdb.org/movie/actor>?actor ;
    <http://data.linkedmdb.org/movie/genre>?genre .
    //estraggo l'identificatore di IMDB dall'URI contenuto su LinkedMDB
    BIND(replace(str(?page),'http://www.imdb.com/title/','') AS ?label1)
    BIND(replace(str(?label1),'/','') AS ?imdb)
    ?wikiResource a wikidata:Film ;
    wikidata:imdbIdentifier ?imdb ;
    wikidata:location ?location ;
    wikidata:mainSubject ?sub .
}
```

Se invece si ha a che fare con due endpoint, senza passaggi intermedi, il join viene fatto attraverso una federated query, introdotte dalla versione 1.1 di SPARQL. Attraverso la parola chiave SERVICE, in queste interrogazioni è possibile specificare su quale endpoint controllare la tripla specificata. La query non va eseguita direttamente sull'endpoint ma preparata, con le sorgenti remote dei dati, e poi fatta eseguire dal motore di esecuzione (es. Jena) che si occuperà del join.

```
CONSTRUCT {
    ?film rdfs:label ?label ;
    <http://myvocabulary.com/starring>?actor ;
    <http://myvocabulary.com/genre>?genre ;
    <http://dbpedia.org/ontology/budget>?budget.
    <http://dbpedia.org/ontology/gross>?gross.
}
WHERE{
SERVICE <http://data.linkedmdb.org/sparql>{
    ?film a movie:film .
    ?film rdfs:label ?label .
    ?film owl:sameAs ?dbpediaLink .
    ?film <http://data.linkedmdb.org/resource/movie/actor>?actor ;
    <http://data.linkedmdb.org/resource/movie/genre>?genre .
}
SERVICE <http://dbpedia.org/sparql>{
    ?dbpediaLink <http://dbpedia.org/ontology/budget>?buget .
    ?dbpediaLink <http://dbpedia.org/ontology/gross>?gross .
}}
```

4.3 Configurazioni notevoli

In questa sezione, verranno selezionati alcuni particolari sottografi immediatamente riconoscibili, e come la loro ricerca può essere fatta in SPARQL. Questi, come tutti i sottografi che riterrà opportuno l'utente finale (stelle comprese) potranno essere riassunti in una risorsa fittizia, da utilizzare per la generazione delle regole grafo implica grafo o risultato.

4.3.1 Clique

In teoria dei grafi, una clique (o cricca, in italiano) è un grafo non orientato in cui tutti i nodi sono connessi tra loro. Nel nostro caso, avendo i lati rappresentati da proprietà che possiedono un soggetto e un oggetto e vengono comunemente disegnati con una freccia, non è possibile avere grafi non orientati, ma la definizione può essere allargata a questo caso.
Una clique viene quindi definita come un grafo orientato in cui tutte le risorse

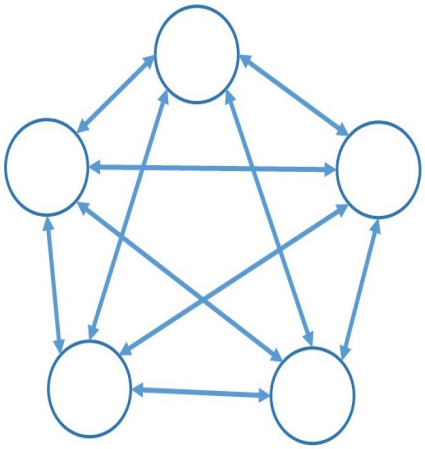

Figure 4.13: Clique

sono collegate tra loro da proprietà bidirezionali. Un esempio tipico può essere quello di alcune relazione di parentela: in una famiglia con molti fratelli, questi avranno la stessa proprietà "fratello" con tutti gli altri, esattamente alla pari. Non è necessario che la proprietà bidirezionale sia la stessa per tutta la clique, ma deve esserlo per ogni coppia di risorse. Dato l'insieme N dei componenti del sottografo notevole, per creare una clique servirà questa construct SPARQL:

CONSTRUCT{
    ?iri a <clique >
}
WHERE{
    $r_i \ p \ r_j \ \forall i, j \in [1, N]; \ i \neq j$
    BIND($f(r_1, \ r_2, \ ..., \ r_n)$)
}

Il significato di una clique è quello di una serie di risorse che hanno lo stesso rapporto di collaborazione, di parità, e quindi la ricerca di regole sarà orientata a risultati o proprietà che si realizzano ogni volta si instaura questa configurazione paritaria.

4.3.2 Hole

Anche questa definizione deriva dalla teoria dei grafi: una hole (o buca) è un sottoinsieme di nodi tali che il sottografo indotto da essi sia un ciclo, cioè un percorso che non passi mai due volte per lo stesso nodo. Il quadrato di esempio per la regola grafo implica grafo è una hole. Ponendo un insieme di n risorse {r1 - rn}, è necessario porre le condizioni sulle triple in modo tale che ogni risorsa sia in relazione con la successiva, tranne per l'ultima che l'avrà con la prima:

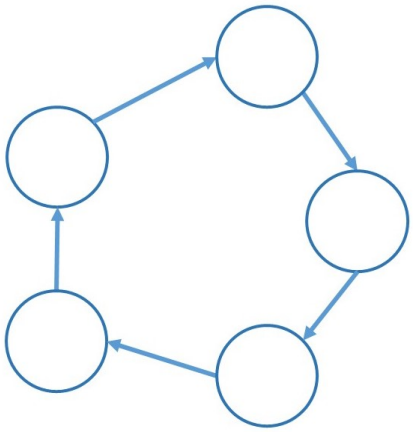

Figure 4.14: Hole

$r_1\ p\ r_2$
$r_2\ p\ r_3$...
$r_{n-1}\ p\ r_n$
$r_n\ p\ r_1$

SPARQL funziona in modo tale da selezionare tutte le triple che soddisfino solamente le condizioni di cui sopra, per cui ogni clique (in cui tutto si relaziona con tutto) sarà automaticamente anche una hole. In teoria dei grafi, una hole richiede nello specifico che ogni nodo sia in relazione solamente con precedente e successivo, e per evitare ciò è necessario porre condizioni supplementari:

FILTER(NOT EXISTS {
$r_i\ pr_j\ \forall i \in [1, n-1]\ j \neq i+1.$
$r_n\ pr_j\ j \neq 1$
})

Un esempio di hole può essere sia quello di cicli nelle reti dei trasporti o delle telecomunicazioni, oppure il susseguirsi di persone per un ruolo specifico: una carica, politica o amministrativa, può passare di mano in mano e, nel corso degli anni, essere occupata più volte dalla stessa persona.
Graficamente, una hole può essere composta da risorse anche completamente diverse tra loro, tuttavia nel nostro caso si parla solamente di nodi collegati dalla stessa proprietà, e quindi normalmente anche dello stesso tipo.

4.3.3 Triangoli

La differenza tra una clique e una hole, in generale, è immediatamente visibile, con la presenza o meno delle "diagonali" della figura piana formata dalle risorse. Non lo è invece quando si hanno solamente tre nodi nel grafo, dove differiscono solamente per l'orientamento delle frecce. Visivamente, siamo sempre

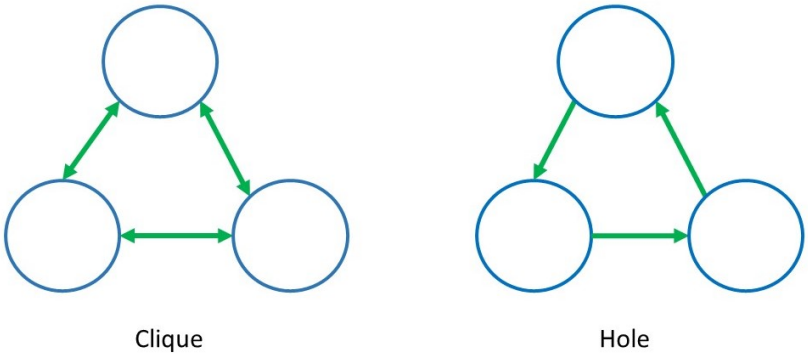

Figure 4.15: Triangoli: clique e hole

in presenza di un triangolo.
Per questo motivo, e dato che esistono solamente tre collegamenti tra le risorse, il triangolo è un altro e distinto sottografo notevole, le cui proprietà vengono illustrate nell'etichetta. Poniamo per esempio la classica situazione delle famiglie reali europee: il re, unito in matrimonio con la regina, e l'erede al trono. Nella specifica di DBPedia di questi casi, esistono due proprietà diverse (dbp:father e dbp:mother) che partono dal principe, mentre una bidirezionale tra re e regina (dbp:spouse).

Il sottografo che si presenta (Figura 4.16) non è nè una clique (non tutte

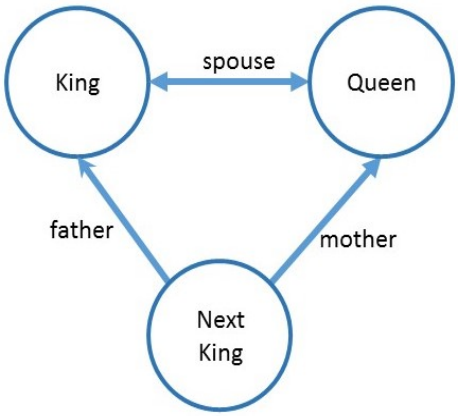

Figure 4.16: Triangolo familiare

le proprietà sono bidirezionali), nè una hole, ma la situazione è abbastanza comune: pensiamo a un film prodotto da due registi che sono fratelli (i Coen, o i Wachowski per esempio), oppure a un articolo prodotto da due autori che fanno parte dello stessa organizzazione.
Le caratteristiche del triangolo che saranno inserite nell'etichetta da una parte ricalcano quelle generale del sottografo (l'omogeneità, i tipi dei componenti), dall'altra lo distinguono dalla clique (il vincolo di bidirezionalità) o dalle hole (l'orientamento delle frecce).

4.4 Etichette

Il ruolo delle etichette, come si è potuto intuire dai riferimenti lungo questo capitolo, è quello di specificare le caratteristiche dei sottografi notevoli che vengono riconosciuti.
In questa sezione, verrà esplicitato come vengono apposte le label alle figure e la loro composizione.

4.4.1 Proprietà delle risorse fittizie

La creazione della risorsa fittizia, dopo il riconoscimento e la formazione dell'IRI, non è sufficiente a definire un sottografo. L'identificatore infatti è un nome come un altro, diverso per tutte le risorse coinvolte e di nessuna utilità nel processo di data mining: serve solo come base per poter assegnargli delle proprietà.
Quella più importante è anche quella più banale, ed è già stata inserita nella CONSTRUCT per le clique: è il tipo. La proprietà rdf:type verrà connessa infatti a una classe appositamente creata, che rappresenterà la configurazione del sottografo notevole.

```
CONSTRUCT{
    ?iri a <triangle>
    }WHERE { [conditions] }
```

Questa sarà una proprietà comune e a tutti i triangoli riconosciuti nell'analisi, e in quanto tale potrà essere utile al processo di data mining. Molte "transazioni" che risulteranno dal modello RDF con le risorse fittizie (vedremo in seguito come) avranno al loro interno un valore "triangolo".
Le etichette relative ai sottografi saranno quindi specificate in RDF, sia per conformità all'analisi, non cambiando l'approccio e il linguaggio di interrogazione, come proprietà delle risorse fittizie.

4.4.2 I componenti

Un altro aspetto particolare da inserire come etichetta, per ogni sottografo, è la lista delle risorse che lo compongono. Questo può essere utile per diversi motivi: innanzitutto la possibilità di ricostruire il sottografo specifico, senza perdita di informazione rispetto al dataset originale, inoltre, permette l'estrazione di un sottoinsieme dei sottografi, ponendo vincoli come la presenza di una risorsa in particolare.
Avere inoltre un riferimento al dataset da cui deriva il modello può essere utile per scoprire nuove informazioni sconosciute al momento del riconoscimento delle figure. Questo procedimento è automatico e fatto da SPARQL, e non è detto che l'utente conosca a priori tutte le caratteristiche dei dati. Una volta selezionati i sottografi, analizzando le proprietà dei loro componenti si potrebbero scoprire pattern significativi.

L'etichetta che indica le risorse che compongono la figura è un'altra proprietà diretta della risorsa fittizia:

```
<rdf:Description rdf:about="http://sws.geonames.org./2658434-3017382-3175395">
    <j.0:comprende rdf:resource="http://sws.geonames.org/3175395/"/>
    <j.0:comprende rdf:resource="http://sws.geonames.org/3017382/"/>
    <j.0:comprende rdf:resource="http://sws.geonames.org/2658434/"/>
    <rdf:type rdf:resource="http://myvocabulary.com/triangolo"/>
</rdf:Description>
```

Questo triangolo è sempre quello formato da Italia, Svizzera e Francia, in quanto stati confinanti. La proprietà comprende (con un prefisso qualsiasi definito dall'utente) è l'etichetta che denota i componenti.
Mentre la proprietà di tipo è fondamentale per l'estrazione di regole di associazione, questa etichetta ha un ruolo più illustrativo e maggiormente utile nel post-processing, tuttavia anche questa caratteristica può essere processata dal tool di data mining. Un pattern interessante può essere un caso particolare della regola grafo implica risultato, in cui la presenza di un componente (anche senza un supporto molto alto) forza le operazioni fatte sulle risorse ad assumere un determinato valore. Riprendendo l'esempio sull'affinità economica tra triangoli di paesi confinanti, è probabile che nel caso due stati europei che confinano tra loro e con la Russia questa somiglianza non ci sia, data la particolarità del terzo, per posizione ed estensione geografica.

4.4.3 Caratteristiche del grafo notevole

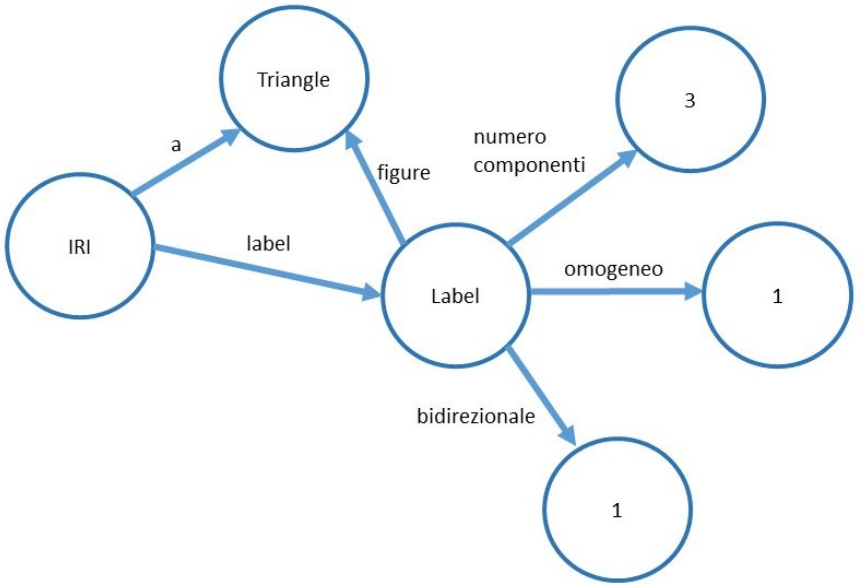

Figure 4.17: Risorsa ed etichetta

Oltre al tipo e ai componenti serve indicare alcune caratteristiche di base che permettano la ricostruzione della figura. Questi valori variano in base al sottografo notevole che viene esaminato: nel caso della hole, ad esempio, è importante indicare il numero di componenti e la proprietà che li collega tutti, ma se in questo caso l'assenza di bidirezionalità è data dalla definizione, nel triangolo è necessario indicarla come caratteristica ulteriore.
Queste informazioni non vengono messe in relazione direttamente con l'entità figura, ma raccolta in un oggetto etichetta, anche questo definito in RDF.

```
CONSTRUCT{
    ?iri a <triangle>;
    my:label <newLabel>.
    <newLabel>my:numeroComponenti 3 .
    <newLabel>my:omogeneo 1 .
    <newLabel>my:bidirezionale 1 .
    <newLabel>my:figura <triangle>.
}
WHERE{ [conditions] }
```

Le proprietà omogeneo e bidirezionale sono semplicemente dei flag binari: 1 se la figura è composta da risorse dello stesso tipo nel primo caso, se tutti i collegamenti sono bidirezionali nel secondo. Nell'esempio abbiamo quindi un triangolo-clique.
Essendo altre risorse RDF, anche questo genere di etichette può essere processato per la scoperta di regole di associazione, e quindi viene ripetuto anche il tipo di figura, per comodità ed evitare successive rielaborazioni. Lo stesso vale per il numero di componenti: se questo può essere ridondante nel caso del triangolo, non lo è per una hole.
Le informazioni di questa etichetta possono essere assegnate direttamente con la risorsa fittizia attraverso la trasformazione già presentata nel caso del modello a stella.

5. Estrazione delle regole: implementazione

In questa sezione, verrà spiegato nel dettagli come dal dataset iniziale in RDF si passa, attraverso trasformazioni e modelli intermedi, a una forma normale pronta (o da rielaborare in maniera semplice) per fare da input agli algoritmi per l'estrazione di regole di associazione. Le diverse fasi, alcune delle quali già anticipate saranno illustrate anche attraverso esempi. Viene dato ampio risalto anche al lato maggiormente pratico del procedimento, riportando codice Java, modelli di query e construct SPARQL, e come vengono richiamati i tool di data mining.

5.1 Riduzione del dataset

In condizioni ideali, le grandi dimensioni delle knowledge base e l'esecuzione in generale di qualsiasi query non dovrebbe rappresentare un problema. Tuttavia, le risorse limitate in locale (in termini di memoria centrale, soprattutto) e possibili defezioni degli endpoint SPARQL, come l'impossibilità di eseguire un gran numero di query in poco tempo, o il linguaggio limitato (spesso alcuni nuovi paragidmi, come il NOT EXISTS o le federated query, non sono supportati), pongono spesso la necessità di passare attraverso un modello intermedio. Per molti dataset, l'aspetto enciclopedico e la grande diversità dei concetti e dei dati rappresentati, è possibile fare una selezione che può ridurre di molto la dimensione e i problemi associati. La creazione di modelli in locale inoltre permette di eseguire, attraverso le librerie Jena, qualsiasi tipo di query alla versione più aggiornata di SPARQL.

5.1.1 Esplorazione ed operazioni di base

La selezione di risorse e proprietà di un dataset RDF non è comunque un processo immediato: raramente si ha una visione di insieme del grafo (anche perchè spesso avrebbe dimensioni tali da essere illeggibile), per cui serve innanzitutto esplorare la base di dati.
Questo può essere fatto con il browser, anche se non sempre con pagine presentate con chiarezza, partendo da una risorsa e navigando attraverso proprietà e link ad altre entità, andando magari a finire da un dataset all'altro.
Nel caso questo non fosse possibile, per esempio GeoNames delle sue risorse

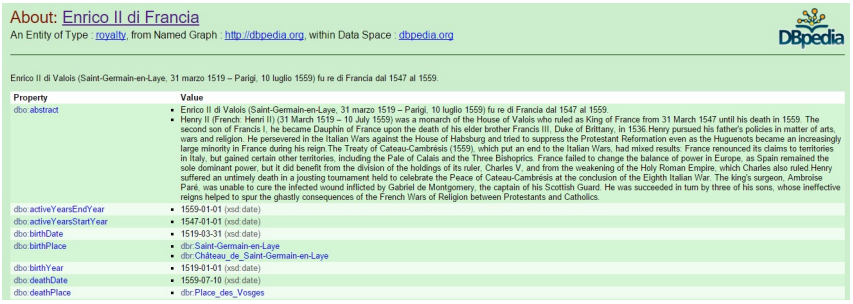

Figure 5.1: Screenshot da DBPedia

presenta solo una selezione sommaria delle proprietà, con le altre che possono essere analizzate facendo il download di un file, il modo più semplice per controllarle tutte è quello di interrogare con SPARQL. Il procedimento è svolto, in maniera automatica, da servizi come LODView (in cui basta inserire l'IRI per iniziare la navigazione), ma può essere anche implementato secondo le proprie esigenze, che magari comprendono un'elaborazione più complessa.
L'estrazione delle entità si fa sfruttando la proprietà rdf:type:

```
SELECT DISTINCT ?concept
WHERE{ [] a ?concept}
```

In cui le parentesi quadre stanno per una qualsiasi risorsa e "distinct" ha la funzione di evitare i duplicati.
Nelle librerie Jena per Java, la query si esegue creando un oggetto Query e QueryExecution:

```
Query q= QueryFactory.create("text", Syntax.syntaxARQ);
QueryExecution qe = QueryExecutionFactory.create(q, "endpoint");
ResultSet rs = qe.execSelect();
```

Il ResultSet rappresenta il risultato della query attraverso una tabella, con gli argomenti sulle colonne e le triple che rispettano la condizione sulle righe. Per analizzarne il contenuto, è sufficiente fare un ciclo:

```
while(rs.hasNext()){
    QuerySolution qs = rs.next();
    RDFNode rn = qs.get("concept");
    String s = rn.toString();
```

Passando ogni riga della tabella con il metodo rs.next() (fino a raggiungerne la fine), si possono estrarre poi le variabili in forma di RDFNode, utilizzandone il nome. Una volta ottenuta l'entità di tipo e memorizzata nella stringa s, questa può essere utilizzata a sua volta, con questa query:

```
    String query =“SELECT ?res WHERE { ?res a <”+s+”>} LIMIT 10”
    q = QueryFactory.create(query, Syntax.syntaxARQ);
    qe = QueryExecutionFactory.create(q, “endpoint”);
    ResultSet risorse = qe.execSelect();
    ResultSetFormatter.out(risorse);
} //fine del ciclo while
```

Vengono estratte le prime dieci triple che siano del tipo trovato, e vengono stampate in forma di tabella attraverso il metodo statico ResultSetFormatter.out. Il ciclo quindi per ogni tipo estrae le prime dieci risorse che trova e queste possono essere visionate attraverso il browser, per vederne le caratteristiche. Un approccio più diretto, ma forse meno utile nel capire il dataset nelle fasi iniziali, è l'estrazione diretta delle proprietà possibili per ogni tipo:

```
SELECT DISTINCT ?prop WHERE {
    ?res a <type>.
    ?res ?prop [] .
}
```

Il meccanismo non cambia nemmeno con un dataset in locale, tuttavia serve avere un oggetto di tipo Model, corrispondente al documento RDF:

```
Model m = FileManager.get().loadModel(”doc.rdf”);
```

Se si ha a che fare con un dump in HDT, serve qualche passaggio in più:

```
HDT hdt = HDTManager.mapIndexedHDT(”file.hdt”, null);
HDTGraph graph = new HDTGraph(hdt);
Model m = ModelFactory.createModelForGraph(graph);
```

Una volta ottenuto l'oggetto Model, l'esecuzione della query viene predisposta così:

```
QueryExecution qe = QueryExecutionFactory.create(q, m);
```

Se invece, ma solo nel caso di file in RDF, si è indicato il documento su cui eseguire già nel testo della query con la clausola FROM, è possibile non mettere il secondo parametro della create, lasciando solo l'oggetto Query.

5.1.2 Modello replica

L'esplorazione del dataset serve all'utente per individuare ciò che ritiene interessante, n questo caso con il fine delle regole di associazione. Una volta individuate le risorse e le proprietà (quest'ultime potranno anche essere facilmente integrate con un ulteriore accesso al dataset), è utile creare un modello intermedio, una sorta di piccola replica della knowledge base che contenga solo

ciò che è stato selezionato.
Questo si fa con una CONSTRUCT semplicissima, in cui le condizioni poste nella parte WHERE (che quindi elimineranno tutto ciò che non è interessante) verranno replicate anche nel definizione del nuovo modello. Per esempio, estraendo informazioni dal dataset della Banca Mondiale:

```
CONSTRUCT{
    ?nation a <http://dbpedia.org/ontology/Country>;
    <http://www.w3.org/2002/07/owl#sameAs>?sameAs ;
    <http://worldbank.270a.info/property/income-level>?level ;
    <http://worldbank.270a.info/property/lending-type>?lending ;
    <http://worldbank.270a.info/property/region>?region .
} WHERE {
    ?nation a <http://dbpedia.org/ontology/Country>;
    <http://www.w3.org/2002/07/owl#sameAs>?sameAs ;
    <http://worldbank.270a.info/property/income-level>?level ;
    <http://worldbank.270a.info/property/lending-type>?lending ;
    <http://worldbank.270a.info/property/region>?region .
}
```

La struttura del nuovo grafo è perfettamente identica a quella del dataset, con tanto di proprietà sameAs che potrebbe servire per join futuri, ma escludendo tutto ciò che non è stato ritenuto interessante (etichette, identificatori interni ecc.). Per questioni di provenance tracking, ovvero di sapere da dove i dati presenti sono stati estratti, è molto consigliato mantenere sempre la proprietà di tipo all'interno del modello.
L'esecuzione è simile a quella di una query e dà come risultato un oggetto di tipo Model, che si può anche scrivere su file:

```
QueryExecution qe = QueryExecutionFactory.create(queryObject, "endpoint");
Model replica = qe.execConstruct();
replica.write(new FileOutputStream("replica.rdf");
```

Direttamente sull'oggetto oppure dal file, su questa versione ridotta del dataset originale, sarà ancora possibile effettuare interrogazioni e altre operazioni. La sintassi del documento sarà quella RDF-XML, e quindi sarà possibile trattarlo anche attraverso le librerie Documnt Object Model (DOM) per l'XML.

5.2 Modellare l'informazione

Se il modello replica può essere visto come un espediente per risolvere problematiche esclusivamente di ordine pratico, mantenendo inalterata la struttura (almeno per le risorse e le proprietà estratte) del dataset, in questo capitolo si vedrà come si deve intervenire per la produzione di un grafo che possa essere facilmente tradotto in transazioni.

5.2.1 Effettuare un join tra modelli ridotti

Qualora l'utente lo ritenga necessario per una raccolta di informazioni sufficienti all'analisi, la prima operazione da fare è quella di unire più knowledge base, sfruttando il paradigma dei Linked Open Data.
Sia le federated query che l'intersezione di due dataset molto grandi possono essere molto dispensiose, in termini di memoria e di tempo. A questo punto del procedimento, tuttavia, dai diversi dataset possiamo ricavare dei modelli replica con dimensioni molto minori. Mantenendo le proprietà di collegamento (owl:sameAs o simili), il join è possibile con lo stesso risultato logico, anche tra modelli replica, risparmiando tempo e risorse.
Questo tipo di query ha un modello ben preciso: data una serie di fonti, basta inserire le condizioni per ognuna (aiutandosi, con il tipo e il provenance tracking, a non confonderli) in modo separato, aggiungendo poi quelle che servono per connetterle (sostituti delle chiavi esterne del modello relazionale). Nella parte di costruzione del modello della construct poi l'utente ha libertà di assegnare le proprietà provenienti dalle diverse fonti come meglio crede. Le fonti, essendo queste documenti RDF precedentemente preparati, si indicano con la clausola FROM.

```
CONSTRUCT{
    [model definition]
}
FROM <sourceRidotta1.rdf>
FROM <sourceRidotta2.rdf>
WHERE{
    [condizioni per source 1]
    [condizioni per source 2]
    ?var1 p ?key1 .
    ?var2 q ?key2 .
    FILTER(?key1 = f(?key2))
}
```

La variabile ?foreignKey potrebbe essere direttamente (specialmente con la proprietà owl:sameAs) l'URI della risorsa nel secondo dataset, tuttavia il modello è dato nel modo più generale: è possibile quindi trovare una corrispondenza biunivoca, rappresentata dalla funzione f, tra le proprietà "in comune" dei due grafi. Queste possono essere semplicemente uguali, oppure possono risultare da aggiungere e togliere caratteri, per passare da un codice alfanumerico a un URI completo e viceversa. Come per il normale accesso a file di Java, i due documenti RDF (se non viene specificato il percorso completo) devono trovarsi all'interno della cartella del progetto.

5.2.2 Il caso semplice: produzione delle stelle

Il modello unito prodotto dalla construct di join, contiene tutta l'informazione utile al processo di data mining. Su questo grafo completo, è quindi possibile

cominciare a ricercare strutture, risorse e figure notevoli e apporre le etichette, orientate alla produzione di regole grafo implica grafo.
Prima però di affrontare questa parte, concentriamoci su un esempio più semplice: quello delle regole valore implica valore, in cui non è riconosciuto nessun grafo notevole, ma solo risorse interessanti con le loro proprietà. Per la rappresentazione di queste informazioni, la forma ideale è quella della configurazione a stella, con l'identificatore della risorsa al centro e le proprietà che si collegano direttamente ad esso. Per il passaggio dal modello join a quello a stella, è necessaria una trasformazione, che contenga le proprietà comuni (che identificano le risorse su cui si intende fare data mining), quelle interessanti che potrebbero dar luogo a regole di associazione e accentri tutte quelle caratteristiche che si trovano ai livelli superiori.
Questo è il modello della trasformazione:

CONSTRUCT{
?center $p_h \; ?o_h \;\; \forall h \in I$;
$lev_k \;\; y_{n,k} \;\; \forall k \in L$;
$ger_x \;\; ?e_{x,i} \;\; \forall i \in [1, N-1], x \in G$;
?t ?opt .
}
WHERE{
?center $q_j \;\; < obj_j > \forall j \in C$.
?center $p_h \; ?o_h \;\; \forall h \in I$.
?center $lev_k \; ?y_{1,k} \;\; \forall k \in L$.
$?y_{i,k} \;\; p_{i,k} \;\; y_{i+1,k} \;\; \forall k \in G, i \in [1, N-1]$.
?center $ger_x \; ?e_{x,1} \;\; \forall x \in G$.
$?e_{x,i} \; sup \; ?e_{x,i+1} \;\; \forall x \in G, iin [1, N-1]$.
$?e_{x,N-1} \; sup \;\; < e_{x,N} >$.
}

Ogni proprietà è assegnata, nella prima parte della construct direttamente al centro, con condizioni diverse a seconda del caso:

- delle serie di livelli (lev_k, insieme L) viene assegnata al centro solo l'ultima proprietà, mentre invece è necessario ripercorrere tutta la serie nella parte where;

- per ogni gerarchia di aggregazione (percorsa con la proprietà sup, insieme G), vengono assegnate tutte le dimensioni della tranne quella fissa posta all'ultimo passaggio;

- le proprietà interessanti (insieme I) vengono messe come condizione e ripetute anche nella definizione del modello;

- le proprietà comuni, insieme C, vengono fissate (con la loro risorsa-oggetto) nella parte where e non trascritte (sarebbero inutili nell'analisi); proprietà interessanti e che tuttavia non vengono ritenute necessarie, pos-

sono essere ricavate nella parte WHERE con la clausola OPTIONAL:
OPTIONAL {?s ?t ?opt }

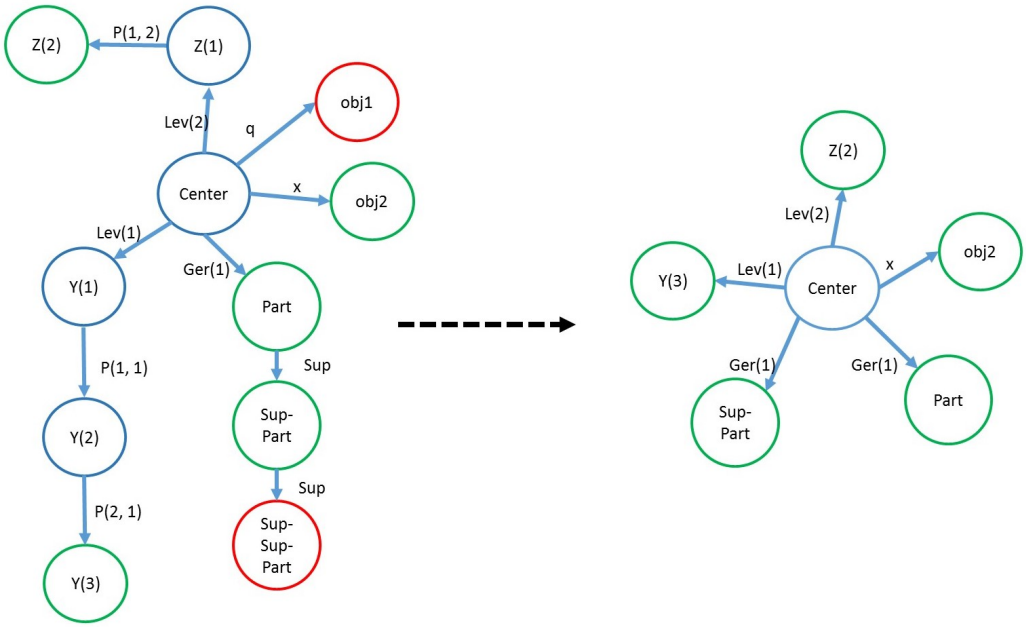

Figure 5.2: Produzione di una stella

La variabile ?o, anche se potrà non avere un valore definito per tutte le triple, può essere utilizzata nella construct ed essere assegnata.
Volendo applicare la produzione di una stella ad un esempio concreto, si può trovare l'esempio del rapporto tra aree geografiche e partiti: si può trovare il rapporto tra settori di diverse dimensioni (comuni, province, regioni) e quindi avere una gerarchia, e ricavare la proprietà del partito (risorsa di secondo livello) del sindaco o governatore (che sono invece al primo livello). Una proprietà comune può essere invece la nazione, o il continente, se per esempio si fa riferimento ai partiti europei.
Al termine di questo procedimento, avremo un documento RDF fatto di sole stelle: ognuna con il proprio centro (l'URI che soddisfa tutte le proprietà comuni, possiede quelle interessanti e gerarchica) e con una serie di informazioni su di essa. Il procedimento di data mining non sarà altro che un'analisi delle "punte" di queste stelle, cercando quali possano implicarne altre, trovando quindi regole valore implica valore.

5.2.3 Stelle e sottografi notevoli

Se la configurazione a stella è quella più valida per passare dall'RDF a una serie di transazioni (vedemo poi in che modo), nel caso precedente tutte le proprietà sono, direttamente o meno, collegate a una risorsa centrale presente nel dataset originale. Il nostro obiettivo però consiste anche nel trovare sottografi

notevoli e trovare ricorrenze frequente di questi, non solamente di proprietà singole.
Le figure che si possono individuare in un grafo e le condizioni siano da inserire in SPARQL (presente nel capitolo 4), sono qui integrate al fine di creare stelle che permettano di generare delle regole grafo su grafo. La creazione della risorsa fittizia, manipolando gli IRI dei componenti della figura, va intesa in questo senso: creare un oggetto che si possa trattare allo stesso modo di tutti gli altri del dataset, generando stelle e regole all'apparenza identiche a quelle presentate in precedenza, aggiungendo poi una fase di analisi.
La risorsa fittizia può essere il centro o una proprietà della stella, a seconda della necessità, con le caratteristiche estratte da quelle dei componenti o ricavate da esse.
Distinguiamo quindi i seguenti casi:

- generazione di regole grafo → risultato: la risorsa fittizia è il centro della stella (e il suo identificatore non ha alcuna utilità), il processo di data mining è fatto sulle diverse proprietà, tra cui quella di tipo che indica la tipologia di grafo riassume la risorsa. Può quindi essere ricavata, se ha supporto e confidenza sufficienti, una regola del tipo grafo implica proprietà, con questa eventualmente ottenuta da calcoli su quelle delle risorse;

- generazione di regole grafo → grafo: sono presenti due risorse fittizie, quella che fa da centro della stella e un'altra, con cui il centro è collegato. Entrambe, essendo due identificatori, non potranno dar luogo a pattern frequenti, per cui anche qui è necessario riportare il tipo di grafo. Per il centro, vale lo stesso discorso del caso precedente, mentre per la risorsa fittizia messa come proprietà è necessario estrarre l'oggetto di rdf:type (quindi di secondo livello) e riportarlo al primo, nella classica trasformazione di avvicinamento;

- generazione di regole valore → risultato: questo caso particolare (il valore non è altro che un grafo composto da una singola risorsa) vuole che la risorsa fittizia sia solamente una proprietà. Anche qui, serve poi avvicinare il tipo in secondo livello.

Un esempio di regola grafo implica grafo, e di costruzione per la ricerca di una grafo implica risultato, è presente nel capitolo 3. In generale quindi, il modello per la creazione di una stella semplice con sottografi notevoli è il seguente:

```
CONSTRUCT{
    ?center a <graphSubj>;
    relation <graphObj>;
    graphProp ?elab.
    <graphSubj>a <graphType1>;
    label <label1>.
    <graphObj>a <graphType2>;
    label <label2>.
```

```
WHERE{
    [riconoscimento di graphSubj]
    [riconoscimento di graphObj]
    ?center prop ?o .
    ?elab = f(?o) .}
```

Vanno quindi riconosciuti singolarmente i singoli grafi notevoli, siano essi al centro o una proprietà, mentre vanno elaborate "le punte" in base alle esigenze della regola: nel caso di un valore implica grafo, possono essere semplicemente ricavate da quella della risorsa centrale, nel caso di un grafo implica risultato, serviranno confronti e operazioni più complesse. Al semplice IRI del grafo, prodotto nel riconoscimento, viene aggiunto il tipo e l'etichetta. Anche se visivamente i due sottografi sono identici, se questi sono composti da risorse e proprietà diverse occorre diversificarli, in modo da poterli distinguere una volta ottenute le regole di associazione.
Riprendendo l'esempio del capitolo 3, il risultato viene calcolato confrontando i luoghi di nascita tra re e predecessore e apponendo un flag in caso positivo:

```
?center <http://dbpedia.org/property/birthPlaceKing>?birth .
?center <http://dbpedia.org/property/birthPlacePredecessor>?b .
FILTER(?birth = ?b)
VALUES ?val {1}}
```

Chiaramente, anche la gestione delle gerarchie e l'avvicinamento di proprietà al primo livello (e magari la successiva rielaborazione), possono essere integrate nella construct per modelli con sottografi notevoli.
Una volta applicate queste trasformazioni, il risultato sarà quindi di un documento RDF composto da stelle di risorse e letterali, corredate dalle label (anche queste descritte nella parte di classificazione). Come si può facilmente capire, non c'è nessuna differenza sintattica tra i due: la differenza sarà esclusivamente data dalle etichette e nel significato dei valori e delle proprietà.

5.3 La forma normale binaria

Il risultato ottenuto è ancora formulato in RDF e, seppure in una versione standard e meno intricata di un dataset intero, non ancora trattabile da un tool di data mining. In questa sezione verrà quindi presentata una forma normale che possa essere utilizzata per produrre l'input di Apriori o FPGrowth, e il procedimento per ottenerla. A differenza delle trasformazioni, in cui era centrale il ruolo di SPARQL per passare da un modello all'altro, qui il linguaggio viene utilizzato solamente per interrogazioni semplici, con i dati estratti rielaborati dal codice Java.
A prescindere dal tool che viene scelto per l'esecuzione degli algoritmi per le regole di associazione, l'input dovrà mantenere la maggior quantità di informazione possibile. Se per i sottografi complessi non è possibile trasformarli di-

rettamente in transazioni (ed è per questo che è stata inserita la risorsa fittizia e l'etichetta), ogni configurazione a stella si può rielaborare in una struttura più semplice senza perdita di informazione: la forma normale binaria.

5.3.1 Concetti preliminari e definizione

Per questa rappresentazione vengono utilizzate le librerie di Weka, in cui ogni istanza è una serie di attributi, ognuno con un proprio valore. Prima di vederne le caratteristiche e come viene svolta la traduzione, presentiamo le classi coinvolte:

- Attribute: ogni oggetto è un attributo, descritto da una stringa;
- Instance: ogni oggetto rappresenta un'istanza, di lunghezza pari a quella degli attributi che contiene. La classe Instance è un'interfaccia con due implementazioni: DenseInstance, in cui ogni attributo ha un valore (lasciando evidente degli spazi vuoti in caso di missing value), e SparseInstance che è una versione più sintetica, in cui è possibile specificare quali attributi hanno valore, mentre tutto il resto è mancante di default;
- Instances: ogni oggetto è una lista di istanze, con tutti i possibili attributi.

Con queste premesse, sarebbe spontaneo associare ogni proprietà di una configurazione a stella con un attributo. Ogni istanza però può avere un solo valore per ogni attributo, per cui questa scelta darebbe luogo a problemi nei moltissimi casi in cui qualche proprietà possiede più valori (in un problema classico, lo scontrino contiene diversi prodotti acquistati).
Weka, per questi casi, offre la possibilità di creare attributi relazionali (1:N), tuttavia questo genere di costrutti non è ancora supportato, nello stesso tool, da FPGrowth o Apriori (lo è invece per gli algoritmi di classificazione, ad esempio), per cui si è preferito creare ogni attributo per ogni coppia proprietà-valore della configurazione a stella.
Questa soluzione chiaramente presenta dei problemi: se gli oggetti sono molto diversi tra loro, avremo una lunghissima lista di coppie che saranno poi utilizzate solamente per poche istanze. Nel nostro caso, in cui si arriva a questo modello dopo trasformazioni e la scelta delle proprietà, questo problema è però meno presente, infatti dovrebbero essere state selezionate solo le caratteristiche frequenti e utili all'analisi.
Il valore di questi attributi è binario: 1 se la risorsa in quella proprietà possiede quel determinato valore, 0 altrimenti. Ogni istanza, cioè ogni centro di stella con il proprio identificatore, avrà quindi una serie di zeri e uno, ognuno corrispondente alla presenza della coppia-attributo. Allargando il campo a tutte le stelle presenti nel modello, avremo quindi una matrice binaria.
Prima di dare la definizione, un esempio molto ridotto può essere d'aiuto. Ogni stella del modello presentato nella Figura 5.3 seguente ha 3 proprietà: starring, per quanto riguarda gli attori; director, il regista del film, e la compagnia di distribuzione. Per ogni coppia proprietà-valore del documento avremo un

attributo, sia che questo sia sempre presente, come starring=Groucho Marx, sia che compaia una sola volta, come director=LeoMcCarey. Ogni stella viene quindi trasformata in istanza, mettendo 1 per ogni coppia presente.

La forma normale binaria è quindi una matrice in cui l'elemento $A_{i,j} = 1$

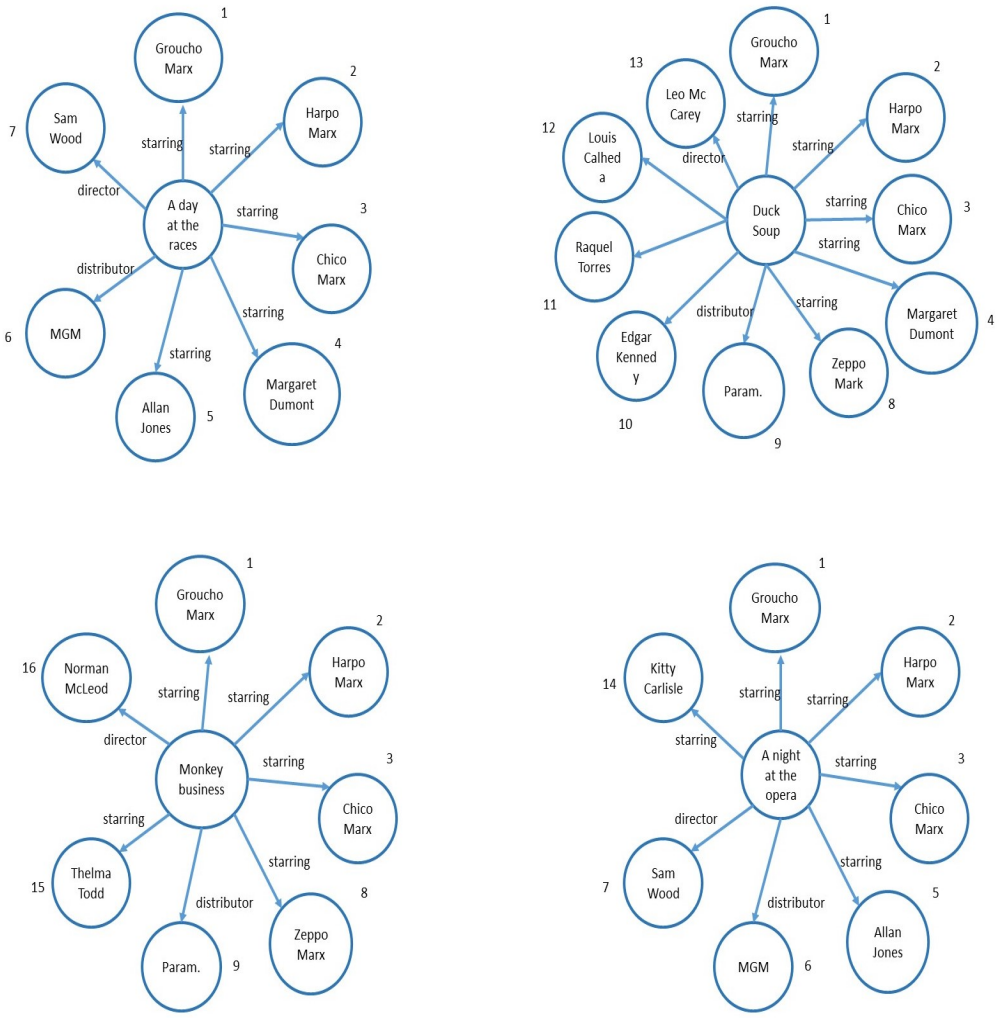

Figure 5.3: Esempio di stelle con quattro film dei Fratelli Marx

se, dato j come coppia tra una proprietà pe un oggetto obj, esiste nel modello RDF la tripla i p obj, ovvero esista per la stella della risorsa i il collegamento con l'oggetto obj attraverso la proprietà p.

Come si può notare dall'esempio (Figure 5.3 e 5.4), anche in un caso molto piccolo le colonne della matrice possono essere molte, data la presenza di risorse utilizzate solamente da pochissime stelle. La dimensione della matrice non rappresenta comunque un problema dal punto di vista del calcolo, essendo infatti una serie di valori binari rappresentabili con pochi bit.

Riprendendo ogni singola coppia proprietà-valore, questa forma normale mantiene tutta l'informazione originale.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
B	1	1	1	0	0	0	0	1	1	1	1	1	1	0	0	0
C	1	1	1	0	0	0	0	1	1	0	0	0	0	1	0	0
D	1	1	1	0	1	1	1	0	0	0	0	0	0	0	1	1

Figure 5.4: Matrice binaria dell'esempio dei Fratelli Marx

5.3.2 Traduzione

Quello che serve per passare dal modello a stella alla forma normale binaria è una vera e propria traduzione, che percorra e rielabori ogni nodo e lato del grafo, creando la matrice e assegnando i valori. In questa sezione quindi vedremo come funziona l'algoritmo, abbastanza semplice, di traduzione, estraendo i dati con SPARQL e rielaborandoli con Java. Il risultato, conforme alla definizione teorica data in precedenza, è direttamente utilizzabile in Weka, trasformabile in file ARFF (formato supportato anche da altri tool di data mining) e traducibile nuovamente nelle versioni richieste da software diversi.
Prima di tutto, la necessità è quella di creare le colonne della matrice, ovvero trovare tutte le possibili coppie proprietà-valore e inserirli in una lista. Per semplificare, l'esecuzione della query è svolta da una classe QueryEngine:

```
ArrayList<Attribute>attributi = new ArrayList<>();
QueryEngine qe = new QueryEngine();
qe.setModel(model);
String properties = "SELECT DISTINCT ?prop WHERE{[] ?prop [] .}";
ResultSet rs = qe.queryModel(properties);
```

Percorriamo il risultato e, per ogni proprietà, troviamo tutti i suoi possibili valori. L'attributo avrà come nome la concatenazione delle due stringhe, separate dall'uguale.

```
while(rs.hasNext()){
    QuerySolution qs = rs.next();
    RDFNode rn = qs.get("prop");
    String prop = rn.toString();
    String values = "SELECT DISTINCT ?value
    WHERE"+ "?subj <"+prop+">?value .";
    ResultSet valori = qe.queryModel(values);
    while(valori.hasNext()){
        QuerySolution qs1 = valori.next();
        RDFNode rn1 = qs1.get("value");
        String value = rn1.toString();
        Attribute = new Attribute(prop+"="+value);
        attributi.add(a);}}
```

L'ArrayList attributi saranno quindi le colonne della matrice. Il prossimo passo è quello di preparare le righe, selezionando il centro di ogni stella. Per ognuno di questi, trovo tutte le possibili coppie, ricreo la stringa proprietà=valore e cerco l'attributo corrispondente nella lista (con il metodo findAttribute(ArrayList<Attributi >, String nome)). Assegno quindi il valore 1 all'attributo trovato.

```
Instances valori = new Instances("valori",attributi,0);
String instances = "SELECT DISTINCT ?subj WHERE{?subj ?prop []}";
ResultSet istanze = qe.queryModel(instances);
while(istanze.hasNext()){
    rs = qe.queryModel(properties);
    QuerySolution qs = istanze.next();
    RDFNode ist = qs.get("subj");
    String istanza = ist.toString();
    Instance i = new DenseInstance(attributi.size());
    while(rs.hasNext()){
        QuerySolution qs1 = rs.next();
        RDFNode rn1 = qs1.get("prop");
        String prop = rn1.toString();
        String rel = "SELECT ?value
        WHERE{<"+istanza+"><"+prop+">?value .}";
        ResultSet coppie = qe.queryModel(rel);
        while(coppie.hasNext()){
            QuerySolution qs2 = coppie.next();
            RDFNode rn2 = qs2.get("value");
            String val = rn2.toString();
            String attributo = prop+"="+val;
            Attribute a = findAttribute(attributi,attributo);
            i.setValue(a, 1);
        }
    }
    valori.add(i);
}
```

La matrice ottenuta ha alcuni 1 nelle posizioni trovate ma nessuna informazione riguardo a tutte le altre combinazioni, che sono quindi assegnate di default come missing. Questi valori vengono messi a 0:

```
for(Instance i : valori){
    for(Attribute a : attributi){
        if(i.isMissing(a))
            i.setValue(a, 0);
    }
}
```

Con questa costruzione, la matrice è riempita da numeri interi. Per trasformarla in una binaria anche dal punto di vista di Weka serve applicare il filtro NumericToBinary. I filtri sono particolari classi che eseguono una trasformazione sui dati, come per esempio la selezione di attributi o l'eliminazione dei missing values. Come si può intuire dal nome, questo filtro riceve in input un file Instances con valori numerici e lo trasforma in uno con valori binari.

```
NumericToBinary ntbb = new NumericToBinary();
ntbb.setInputFormat(valori);
valori = NumericToBinary.useFilter(valori, ntbb);
```

L'oggetto valori è la nostra forma normale binaria e può essere serializzato come file ARFF, sia tramite le normali classi di input/output di Java, oppure con la classe ArffSaver delle librerie Weka:

```
ArffSaver saver = new ArffSaver();
saver.setInstances(valori);
saver.setFile(new File("file.arff"));
```

Ogni attributo del file ARFF avrà questa forma:

```
@attribute starring=Groucho_Marx_binarized {0,1}
```

Alla lista degli attributi seguirà la matrice binaria, con le colonne nello stesso ordine della lista:

```
@data
1,1,1,1,1,1,1,0,0,0,0,0,0,0,0
1,1,1,0,0,0,0,1,1,1,1,1,1,0,0
1,1,1,0,0,0,0,1,1,0,0,0,0,1,0
1,1,1,0,1,1,1,0,0,0,1,0,0,0,1
```

5.4 Trovare le regole di associazione

La forma normale binaria ottenuta dopo i passi precedenti ha di fatto tolto le complicazioni dovute al linguaggio RDF, creando un input molto più simile a quello del classico modello relazionale e quindi (eventualmente con una breve fase di preparazione) utilizzabile anche dai tool di data mining. In questa sezione quindi vedremo come, nei vari software, viene svolta l'estrazione delle regole di associazione.

5.4.1 Weka

Dato che per la costruzione della forma normale si sono utilizzate le classi fornire dalle API di Weka, quest'ultimo è il primo tool preso in considerazione. L'oggetto di tipo Instances ottenuto dal procedimento può essere utilizzato

come input per Apriori e FPGrowth.
Per utilizzarli, va creato un oggetto della classe dell'algoritmo:

```
Apriori apriori = new Apriori();
FPGrowth fpGrowth = new FPGrowth();
```

Le metriche di supporto e confidenza (nella versione relativa, compresa tra 0 e 1) possono essere fissate rispettivamente attraverso i metodi comuni setLowerBoundMinSupport e setMinMetric.
Per l'esecuzione vera e propria dell'algoritmo invece sono necessarie due fasi: l'inserimento dell'input

```
apriori.buildAssociations(valori);
```

e la ricerca delle regole:

```
AssociationRules ars = apriori.getAssociationRules();
```

Il risultato è un oggetto iterabile, per cui sarà possibile eseguire un ciclo del tipo per vedere le singole regole:

```
for(AssociationRule ar : ars){
    System.out.println(ar);
}
```

Ognuna di queste è un oggetto di tipo AssociationRule, con i metodi per ricavare gli insiemi di premessa e conseguenza (getPremise() e getConsequence(), entrambi che ritornano una collezione), e il supporto di essi in versione assoluta, cioè il conteggio delle transazioni che contengono l'insieme.

5.4.2 Apache Spark

Spark lavora su file di testo tab-delimited, in cui ogni riga rappresenta una transazione. Per poter quindi richiamare gli algoritmi di questo tool, è necessario manipolare la forma normale binaria in questo nuova forma. Il procedimento è molto semplice e non comporta perdite o modifiche dei dati: si percorre la matrice e, dove si trova un 1, si riporta la coppia proprietà-valore corrispondente.
Prepariamo quindi una stringa che rappresenterà tutte le transazioni, ricaviamo la stringa che indica la coppia e se il valore è 1 la aggiungo alla transazione corrente, con una virgola come separatore, che verrà poi aggiunta alle altre andando a capo.

```
String transazioni = ""; for(Instance i : valori){
String transazione = "";
```

```
for(int j = 0; j <i.numAttributes(); j++){
    Attribute a = i.attribute(j);
    String s = pulisci(a.toString());
    s = s + "="+i.value(a);
    if(i.value(a)==1){
        if(transazione.equals("")){
            transazione = 1; }
        else{
            transazione = transazione + ","+ s; }
        }
    transazioni = transazioni + "\n"+transazione;
}
```

Una transazione è quindi una riga fatta in questo modo:

starring=Groucho_Marx=1.0,starring=Harpo_Marx=1.0,starring=Chico_Marx=1.0,Metro-Goldwyn-Mayer=1.0,director=Sam_Wood=1.0

Una volta scritta la stringa transazioni su un file di testo (per comodità chiamiamolo transazioni.txt), passiamo all'utilizzo di Spark vero e proprio, prima richiamando il software in locale:

```
SparkConf conf = new SparkConf().setAppName("FP-growth Example");
conf.setMaster("local");
JavaSparkContext sc = new JavaSparkContext(conf);
```

Poi leggendo l'input:

```
JavaRDD<String>data = sc.textFile("transazioni.txt");
JavaRDD<List<String>>transactions = data.map(
new Function <String, List<String>>() {
    public List<String>call(String line) {
        String[] parts = line.split(",");
        return Arrays.asList(parts);}
});
```

Questa ultima funzione è una map, per cui esegue su tutte le righe l'operazione di dividere le stringhe in base al separatore (line.split, che infatti prende la virgola come parametro) e ne crea una lista. Una JavaRDD è una collezione distribuita di oggetti, anche se qui viene sfruttata solo con un calcolatore, presente nelle librerie di Spark.
A questo punto con software e input pronto, chiamiamo l'algoritmo:

```
FPGrowth fpg = new FPGrowth();
fpg.setMinSupport(0.8);
FPGrowthModel<String>model = fpg.run(transactions);
```

Il risultato è una classe intermedia, FPGrowthModel, da cui posso generare sia i frequent itemset (che magari possono essere interessanti), in un'altra collezione RDD:

```
model.freqItemsets().toJavaRDD();
```

Oppure passare direttamente alle regole di associazione, passando la confidenza come parametro:

```
JavaRDD<Rule<String>>rs = model.generateAssociationRules(0.7).toJavaRDD();
```

Applicando alle liste JavaRDD il metodo collect(), possono essere percorse da un iteratore, come in questo for, dove ricavo antecedente, conseguente e confidenza da ogni regola:

```
for(Rule<String>rule : rs.collect()){
    System.out.println(rule.javaAntecedent() + " =>" + rule.javaConsequent()
    + ", " + rule.confidence());}
```

5.4.3 SPMF

Questo tool, certamente meno conosciuto, specializzato nella ricerca di regole di associazione supporta anche i file ARFF per cui, dopo aver stampato su file l'oggetto di tipo Instances, è possibile, con poche semplici operazioni, renderlo l'input per un algoritmo implementato da SPMF:

```
TransactionDatabaseConverter tdc = new TransactionDatabaseConverter();
Map<Integer, String>mapping = converter.convertARFFandReturnMap
("dati.arff", "dati.txt", 0);
```

L'operazione descritta da queste righe di complice è semplicemente di traduzione uno a uno tra attributi di Weka e item di SPMF. Il file dati.txt sarà quindi una lista di item, del tipo:

```
@ITEM=3=starring=Groucho_Marx=1
```

E di sequenze di numeri che rappresentano le transazioni, riportando i numeri degli item che possiede. La forma normale binaria come è concepita, con 0 dove la coppia proprietà-valore non è presente per la risorsa, viene tradotta tenendo conto anche di tutti gli zeri, per cui potrà esserci anche un item numero 4, con starring=Groucho_Marx=0. Per evitare questo inconveniente, è sufficiente mettere per ogni zero un missing value.

A questo punto, possiamo creare un oggetto Database a partire dal file di input:

```
Database db = new Database();
db.loadFile("dati.txt");
```

Eseguiamo, per esempio, l'algoritmo TopKRules:

```
AlgoTopKRules topK = new AlgoTopKRules();
topK.runAlgorithm(10, 0.7, db);
topK.writeResultTofile("topK.txt");
```

L'algoritmo prende tre parametri: il numero di regole di associazione da estrarre, il supporto minimo e il database di partenza. Il risultato è legato ancora alla mappatura tra numeri e attributi, per cui le regole saranno di questo tipo:

```
3 ==>5 #SUP: 4 #CONF: 1.0
```

Per rendere il risultato più leggibile, è necessaria un'altra conversione, per legare al numero l'attributo effettivo, di seguito il codice e la regola precedente ritradotta:

```
ResultConverter rc = new ResultConverter();
rc.convert(mapping, "topK.txt", "final_output.txt");
starring=Groucho_Marx=1 ==>
starring=Chico_Marx=1
#SUP: 4 #CONF: 1.0
```

5.5 Lettura e analisi delle regole

Ogni tool di data mining produce oggetti, file o semplicemente delle stringhe che rappresentano le regole di associazione. Con metodi definiti già dalle librerie, o estrapolando segmenti da un insieme di caratteri, sta poi all'utente capire il significato di ogni regola, separandone i diversi elementi che formano l'insieme delle premesse, le conseguenze e comprendere il risultato alla luce degli obiettivi fissati inizialmente.
Finchè si trattano semplici item delle transazioni, come nell'esempio scolastico degli scontrini, la lettura e l'analisi è abbastanza semplice: premesse e conseguenze sono semplicemente insiemi di oggetti presenti in delle liste; nel nostro caso invece, la presenza di item composti come le figure richiede un ulteriore passaggio, per riallacciare la regola di associazione con il vero significato, riassunto nelle etichette.

5.5.1 RuleML

Le diverse forme di output dei software di data mining, una volta visibili su un monitor, sembrano fondamentalmente tutte uguali e leggibili dall'utente. Tuttavia, le piccole differenze lessicali (la presenza di parentesi, come viene resa

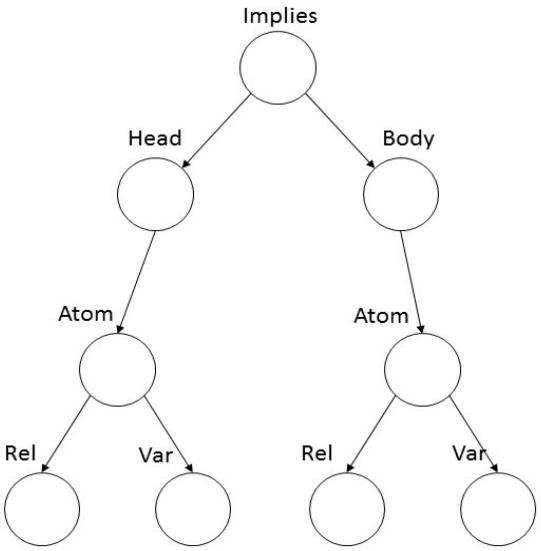

Figure 5.5: Schema di RuleML

graficamente l'implicazione) e in particolar modo la differenza fondamentale tra oggetti, da cui si possono estratte ulteriori proprietà, e semplice testo rendono necessaria la standardizzazione in un unico modello.
La scelta su quale utilizzare è ricaduta su RuleML, che rappresenta le regole di associazione in XML ed è compatibile con RDF. In generale, i documenti in questo linguaggio possono essere rappresentati con un albero, e RuleML non fa eccezione, come si può vedere dalla Figura 5.5.

Dalla radice ("implies") si diramano gli insiemi head delle premesse e body delle conseguenze, ognuno composto da un numero variabile di atom, gli item delle transazioni, a loro volta suddivisi in "rel" e "var", corrispondenti a una proprietà e al suo valore di un modello RDF.
Il procedimento per passare a un modello di questo tipo è molto semplice: per ogni regola, si estraggono gli insiemi di premesse e conseguenze e si creano tanti elementi atom quanti sono gli elementi contenuti, con ognuno di questi formato da una coppia proprietà-valore derivante dalla configurazione a stella. A livello pratico, la coppia è semplicemente una stringa e basta trovare il separatore "=" per isolare le due parti e inserirle come testo dei nodi rel e var. Essendo XML, per fare ciò è molto utile ricorrere alle apposite librerie Document Object Model (DOM) del W3C.

5.5.2 Ritraduzione in RDF

La rappresentazione in RuleML non è tuttavia sufficiente per integrare le informazioni contenute nelle etichette. Dato che queste sono rappresentate in RDF, il passo seguente è tradurre il modello XML standard nella forma a tripla, in modo tale da poter trattare il documento finale contenente le regole di associazione allo stesso livello delle etichette, e quindi poter eseguire query

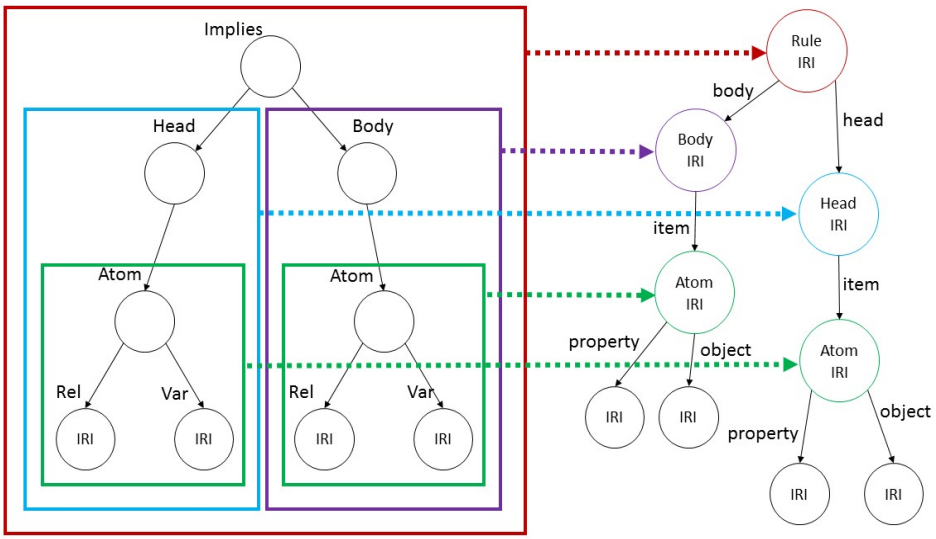

Figure 5.6: Traduzione di RuleML in RDF

e trasformazioni in SPARQL tra i due documenti.
Non avendo, in XML, il concetto di risorsa, ma solo quello di elemento con il proprio contenuto (cioè il sottoalbero che forma, da un punto di vista grafico), è necessario per ogni parte della struttura di RuleML (la regola stessa data dal tag implies, le premesse head, la conseguenza body e ogni atom) creare un IRI; le risorse prodotte ad hoc vanno poi collegate con delle proprietà. Le possibilità di rappresentazione una regola di associazione in RDF possono essere diverse,tuttavia qui si è scelto di mantenere la stessa conformazione di RuleML, salvo sostituire "rel" e "var" con i loro concetti corrispondenti.
La traduzione concettuale va completata anche con la scrittura vera e propria del modello, che può essere fatta nelle diverse sintassi dell'RDF. Per questioni di leggibilità e compattezza, si è scelta quella RDF-XML, per cui la struttura si costruisce anche in questo con le librerie DOM, badando all'inserimento di ogni risorsa una sola volta con il tag rdf:Description, il posizionamento del relativo IRI nell'attributo rdf:about e di inserire solamente le proprietà di primo livello (con tag e attributo rdf:resource).
Questa, per esempio, è la regola per cui la presenza di Groucho Marx in un film implica che vi siano anche Harpo e Chico:

```
<rdf:Description rdf:about="Groucho_MarxIMPLIESHarpo_MarxChico_Marx">
    <my:head rdf:resource="starringGroucho_Marx"/>
    <my:body rdf:resource="starringHarpo_MarxstarringChico_Marx"/>
</rdf:Description>
```

La traduzione però deve necessariamente comprendere anche le risorse che rappresentano i due componenti. Qui sotto viene riportato l'insieme delle conseguenze:

```
<rdf:Description rdf:about="starringHarpo_MarxstarringChico_Marx">
    <my:item rdf:resource="starringHarpo_Marx"/>
    <my:item rdf:resource="starringChico_Marx"/>
</rdf:Description>
```

I due item, come per altro si può intuire dall'IRI, sono formati dalla proprietà starring e dall'identificatore dell'attore:

```
<rdf:Description rdf:about="starringChico_Marx">
    <my:object rdf:resource="Chico_Marx"/>
    <my:property rdf:resource="starring"/>
</rdf:Description>
```

La risorsa finale, collegata dalla proprietà object, sarà l'IRI di un valore semplice (come in questo caso) oppure di una figura, e potrà essere quindi integrabile con le etichette e le loro proprietà

```
SELECT ?label ?prop ?val WHERE{
    ?s my:object ?res .
    ?res my:label ?label .
    ?label ?prop ?val .
}
```

6. Conclusioni e sviluppi futuri

Questa sezione mostra come, alla luce della metodologia di soluzione del problema, gli obiettivi posti nell'introduzione sono stati raggiunti, rendendo anche conto delle difficoltà incontrate e illustrando alcuni sviluppi futuri.

6.1 Conclusioni

L'adattamento della disposizione a tripla tipica dell'RDF in una forma più consona ai concetti del data mining ha richiesto una lunga fase di preparazione composta da diversi punti:

- la selezione delle proprietà e risorse che si intende utilizzare da ciascun dataset preso in esame;
- dove fosse necessario, l'intersezione logica tra le diverse fonti seguendo i collegamenti del progetto Linked Open Data;
- il riconoscimento dei concetti chiave, rappresentati da una figura notevole, e la loro trasformazione in una risorsa singola che li rappresenti, con le relative caratteristiche;
- la creazione di un modello standard, che possa essere tradotto in transazioni.

La realizzazione di questa procedura ha richiesto un lavoro di ricerca, definizione e classificazione. Se infatti nel data mining per il modello relazionale, la regola di associazione è tra insiemi di item monolitici, la struttura dell'informazione dell'RDF può richiedere che interi sottografi possano implicarne altri. Oltre quindi alla regola valore implica valore, che richiama subito alla mente scontrini e prodotti degli esempi più classici, sono state presentate le regole grafo implica grafo e quelle grafo implica risultato, con quest'ultimo che si definisce come una caratteristica risultante, dalla somiglianza o da operazioni matematiche sulle proprietà che compongono il grafo. La trasformazione di un insieme di risorse connesse tra loro in una sola risorsa, ha richiesto l'aggiunta di etichette (ancora sottoforma di triple) per non perdere informazione rispetto al dataset originale. Queste non sono utili direttamente agli algoritmi per le regole di associazione, ma vengono utilizzate per comprendere meglio l'output di questi ultimi, ricollegando l'oggetto al grafo che riassume.
Il riconoscimento delle figure notevoli, la preparazione di modelli RDF intermedi e in generale tutte le trasformazioni, richiedono interrogazioni e construct

SPARQL. Se, nel dettaglio, ogni query si basa su risorse e proprietà anche completamente diverse, sono state riconosciuti alcuni pattern fondamentali di triple, da introdurre come condizione nelle query, che possono guidare l'utente nel trattare con proprietà non immediatamente collegate con la risorsa selezionata, con i parametri dimensionali come il tempo o lo spazio, con le figure ricorrenti e nel passare da un modello all'altro.
Le difficoltà tecniche, date dalle caratteristiche degli endpoint che a volte possono non supportare alcuni costrutti in SPARQL o aver tempi di risposta troppo lunghi, sono satate anch'esse prese in considerazione nella costruzione della procedura, che infatti riporta come prima fase quella di passare da un dataset pubblico, disponibile via web e possibilmente anche molto grande, a uno locale che contenga solamente la porzione utile per l'analisi, senza nessuna modifica alla struttura. Le librerie Apache Jena, utilizzate su documenti RDF presenti sul disco fisso, sono abbastanza efficienti e permettono anche calcoli più complessi, che spesso causano il rifiuto della query da parte dell'endpoint (anche di servizi molto importanti, come Wikidata) e le generazione di eccezioni HTTP. Anche l'intersezione tra diverse fonti, qui per comodità e affinità spesso chiamata join, è possibile eseguirla attraverso il web, ma risulta molto più efficiente se eseguita in locale. Il risultato di questa procedura è la configurazione a stella, ed è composta da una risorsa centrale con solamente proprietà direttamente collegate al centro. Questa forma, estremamente semplice, è stata definita (e poi effettivamente ricavata in tutti gli esempi) come entità ponte tra l'RDF generale e le transazioni, in modo da facilitarne le traduzione nell'input per l'estrazione di regole di associazione, come Apriori o FPGrowth.
Essendo questi presenti ed eseguibili su diversi software di data mining, è stata definita anche la forma normale binaria, come matrice in cui ogni uno rappresenta se la stella possiede determinate caratteristiche, senza perdita di informazione rispetto al modello a stella e con la possibilità di effettuare una traduzione puramente lessicale dalla matrice al vero e proprio input di Weka, Apache Spark o SPMF.
Questi tre tool di data mining sono stati selezionati, tra diversi provati e analizzati per le loro caratteristiche e la semplicità di utilizzo: i file Attribute-Relation di Weka sono infatti molto utilizzati anche da altri sistemi, ed è quindi utile avere una traduzione da RDF a questo formato, Spark è specializzato nel calcolo distribuito e quindi anche per grandi moli di dati ed SPMF è specializzato nelle regole di associazione, e per tanto particolarmente adatto in questo contesto. Le difficoltà nella produzione dell'input per RapidMiner ed Elki, e la solo parziale integrazione di R e del suo linguaggio scripting in Java (di fatto, per ora è possibile solo inviare comandi) hanno spinto a scartare questi ultimi. Come è stato necessario trovare una forma normale per l'input dei tool, allo stesso modo è stata realizzata anche la traduzione dell'output di questi ultimi in uno standard già definito: RuleML, in XML. Per la lettura e l'interpretazione, anche alla luce delle etichette precedentemente assegnate alle risorse-grafi, è stato definito anche un algoritmo di traduzione in RDF di RuleML, rispettando la medesima struttura.
In generale quindi sono state analizzate le principali problematiche, teoriche

e pratiche, relative all'estrazione di regole di associazione da strutture dati composte da grafi, modellando una soluzione che utilizzi, quanto più possibile, conoscenze già acquisite come SPARQL, Apriori e FPGrowth. La possibilità di esprimere concetti complessi , attraverso la struttura generale ed adattabile soggetto-predicato-oggetto, la connessione di dataset appartenenti a settori di conoscenza anche molto diversi e la presenza di numerose fonti anche molto voluminose, permette di generare, dai modelli RDF, regole di associazione potenzialmente molto interessanti.

6.2 Sviluppi futuri

Un problema abbastanza comune nel data mining, che infatti fa ampio uso delle tecniche di visualizzazione e della statistica è avere un'ampia conoscenza dei dati, prima di procedere con algoritmi e metodologie, in modo tale da poter effettuare un'analisi (spesso lunga e costosa) maggiormente centrata ed efficiente. Anche la procedura mostrata in questa tesi richiede, prima di tutto, un lavoro di esplorazione e decisione di argomenti, che possa riconoscere delle strutture notevoli che potrebbero implicare qualche risultato, da testare poi attraverso la produzione delle regole di associazione. Una possibile continuazione del lavoro potrebbe essere quindi l'implementazione di tecniche per scoprire questi concetti ripetuti in un dataset potenzialmente molto esteso, utilizzando anche le tecniche di graph mining e l'estrazione di sottografi frequenti, in modo da ridurre la complessità dell'esplorazione.
La metodologia si basa inoltre su schemi di query, da adattare in base al caso scelto, sostituendo alle variabili astratte le risorse e le proprietà vere e proprie; abbinando anche a questa fase del lavoro le tecniche di graph mining, si potrebbe ricavare e testare la corrispondenza dei grafi frequenti attraverso le regole di associazione.
Essendo SPARQL un linguaggio in espansione, con diverse migliorie proposte e valutate dal W3C, alcune trasformazioni mirate, senza la specifica continua della forma del grafo desiderato, potrebbero essere integrati in costrutti per facilitare il passaggio da una versione all'altra, riducendo la complessità delle query e le possibilità di errore, in particolare alcune operazioni più semplici come la creazione del modello replica e le trasformazioni di avvicinamento.
Lungo tutto il lavoro di questa tesi, gli algoritmi per l'estrazione delle regole di associazioni sono stati considerati a scatola chiusa, solamente utilizzabili. Questo ha portato alla definizione della forma normale binaria, input generico per questi, che di fatto è però solamente una trasposizione in una forma leggibile della configurazione a stella. Un ulteriore sviluppo futuro potrebbe essere quindi l'adattamento di Apriori, facendolo operare direttamente in questa versione semplificata dell'RDF, senza la produzione di una matrice che, a causa della variabilità e molteplicità delle risorse, risulta sparsa e piena di zeri, o un sistema di traduzione nell'FPTree.
Avendo inoltre realizzato un sistema che dalla regola di associazione testuale, trasforma e ripropone le stesse informazioni nuovamente in RDF, complete di etichette ed eventualmente altre proprietà, questi risultati hanno la stessa

forma teorica del dataset iniziale. Potenzialmente, le regole di associazione potrebbero essere riprocessate con questa stessa metodologia, in una sorta di meta-inferenza.

Dati, software e linguaggi utilizzati

A. Fonti dei dati

DBPedia: http://wiki.dbpedia.org

Wikidata: http://www.wikidata.org/wiki/Wikidata:Main_Page

World Bank: http://worldbank.270a.info/about.html

LinkedMDB: http://www.linkedmdb.org

GeoNames: http://www.geonames.org

Muninn: http://blog.muninn-project.org

Yago: http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/

B. Software

Apache Jena: http://jena.apache.org

Weka: http://www.cs.waikato.ac.nz/ml/weka/

Apache Spark: http://spark.apache.org

SPMF: http://www.philippe-fournier-viger.com/spmf/

R: http://www.rdatamining.com

C. Linguaggi

RDF: http://www.w3.org/TR/rdf-syntax-grammar/

SPARQL 1.1: http://www.w3.org/TR/sparql11-overview/

Turtle: http://www.w3.org/TR/turtle/

HDT: http://www.w3.org/Submission/2011/03/

RuleML: http://wiki.ruleml.org/index.php/RuleML_Home

Bibliografia

[1] B. Florian and K. Martin, *Linked Open Data: The Essentials - A Quick Start Guide for Decision Makers*. edition mono/monochrom, Vienna, Austria, 2012.

[2] M. Arias, J. D. Fernández, M. A. Martínez-Prieto, and C. Gutiérrez, "Hdt-it: Storing, sharing and visualizing huge rdf datasets," in *10th International Semantic Web Conference (ISWC 2011)*, 2011.

[3] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," *SIGMOD Rec.*, vol. 22, pp. 207–216, June 1993.

[4] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, (San Francisco, CA, USA), pp. 487–499, Morgan Kaufmann Publishers Inc., 1994.

[5] A. S. H. Yazdi and M. Kahani, "A novel model for mining association rules from semantic web data," in *Intelligent Systems (ICIS), 2014 Iranian Conference on*, pp. 1–4, Feb 2014.

[6] H. Boley, S. Tabet, and G. Wagner, "Design rationale of ruleml: A markup language for semantic web rules," pp. 381–401, 2001.

[7] H. Boley, J. Mei, M. Sintek, and G. Wagner, "Rdf/ruleml interoperabiity," 2005.

[8] D. Braga, A. Campi, S. Ceri, M. Klemettinen, and P. Lanzi, "Discovering interesting information in xml data with association rules," in *Proceedings of the 2003 ACM Symposium on Applied Computing*, SAC '03, (New York, NY, USA), pp. 450–454, ACM, 2003.

[9] J. F. Roddick and P. Fule, "Semgram - integrating semantic graphs into association rule mining," in *Sixth Australasian Data Mining Conference (AusDM 2007)* (P. Christen, P. J. Kennedy, J. Li, I. Kolyshkina, and G. J. Williams, eds.), vol. 70 of *CRPIT*, (Gold Coast, Australia), pp. 129–137, ACS, 2007.

[10] Y. Chi, R. Muntz, S. Nijssen, and J. Kok, "Frequent subtree mining - an overview," 2005.

[11] P. Fournier-Viger, W. Cheng-Wei, and V. S. Tseng, "Mining topk association rules," 2012.

[12] P. Haase, T. Mathäß, and M. Ziller, "An evaluation of approaches to federated query processing over linked data," in *Proceedings of the 6th International Conference on Semantic Systems*, I-SEMANTICS '10, (New York, NY, USA), pp. 5:1–5:9, ACM, 2010.

[13] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," *SIGMOD Rec.*, vol. 29, pp. 1–12, May 2000.

[14] T. Jiang and A.-H. Tan, "Mining rdf metadata for generalized association rules: Knowledge discovery in the semantic web era," in *Proceedings of the 15th International Conference on World Wide Web*, WWW '06, (New York, NY, USA), pp. 951–952, ACM, 2006.

[15] C. Jiang, F. Coenen, and M. Zito, "A survey of frequent subgraph mining algorithms," *The Knowledge Engineering Review*, vol. 28, pp. 75–105, 3 2013.

[16] M. Mazuran, E. Quintarelli, and L. Tanca, "Mining tree-based association rules from xml documents," in *SEBD*, 2009.

[17] V. Nebot and R. Berlanga, *Trends in Applied Intelligent Systems: 23rd International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2010, Cordoba, Spain, June 1-4, 2010, Proceedings, Part II*, ch. Mining Association Rules from Semantic Web Data, pp. 504–513. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[18] Q. Quboa and M. Saraee, "A state-of-the-art survey on semantic web mining," *Intelligent Information Management*, vol. 5, no. 1, 2013.

[19] Radha.M, Theepigaa.TH, Rajeswari.S, and Suganya.P, "Tree based association rules for mining in xml query –answering," *International Journal of Innovative Research in Science, Engineering and Technology*, vol. 3, 2014.

[20] J. W. W. Wan and G. Dobbie, "Mining association rules from xml data using xquery," in *Proceedings of the Second Workshop on Australasian Information Security, Data Mining and Web Intelligence, and Software Internationalisation - Volume 32*, ACSW Frontiers '04, (Darlinghurst, Australia, Australia), pp. 169–174, Australian Computer Society, Inc., 2004.