Alma Mater Studiorum – Università di Bologna

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI

Corso di Laurea Triennale in Informatica

Un Servizio Web per l'analisi Semantica degli Open Data

Tesi di Laurea in Tecnologie Web

Relatore:

Chiar.mo Prof. Fabio Vitali

Correlatore:

Dott. Francesco Poggi

Presentata da:

Gabriele Cigna

Anno Accademico 2016/2017

Indice

1. Introduzione

Questo lavoro di Tesi ha come obiettivo quello di automatizzare il più possibile la comprensione automatica degli *Open Data*. Ciò è stato realizzato mediante la progettazione e lo sviluppo del "*Semantic Detector*", una soluzione che si interpone tra il dato grezzo, quindi il *dataset*, e qualsiasi software ad alto livello che sfrutta questi dati per poterli effettivamente riutilizzare o riorganizzare opportunamente in un formato aggregabile.

Nel corso degli ultimi anni, il volume di *Open Data* (dati aperti) pubblicato sul Web è cresciuto enormemente, aumentando l'interesse sia di istituzioni pubbliche e private sia dei cittadini.

Il motivo di tale interesse è da ricercarsi in quella evoluzione che ha subito il Web, conosciuta con il nome di *Linked Data*. Quest'ultima è una metodologia che permette di aggregare e collezionare i dati provenienti da fonti distribuite [BHAR07], con la consapevolezza del fatto che i dati, se isolati, hanno poco valore, ma che tale valore aumenta enormemente nel momento in cui diversi *dataset* indipendenti vengono incrociati liberamente.

Con tale evoluzione il Web, da semplice spazio di raccolta e collegamento di documenti, diviene uno spazio in cui i dati grezzi contenuti nelle risorse sono collegati o facilmente collegabili dalle macchine. Per effettuare tali collegamenti, i dati devono essere pubblicati sotto delle condizioni di utilizzo "aperte" ovvero condizioni che ne consentano la consultazione, navigazione e aggregazione.

La corrente di pensiero dell'*Open Data* risponde al meglio all'esigenza di poter fruire di dati aperti (e quindi liberamente usabili per qualsiasi scopo), andando di

fatto a rappresentare quell'infrastruttura di cui il *Linked Data* necessita per creare una rete di connessioni tra i dati sparsi sul Web. Questo argomento è trattato in modo approfondito nel paragrafo 2.3.

Un esempio del loro effettivo utilizzo è da ricercarsi negli USA dove l'informazione riguardante il settore federale è divenuta di pubblico dominio [PA11]. Ciò conferma che il *Linked Data* è ormai una tecnologia abbastanza matura e con enormi potenzialità, ma che richiede una grande quantità di dati tra loro collegati, ovvero *linkati*, per poter concretamente divenire utilizzabile ed applicabile.

Un contributo molto importante in questa direzione è stato dato da progetti come DBPedia [ABKLCI07], una raccolta di dati semi-strutturati riutilizzabili e facilmente interrogabili che ha come obiettivo il consumo di tali dati da parte sia di software sia di utenti.

Grazie a progetti come DBPedia, che, come spiegato meglio nel paragrafo 2.3.1, garantiscono la disponibilità dei dati e la loro piena riutilizzabilità, il *Linked Data* ha la possibilità di offrire una rappresentazione molto ricca degli stessi (in quanto a relazioni), così convergendo, insieme all'*Open Data*, in quello che è conosciuto come approccio *Linked Open Data*.

Purtroppo, ancora oggi si devono affrontare una serie di difficoltà quando si cerca di accedere, comprendere o utilizzare gli *Open Data*. In particolare, è spesso necessario effettuare uno sforzo manuale, sia per l'analisi, sia per l'estrazione di dati.

L'utilizzo di *Open Data* è minacciato da tale sforzo, che rende molto arduo riorganizzare i dati in un formato comprensibile su cui operare in base alle proprie

esigenze, e limita quello che potrebbe essere il loro reale contributo alla comunità.
Uno dei sistemi che affrontano questi argomenti è *Tableau* [CSH03], una soluzione interattiva che permette di analizzare uno o più *dataset* per poi effettuare accurate valutazioni statistiche su qualsiasi ambito, lasciando totale libertà all'utente finale su come incrociare i dati che sono stati riconosciuti ed estratti dai *dataset* processati. Esso quindi interpreta i *dataset*, li predispone per l'aggregazione e infine lascia al fruitore il compito di collegare effettivamente i dati in base alle proprie esigenze, affermandosi come una delle possibili applicazioni derivanti dall'analisi dei dati.
I principali limiti di questo software sono l'impossibilità, da un lato, di personalizzare il processo di analisi, ovvero decidere quali entità riconoscere e come riconoscerle, e dall'altro, di utilizzare i risultati derivati dall'analisi per scopi differenti dalla visualizzazione grafica. Il ruolo di Tableau viene trattato in modo più approfondito nel paragrafo 2.5.
A differenza di *Tableau*, il *Semantic Detector*, risultato di questa tesi, interroga le ontologie e i dati aggregati contenuti in DBPedia per risalire alla tipologia e alla semantica del dato analizzato, potendo così contare su definizioni sempre aggiornate che coprono molti ambiti di interesse diversi. Lo scopo del sistema è quello di fornire una descrizione strutturata dei *dataset* e del loro contenuto affinché i dati possano essere trattati in modo automatico, eliminando lo sforzo manuale necessario in molti casi per capire cosa essi siano concretamente e come siano stati memorizzati. Del sistema e del suo contributo se ne parla in modo più dettagliato nel capitolo 3.
Progettato con un'architettura software modulare ed estendibile, il processo di

analisi del *Semantic Detector* può essere personalizzato per soddisfare le esigenze specifiche di rappresentazione dei dati finali e modellazione degli stessi. Tale architettura, insieme all'utilizzo di DBPedia, costituisce uno dei punti di forza di questo progetto poiché consente una personalizzazione elevata dei parametri di analisi sia all'utente (o al software cliente), che può decidere cosa riconoscere e come riconoscerlo, sia al fornitore del servizio, che può implementare facilmente nuovi meccanismi di analisi e fornirne la sintassi di utilizzo all'utente senza dover modificare il *Core* dell'applicazione. Nel capitolo 4 viene mostrata nel dettaglio la struttura e la progettazione del software.

Per valutare l'efficienza e l'affidabilità del *Semantic Detector*, durante lo svolgimento della ricerca sono stati effettuati test su numerosi *Open Data* trovati in rete. I risultati, riportati in dettaglio nel capitolo 5, hanno confermato che, incrociando analisi sintattiche statiche con le definizioni di DBPedia, è possibile risalire alla semantica del dato con una discreta sicurezza e velocità. Dall'altra parte, i test hanno anche evidenziato la presenza di alcune imperfezioni legate alla frequente disorganizzazione dei dati contenuti all'interno dei *dataset* trovati in rete. Questi ultimi, probabilmente non pensati inizialmente per essere analizzati da sistemi automatici, richiedono spesso la realizzazione di meccanismi di riconoscimento non standard allo scopo di adattarsi alle sintassi *scorrette* più diffuse (per esempio, nonostante esistano sintassi universali per gli indirizzi geografici del mondo, tale dato viene frequentemente archiviato con una sintassi variabile e personalizzata). A tal proposito un'interessante estensione futura del *Semantic Detector* potrebbe essere la realizzazione di meccanismi di *Machine Learning* che, a partire da un set di dati dello stesso tipo, consentano al

programma di imparare e generare dei modelli (*pattern*) che riconoscano la sintassi di quel determinato dato ed utilizzare tale modello nelle analisi future, aumentando considerevolmente le capacità di riconoscimento del sistema.

Le sperimentazioni effettuate durante la Tesi, su *Open Data* reali e con configurazioni sensate, dimostrano che utilizzando *Semantic Detector* si riesce effettivamente ad ottenere una tipizzazione dei dati molto affidabile (con precisioni intorno al 70%). Analizzando solo il 5 - 10% del contenuto dei dataset, e sfruttando DBPedia per riconoscere le entità e per classificarle utilizzando sintassi semanticamente note, si riesce a predisporre i dati per un loro riutilizzo immediato (come *Tableau*) o per una successiva conversione in *Linked Open Data* (riorganizzandoli ad esempio in *triple RDF - Resource Description Framework*) permettendo di escludere gradualmente, se all'interno di in una catena di processi software, l'intervento manuale e dando quindi un contributo molto importante al sempre più crescente volume di dati aggregati presenti nel Web.

.

2. L’evoluzione del Web

2.1 Web of Data

Sin dalla sua nascita, il Web ha significativamente cambiato il modo in cui un autore può condividere informazioni e contenuti di vario genere, facilitandone la pubblicazione e la consultazione da parte dei lettori, i quali hanno potuto finalmente accedere e navigare liberamente tra una grande quantità di materiale caricato da altri. In tal modo, il Web diventa un contenitore di informazioni che possono essere catalogate in un archivio della conoscenza umana estremamente vasto. Tali informazioni sono collegate tra loro attraverso dei collegamenti ipertestuali chiamati *hyperlink* che permettono di creare delle associazioni tra differenti informazioni e parole chiave fornendo al lettore dei rapidi accessi ad altre fonti di informazione inerenti a quella a cui stava accedendo. Da sempre il Web ha mantenuto queste caratteristiche affermandosi come una tecnologia estremamente utile ed in perenne crescita [BHT09].

Un punto di svolta fondamentale che ne ha favorito l’evoluzione, è stato il passaggio dalla condivisione di documenti come unità di base a quella di dati grezzi (*raw*), divisibili in più parti collegabili tra loro o collegabili ad altre informazioni arricchendo l’informazione stessa. Grazie a questo cambiamento, il Web diviene ufficialmente uno spazio globale in cui condividere i dati, dando vita al *Web of Data*. I dati contenuti nelle risorse possono adesso essere strutturati affinché risultino interrogabili in un secondo momento e, in risposta alle esigenze

dell’utente, aggregabili con altri. Solitamente un insieme isolato di dati contiene informazioni circa un singolo argomento. Se molteplici insiemi fossero interconnessi tra loro, un sistema automatico avrebbe la possibilità di navigare tra più ambiti della conoscenza sfruttando i collegamenti tra le varie risorse connesse in un’unica grande rete.

Tipicamente nel Web siamo abituati ad accedere alle informazioni in formato HTML mediante degli spazi di aggregazione conosciuti da tutti come “siti”. In seguito alla sempre più crescente necessità di potervi accedere utilizzando sistemi automatici, le risorse hanno iniziato ad essere pubblicate sotto forma di dati strutturati, come ad esempio tutti quei dati provenienti da tabelle eterogenee ed esposti successivamente sul Web in formati quali XML, CSV o semplici tabelle HTML, perdendo così gran parte della semantica del dato iniziale. Tale perdita è dovuta al fatto che l’HTML non è un linguaggio abbastanza espressivo per gestire i collegamenti tra varie risorse, ed è per questo motivo che nasce l’esigenza di descrivere i dati in una forma strutturata che porterà all’evoluzione conosciuta come *Linked Data*.

2.2 Linked Data e Web Semantico

Dare una definizione formale di *Linked Data* non è così semplice come si potrebbe pensare in quanto si tratta in realtà di un concetto molto complesso. Potremmo semplificarne il significato affermando che il *Linked Data* è un insieme di *best-practice* inerenti alla pubblicazione e interconnessione dei dati sul Web allo scopo di poter essere in futuro interrogabili da una macchina [BHAR07].
Le risorse, referenziate tramite un URI (*Uniform Resource Identifier*) ed inerenti ad un determinato dominio, sono collegate ad altre risorse esterne al dominio e così via, creando un *reticolo* di conoscenza navigabile e sempre più esteso.
Il *Linked Data* è fortemente connesso al concetto di *Semantic Web* che, nonostante non abbia particolari pretese tecniche, ha bisogno di alcune regole atte a creare dei contenuti accessibili da una macchina. Entrambi appartengono in realtà allo stesso ambito, ovvero il *Linked Data* è la tecnologia necessaria per la realizzazione del Web semantico. È interessante esaminare come Tim Berners-Lee, creatore del www (*World Wide Web*), definisce il *Semantic Web*: "*A web of things in the world, described by data on the web*" quindi una rete di *cose* del mondo descritte tramite i dati nel Web. Con questa espressione egli fa preciso riferimento a concetti importanti come la rete, il *reticolo*, le cose e i dati.
Da qui è chiara la distinzione che vi è tra il Web tradizionale e Web Semantico (o *Web of Data*): il primo è uno spazio piatto di condivisione di documenti HTML collegati tra loro tramite *hyperlink*,mentre il secondo è un contenitore di oggetti (non della loro rappresentazione) la cui concretezza si oppone all’astrattezza del

Web tradizionale.
Il *Semantic Web* quindi non vuole sostituirsi al Web tradizionale, bensì vuole estenderlo per riuscire a plasmare un mondo in cui “*i meccanismi quotidiani del commercio, della burocrazia, e delle nostre vite quotidiane saranno gestiti da macchine che interagiscono con altre macchine, lasciando agli umani il compito di fornire l’ispirazione e l’intuizione”* (Tim Berners-Lee, *L’architettura del nuovo Web*, 1999).

2.3 Open Data

Risulta evidente che il reticolo di *Linked Data* non può esistere senza una discreta quantità di dati pubblicati e interconnessi tra loro, pubblicazioni che devono poter contare su delle condizioni d’uso "aperte". L’*Open Data* (dati aperti) è quella corrente di pensiero che sembra rispondere al meglio alla necessità di poter contare su dati liberamente pubblicabili e consultabili, configurandosi come quell’infrastruttura di cui il *Linked Data* ha bisogno per creare quel reticolo di informazioni aggregate partendo dai dati sparsi nel Web. Ma quali sono i vantaggi concreti del modello *Open Data*?
Il principale vantaggio di questo modello è sicuramente l’interoperabilità dei dati: mentre i dati isolati hanno un valore fine a se stesso, se collegati ad altri archivi di dati (i *dataset*) pubblicati indipendentemente allora tale valore cresce sensibilmente e consente al fruitore di navigare tali collegamenti in totale libertà.
La pubblicazione di risorse *Open Data* ovviamente richiede che siano rispettate alcune regole di *best-practice*. Tale richiesta spinge i creatori delle informazioni a

pubblicare insiemi di dati grezzi lasciando al lettore, e non più al fornitore, la decisione riguardo la forma del dato. Dall’altra parte, invece, il fornitore decide esclusivamente il contenuto del dato.
Per garantire il riuso dei dati quindi occorre poter mescolare liberamente i *dataset* stabilendo dei collegamenti diretti, detti *link*, qualora dati provenienti da diverse fonti si riferiscono alla stessa risorsa o a risorse interconnesse tra loro. Con tali collegamenti si ha la possibilità di effettuare dei veri e propri salti da un *dataset* ad un altro e questo è il motivo per il quale questo scenario possiede delle potenzialità enormi ed in continua espansione. Per esempio, basti guardare la quantità di dati in possesso delle pubbliche amministrazioni, dati che potrebbero, e secondo alcuni "dovrebbero" in quanto dati raccolti per finalità pubbliche [AB04, PB08], essere resi disponibili al pubblico (nel rispetto ovviamente della privacy dei cittadini). A conferma di ciò, un esempio significativo è dato dal governo Americano e dal governo Inglese in quanto per primi hanno dato vita a un portale di raccolta e consultazione di cataloghi di dati aperti, rispettivamente *data.gov* e *data.gov.uk*.

2.3.1 DBPedia e lo standard RDF

RDF (*Resource Description Framework*) è il modello utilizzato per strutturare i *Linked Data*, uno standard portato avanti dal W3C (*World Wide Web Consortium*) con lo scopo di preservare la semantica delle risorse e le relazioni tra esse.

L’informazione viene codificata in asserzioni (le *triple RDF*) così suddivise:

- soggetto, la 'cosa' descritta
- predicato, la proprietà del soggetto
- oggetto, il valore di tale proprietà

dove ognuno dei tre elementi dovrebbe, secondo Tim Berners-Lee, essere rappresentata utilizzando URI de-referenziabili. Nonostante questo aspetto non sia obbligatorio, considerando che dall’aumento della quantità di URI ne deriva una maggiore riusabilità dell’informazione, questa buona norma di fatto risulta essere un *must* molto importante. Le triple, che in questo modo rappresentano le risorse e le loro proprietà sotto forma di Grafi, vengono codificate utilizzando la sintassi XML-*based* (*EXtensible Markup Language*) affinché risultino facilmente leggibili ed utilizzabili sia da una macchina nata per leggere quei dati sia da una macchina nata per scopi differenti.
Questa è la struttura che caratterizza le risorse pronte ad essere integrate in sistemi di raccolta e connessione di *Linked Data* quali DBPedia, un progetto per l’estrazione di informazioni strutturate da Wikipedia e per il rilascio di queste informazioni sul Web come *Linked Open Data* in formato RDF. DBPedia contiene al suo interno un’altissima quantità di risorse strutturate in formato RDF e interconnesse con altre risorse, un’aggregazione di grafi nei quali è possibile navigare a proprio piacere o estrarre informazioni più o meno complesse utilizzando ad esempio degli *end point* (http://dbpedia.org/sparql) che accettano '*query*' SPARQL (*SPARQL Protocol and RDF Query Language*). Le risorse qui

sono state classificate manualmente in base ai contesti più frequenti ai quali si riferivano, le Ontologie, che specificano in modo significativo i concetti e le relazioni che caratterizzano un certo dominio della conoscenza.

2.3.2 Le 5 Star Open Data

Tim Berners-Lee ha suggerito uno schema di classificazione degli *Open Data* introducendo una scala a cinque stelle per valutare quanto i dati siano, non solo a livello formale, ma anche a livello sostanziale, aperti ed accessibili (Figura 1).

Tim Berners-Lee indica cinque criteri, a ciascuno dei quali assegna una “stella” se il *dataset* rispetta un determinato criterio:

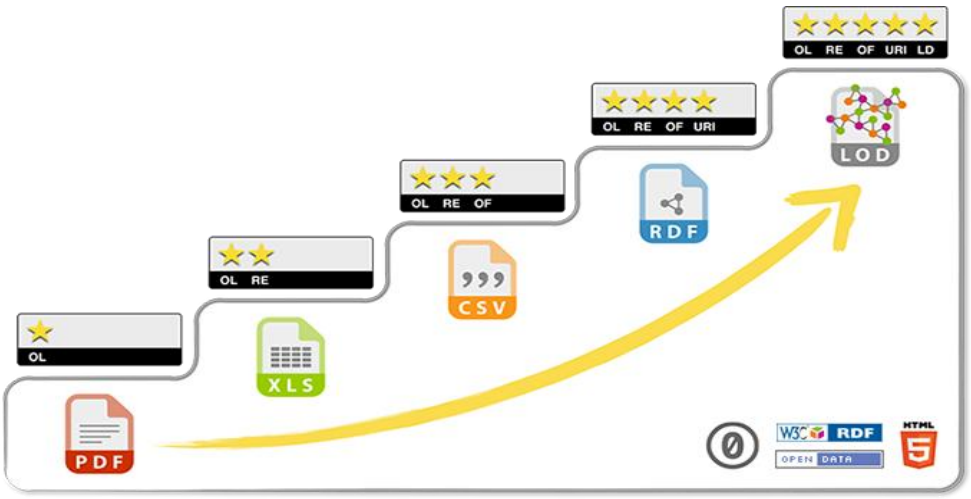
1. Il dato è disponibile sul Web (in qualsiasi formato) con una licenza aperta.

2. Il dato è disponibile in un formato strutturato (per esempio, un foglio di calcolo Excel piuttosto che la scansione di una tabella).

3. Il dato è in un formato non proprietario (CSV un formato preferibile ad Excel in quanto non soggetto a licenza).

4. Il dato è denotato da URIs in modo che le altre persone possano raggiungere tali dati.

5. Il dato contiene collegamenti ad altri dati fornendo un contesto alle proprie informazioni.



Firuga 1: 5 star Open Data, http://5stardata.info/en/

Risalendo la scala di classificazione, il dato racchiude sempre di più le caratteristiche necessarie affinché questo dato risulti navigabile e soprattutto collegabile ad altre risorse del reticolo della conoscenza.

In particolar modo, questo lavoro si è concentrato sull’interpretazione automatica degli *Open Data* a due e tre stelle sia per la loro numerosa presenza in rete, sia perché comprendono quei *dataset* che spesso presentano al loro interno o assenza

di significato (riferito al dato contenuto) o una maggiore propensione ad utilizzare una struttura interna molto personalizzata (in particolare nei formati come XML e JSON, *JavaScript Object Notation*), rendendo molto laborioso per una macchina agire direttamente su di essi. In questo contesto, si è cercato di inserire un sistema 'rivelatore', un *Detector*, capace di identificare la struttura di questi *dataset* e di identificare entità note già presenti all’interno del retico di *Linked Open Data*. Gli *Open Data* che rispecchiano gli standard a quattro stelle sono risorse che sono state strutturate seguendo le regole, insieme con le buone norme, del formato RDF e quindi pronte per essere interrogate e pubblicate.

2.4 Pubblicare Linked Data

Affinché i dati possano essere pubblicati ed aggregati ad altre entità, essi devono necessariamente rispettare le linee guide introdotte dall’approccio *Linked Open Data* e dal formato RDF.
Gli enti pubblici, privati e i cittadini sono costretti a dover manualmente intervenire su tutti quei *dataset* già esistenti che, in quanto 'datati', non sono stati elaborati seguendo tali linee guida. Solitamente tutti questi dati sono stati creati utilizzando formati come il CSV (*Comma Separated Value*) e XML oppure risultano essere estratti di tabelle relazionali di vecchi database. Se quindi da un lato può risultare semplice creare un archivio di dati secondo le norme che ne consentono la pubblicazione, risulta spesso molto arduo riciclare vecchi archivi e renderli pubblicabili senza perdere dati significativi e senza intervenire manualmente.

Solitamente i *dataset* strutturati come SQL (*Structured Query Language*), XML e CSV conservano informazioni circa delle entità e le sue proprietà. Ad esempio, un *dataset* anagrafico conterrà dati circa persone insieme con proprietà (età, indirizzo, telefono ecc.) di tali persone. Se strutturato, un archivio dati risulta essere più facilmente interpretabile da una macchina ma nonostante ciò ha bisogno di essere convertito utilizzando le regole tipiche del nuovo standard.
Sono stati sviluppati numerosi software al fine effettuare le conversioni sopra citate; nel caso di database relazionali, esempi significativi sono tutti quegli strumenti chiamati RDBtoRDF (*Relational Database to RDF*) come Virtuoso

(http://virtuoso.openlinksw.com/) e D2R Server [BC06].
D2R Server in particolare, utilizzando il D2RQ *Mapping Language* per il mapping dei contenuti, consente ai browser HTML e RDF la navigazione di database relazionali e permette a terzi software di interrogare tali database tramite *query* SPARQL. Ogni regola del D2RQ *Mapping Language* specifica *come* debbano essere assegnati gli URIs alle risorse e *quali* siano le proprietà migliori per descriverle.

Altri esempi molto significativi sono tutti quei *tool* che, nel caso di analisi da effettuare su del testo non strutturato, basano il proprio funzionamento su meccanismi di *Machine Learning*, quali Weka [HFHPRW09] e Rapid-Miner [RBP15]. Di particolare interesse è Rapid-Miner per la sua estensione *Linked Open Data*. Con tale estensione, gli utenti hanno accesso ai *Linked Open Data* consentendone il riutilizzo in più o meno sofisticati meccanismi di analisi dei dati senza la necessità di conoscere SPARQL o RDF.

2.5 Interpretare gli Open Data

Un altro approccio interessante è quello usato da strumenti come Wrangler [KPHH11] o OpenRefine [VM12] per estrarre, pulire e trasformare *dataset* semi-strutturati (ad esempio tabellari) in *Linked Data*. L’obiettivo di questi strumenti è quello di facilitare agli utenti il compito di analisi degli *Open Data* e la conversione degli stessi in *Linked Open Data*. Sebbene molto efficace, la

conversione fornita da tali strumenti non è completamente automatica e in alcune fasi richiede ancora un intervento umano.
Tableau è un software che più di altri può essere preso come esempio per vedere concretamente come, partendo da *dataset* più o meno strutturati, è possibile ottenere dei risultati applicabili nel mondo. A conferma della sua bontà ricordiamo una serie di riconoscimenti che gli sono stati attribuiti nel corso degli anni: "*Best Overall in Data Visualization*" DM Review, "*Best of 2005 for Data Analysis*" PC Magazine, ["*Software: Business - Tableau - Best of the Year*". PCMag.com. November 11, 2005] e "*2008 Best Business Intelligence Solution (CODiE award)*" Software & Information Industry Association [*"2008 Codie Award Winners*". SIIA.net].

Tableau riesce a produrre molteplici tipologie di visualizzazione dati focalizzare principalmente per lo studio di nuove strategie di Business. Disponibile in diverse versioni (*desktop*, *server*, *reader*, ecc.), questo software prende in input uno o più *dataset* e li analizza utilizzando delle proprie meccaniche di riconoscimento per poi, a seconda delle entità rilevate all’interno di essi, offrire al fruitore in automatico delle tipologie di visualizzazione dei dati il più inerenti possibile alla tipologia degli stessi. Se ad esempio venissero rilevati dei dati contenenti nomi di città note, Tableau, tra le varie opzioni, suggerirebbe una visualizzazione su mappa in cui sarebbe possibile visualizzare dei *marker* in corrispondenza di tali città; se insieme ad esso gli fornissimo un *dataset* contenente dati anagrafici, allora il software ci permetterebbe di incrociare i due *dataset* analizzati e potremmo vedere magari in quali città si sono maggiormente concentrate le

persone. Ovviamente questo è solo un esempio per far capire che tipo di operazioni consente di effettuare questo software. Il punto veramente interessante da rilevare è che, partendo da dati privi di semantica e indipendenti tra loro, il software permette di ottenere, analizzandoli e aggregandoli, dei risultati estremamente concreti ed applicabili, confermando la tesi secondo cui, se opportunamente interconnessi, i *Linked Data* possono oggi effettivamente essere riutilizzati da tutti per ottenere dei risultati utili per il domani.

I limiti di questo software sono principalmente due: in fase di pre-analisi non è possibile intervenire per modificare le meccaniche o le politiche di anilisi e non è possibile configurare dei riconoscimenti personalizzati, costringendo l’utente a lavorare solo con ciò che il software è già in grado di riconoscere; in fase di post-analisi il limite sta nell’utilizzo stesso dei risultati dell’analisi in quanto consente di lavorare con i dati analizzati esclusivamente a livello grafico. *Tableau* quindi nonostante offra una potente interfaccia interattiva con cui incrociare i dati per molteplici finalità, non riesce a dare la libertà di personalizzazione e riutilizzo che ne consentirebbero l’inserimento in un contesto più vasto di elaborazione degli Open Data.

3. Semantic Detector

Semantic Detector è un tool sviluppato per dare un contributo importante in questo contesto storico in cui la quantità di dati presenti in rete e possibilmente analizzabili e aggrebabili è eguagliata dalla grande quantità di sforzo umano ancora richiesto per la loro effettiva conversione.

Non si è cercato solo di far meglio rispetto ad alcune soluzioni esistenti bensì si è cercato di creare qualcosa di genuino ed universale la cui parola d'ordine è 'espandibilità'. Se utilizzato al meglio potenzialmente non vi sono grandi limiti per ciò che concerne l'analisi dei dati e soprattutto non vi è alcun limite nell'utilizzo dei risultati di tale tool in applicazioni successive.

Il funzionamento di Semantic Detector concettualmente è molto semplice: riceve in ingresso l'URI di un datasets o una lista di URIs, analizza la struttura interna (che sia tabellare o strutturata), effettua delle valutazioni sintattiche e semantiche sui dati contenuti in tali datasets e restituisce al fruitore un output strutturato (in formato XML ad esempio) contenente sia una rappresentazione della struttura interna del dataset sia dei valori semantici che ne definiscono la tipologia. Tale output può essere visto quindi come una descrizione completa di un dataset, un report di analisi strutturato processabile da altri software più ad alto livello sia per scopi finali sia all'interno di una catena di processi più vasta, motivo per cui il reale utente di questo tool non è principalmente un essere umano bensì una qualsiasi macchina abilitata a processarne l'output in quanto la sua struttura non cambia al variare del tipo di dataset di cui ne è la descrizione.

3.1 Utilizzo del sistema

In questo paragrafo viene descritto come è possibile interrogare il sistema per sfruttarne le funzionalità. Vi sono principalmente due modi con i quali è possibile interagire con il sistema a seconda delle necessita: il primo, forse meno significativo, è l'utilizzo di un'interfaccia Web-based (quindi una comune pagina HTML) dalla quale è possibile inserire l'URI del dataset che si vuole processare; il secondo canale, sicuramente più utile da sfruttare, è un'interfaccia REST messa a disposizione dal tool alla quale è possibile mandare, con delle classiche richieste HTTP, l'URI del dataset da analizzare ottenendo in risposta l'output in formato strutturato.

Il motivo per cui si predilige questa seconda opzione è intrinseco nello scopo stesso di questo lavoro di tesi ovvero la ricerca di una soluzione che sostituisca l'intervendo umano, quindi è necessario ed utile allo scopo che sia una macchina ad effettuare tali richieste e ad interpretarne i risultati. In effetti la prima opzione non è altro che una maschera utente che se utilizzata non farà altro che effettuare le richieste al sistema esattamente come farebbe un qualsiasi altro programma e stamperà l'output otenuto direttamente a video per una visualizzazione 'umana' (quindi potremmo dire fine a se stessa).

Una volta ricevuto il dataset, il sistema effettuerà principalmente tre macro operazioni:

1. Analisi della struttura
2. Analisi dei contenuti
3. Creazione dell'output

L'analisi della struttura è la fase iniziale del processo che si occupa sia di rilevare il formato del dataset sia di estrarre la struttura ricorrente interna utilizzata in questo dataset per conservare i dati. Ma quali e quanti formati è possibile analizzare? Potenzialmente qualsiasi. Un'affermazione cosi forte necessita di essere motivata e tale motivazione è da ricercarsi nella già accennata espandibilità: il sistema è predisposto a supportare qualsiasi formato in ingresso in quanto qualsiasi formato, come spiegato più in dettaglio nel capitolo 4, è in qualche modo convertibile in una struttura comune. Quindi al sistema non importa che tipo di dataset si sta processando.

Rilevata la struttura del dataset il sistema procede con l'analisi dei contenuti, ovvero inizia ad effettuare delle valutazioni ragionate sui dati mirate ad identificarne il significato o la tipologia fisica (banalmente se un campo del dataset contiene un intero, la minima risposta che si otterrà sara appunto Integer). Tutte queste valutazioni sono configurabili ed espandibili; al sistema è possibile indicare, se voluto e se necessario, esattamente cosa si vuole riconoscere e quali requisiti debba avere un'entità per rientrare in una determinata classificazione.

Rilevata la struttura e la semantica dei dati il sistema conclude il suo lavoro generando un report strutturato in cui riporta i risultati ottenuti con i punti precedenti. Il risultato sarà una perfetta e riassuntiva immagine di come è strutturato il dataset e di quali entità interessanti esso è portatore.

3.2 Personalizzare l'analisi

Poter personalizzare l'analisi è uno dei punti forti del sistema. Ad esso è possibile specificare cosa riconoscere, con quale meccanismo riconoscere, quali fonti di dati sfruttare e come etichettare le entità. Questo è possibile affiancando il dataset che si vuole analizzare ad un file di configurazione in formato JSON come riportato di seguito.

```
{
  "type": "2%",                                    1
  "output": "xml",
  "modules": {
    "pattern": [                                   2
      {
        "regex": "^(.+)@(.+)(\.)([a-zA-Z]+)$",
        "label": "EmailAddress",
        "goal": "10%"                              3
      },
      {
        ...
      }
    ],
    "list": [                                      4
      {
        "goal": "30%",
        "label": "BloodGroup",
        "items": ["A","B","AB","0","A+","B+","AB+","0+","A-","B-","AB-","0-"]   5
      },
      {
        ...
      }
    ],
    "dbpedia": [                                   6
      {
        "goal": "10%",
        "label": "DBPedia_Location",
        "criteria": "MATCH",                       7
        "searchTypes": ["PopulatedPlace", "Place", "Settlement", "Location", "Country"]   8
      },
      {
        "goal": "10%",
        "label": "DBPedia_Person",
        "criteria": "CONTAINS",
        "searchTypes": ["Agent", "Person", "NaturalPerson", "GivenName"]
      }
    ],
    "embedded": [                                  9
      {
        "goal": "10%",
        "id": "isDateTime"                         10
      },
      {
        ...
      }
    ]
  }
}
```

In cima al documento (1) sono presenti i primi due parametri di configurazione:

- Type: in questo campo si specifica in percentuale la porzione di dataset che si vuole analizzare. Spesso non è necessario processare tutti i dati per identificarne la tipologia, si andrebbe altrimenti incontro a delle tempistiche di elaborazione potenzialmente elevate in quanto proporzionali alla quantità di dati da processare. Da una certa percentuale in poi, statisticamente, non si ottengono più informazioni aggiuntive rispetto a quelle già ottenute.

- Output: in questo campo si ha la possibilità di decidere in che formato si desidera ricevere l'output (XML, JSON).

A seguire nel documento vi sono le configurazioni relative alle meccaniche ed alle modalità di riconoscimento delle entità. Con queste istruzioni è possibile accedere a tutte le meccaniche di riconoscimento attualmente implementate (rigorosamente espandibili) e sfruttarne a propria discrezione il funzionamento. Per ognuna delle meccaniche è possibile configurare il numero di regole che si desidera, libertà che deve però essere gestita opportunamente (ragionando su quantità critiche è un dato di fato che maggiori sono le regole che si inseriscono e maggiore sarà il tempo di esecuzione).
Pattern (2), la prima meccanica, sfrutta politiche di *matching* utilizzando Espressioni Regolari. Una singola regola (3) di tipo Pattern è costituita da una *regex* (l'espressione regolare da utilizzare), una *label* (l'etichetta che si vuole dare

alle entità che fanno match con con la relativa *regex*) e *goal* (la soglia di accettazione). La soglia di accettazione è un valore in percentuale per indicare la quantità di dati, ripetto al totale degli analizzati, che devono essere riconosciuti con la regola a cui appartiene affinchè l'etichetta di tale regola compaia in output. La presenza di tale valore deriva dalla necessità di avere meno palsi positivi dovuti a match fortuiti ed isolati.

List (4) sfrutta invece politiche di contenimento. L'unica differenza rispetto a Pattern, che vuole una RegEx, è che List necessita di una lista di elementi (5) alla quale il dato in input deve appartenere affinchè il match abbia successo.

Embedded (9) racchiude tutte quelle meccaniche di riconoscimento, che si possono o meno utilizzare, che non necessitano di configurazione esterna. Solitamente queste sono o meccaniche per identificare qualcosa di sintatticamente uguale per tutti (per sempio date, coordinate geografiche ecc.) o meccaniche complesse mirate a precisi riconoscimenti. L'id (10), corrispondente al nome della regola *embedded* che si vuole utilizzare, è l'unico campo necessario per sfruttare tale regola.

Dbpedia (6) è sicuramente la meccanica più interessante in quanto consente di consultare i Linked Data presenti in DBPedia. Vi sono attualmente due configurazioni possibili: la prima, utilizzando come criterio (7) di ricerca *MATCH* , consiste nel cercare delle risorse la cui etichetta combaci esattamente con il dato in input e la cui tipologia sia una di quelle specificate in *searchTypes.* L'utilità di questa regola, oltre alla consultazione di DBPedia, è poter selezionare tipologie più specifiche o inerenti già note e racchiuderle sotto un unica etichetta. Nell'esempio sono state selezionate le tipologie che più comunemente sono

attribuite ai luoghi in DBPedia (varietà dovuta all'effettiva differenza che vi è tra una città, una nazione, un villaggio ecc.) e si è deciso di assegnare l'etichetta più generica *DBPedia_Location* all'occorrenza di una delle tipologie coinvolte.
Si ha inoltre la possibilità di utilizzare il criterio di ricerca CONTAINS che, a differenza di MATCH, consiste nel cercare delle risorse la cui etichetta contenga al suo interno almeno il dato in input. Tale strategia può essere utile per identificare nomi propri cercandoli all'interno di nomi estesi, per luoghi il cui nome si trova scritto spesso in modo differenza (per esempio New York City e New York) e volendo per strategie piu complesse che si basato sulla valutazione parziale del dato in input.

3.3 Comprensione dei risultati

Vediamo un esempio di cosa produrrebbe in output il sistema se gli dessimo un dataset CSV strutturato come segue.

Persone

Nome	Cognome	NatoA
Mario	Rossi	Bologna
Francesco	Bianchi	Milano

Di seguito il risultato ottenuto da tutto il processo descritto nel paragrafo precedente.

```
<analysis>
    <parameters>
        <analysisType>10%</analysisType>
        <datasetURL>http***.csv</datasetURL>
        <recordsCount>100</recordsCount>
        <recordsAnalyzed>10</recordsAnalyzed>
    </parameters>
    <items>
        <item>
            <label>ROOT</label>
            <children>
                <item>
                    <label>Nome</label>
                    <meaningSet>
                    <value confidence="5">dbo:GivenName</value>
                    <value confidence="5">dbo:Person</value>
                    </meaningSet>
                </item>
                <item>
                    <label>Cognome</label>
                    <meaningSet>
                    <value confidence="8">dbo:Person</value>
                    </meaningSet>
                </item>
                <item>
                    <label>NatoA</label>
                    <meaningSet>
                    <value confidence="8">dbo:PopulatedPlace</value>
                    <value confidence="8">dbo:Location</value>
                    </meaningSet>
                </item>
            </children>
        </item>
    </items>
</analysis>
```

La formattazzione delle informazioni riportate nell'output consentono una facile lettura dello stesso.
Partendo dall'inizio, si può leggere che il dataset CSV analizzato contenente 100 record, di cui solo il 10% è stato processato, presenta una struttura composta da tre elementi (i cui nomi rilevati sono Nome, Cognome e NatoA) allo stesso livello di annidamento e che rispettivamente contengono dati di tipo **GivenName**, **Person** e **PopulatedPlace.**

Considerando questo esempio è importante sottolineare che questi risultati sono stati ottenuti in maniera totalmente automatica, ovvero non si è intervenuti manualmente per stabilire ad esempio che una determinata cella contenesse un *PopulatedPlace* (valutazione abbastanza semplice da fare per un utente). Questo è stato possibile sfruttando DBPedia come database universale: il sistema, dato un input da valutare, cerca dentro DBPedia un'entità denominataesattamente come l'input oppure strettamente pertinente con essa. Banalmente, nel valutare la semantica di *Bologna* il sistema rileverà che *Bologna* è un'entità nota ed è una città; nel valutare *Mario* invece il sistema si accorgerà che è spesso presente nel nome per esteso di entità *Person* note a DBPedia e, nello specifico, alla loro proprietà *GivenName* (nome proprio). Il sistema quindi rileva delle entità significative e ne riporta la tipologia utilizzando le sintassi ed i riferimenti a risorse note dello stesso dominio.

Nel report, oltre alla tipologia delle entità rilevate, è possibile vedere alcuni dettagli molto interessanti derivati dall'analisi:

- **Confidence**: la *confidence* indica quanti input, rispetto al totale degli analizzati, sono stati classificati secondo quella specifica definizione. Se analizzo dieci nomi di luoghi ma solo sei di essi vengono riconosciuti dal sistema, nel report troveremo (relativamente alla definizione di luogo) una confidenza di sei. Il tolate degli analizzati è indicato tra i parametri in testa al documento sotto la voce *recordsAnalyzed*. La confidenza è molto importante perchè, associato ad una tipologia, fa la differenza tra un risultato affidabile e un risultato casuale, riducendo cosi il rischio di utilizzo di tipologie scorrette.
- **Annidamenti**: la struttura interna del report, nello specifico entro il tag *items*, è potenzialmente sempre differente poichè deve rispecchiare la struttura del dataset. In questo specifico esempio, essendo un CSV, tutti i dati sono organizzati in righe le cui colonne stanno ovviamente allo sesso livello. Per questo motivo i tre elementi (i tre *item*) contenuti (*children*) in ogni riga del documento (*ROOT*) sono stati inseriti uno dopo l'altro. Il discorso è diverso se si anlizza un dataset che ha una struttura interna non tabellare come ad esempio un documento XML e JSON. In questi casi gli *item* non saranno posizionati allo stesso piano bensì manterranno lo stesso livello di annidamento che presentano nel dataset originale, così facendo l'informazione circa la struttura interna non viene semplicemente riportata bensì viene mantenuta.

Sfruttando l’output di *Semantic Detector* è sicuramente facile per una macchina leggere e comprendere le informazioni ottenute in seguito all’analisi di un dataset. Altrettanto facile di conseguenza è il riutilizzo di tali informazioni per scopi vari, dalla creazione di visualizzazioni interessanti alla riorganizzazione in formati semanticamente più adatti. Se configurato opportunamente *Semantic Detector* può essere inserito con estrema facilità all’interno dell’attuale catena di processi necessari per l’analisi degli Open Data. In particolare dovrebbe essere inserito tra i dati e qualsiasi altro software che ha bisogno di comprenderli e di usarli, creando uno scenario in cui l’intervento umano è necessario una sola volta in fase di configurazione e non più in fase di comprensione.

4. Struttura e Progettazione

4.1 Introduzione alle Tecnologie

Semantic Detector è una Java Web Service Application modulare ed estendibile che preso in input un dataset, e opzionalmente un file di configurazione per esso, analizza i Record contenuti al suo interno e fornisce in output una dettagliata descrizione circa la struttura e il contenuto del dataset stesso. L'output si presenta come un file ben strutturato, disponibile in più formati, con tutte le informazioni circa l'analisi effettuata.
Vi sono molteplici canali messi a disposizione dal sistema per accedere alle sue funzionalità, ognuno dei quali contribuisce ad una più vasta gamma di modalità di utilizzo. Innanzi tutto il sistema implementa, tramite un proprio server embedded, un'interfaccia REST interrogabile da un qualsiasi client capace di effettuare richieste HTTP; inoltre è presente una classica interfaccia HTML dalla quale è possibile interagire con il sistema tramite un classico form di ricerca.

La comunicazione tra Client e Server può avvenire tramite differenti tipologie di richiesta:

- GET: "*Il* ***metodo Get*** *è utilizzato per il recupero di risorse e può essere utilizzato per inviare parametri tramite la struttura dell'Url."* *(*http://www.html.it/pag/18465/codici-di-rispostae-metodi/*)*

- POST: "*l **metodo Post** è utilizzato per effettuare azioni, non viene salvato nella cache del browser (è per questo per esempio che ,durante la navigazione di un sito web, tentando di tornare alla pagina precedente spesso riceviamo un messaggio di errore che ci comunica che la pagina è scaduta*)" *(*http://www.html.it/pag/18465/codici-di-rispostae-metodi/*)*

Abbiamo trattato solo Get e Post in quanto sono i metodi più diffusi. Un'applicazione che implementa opportunamente questi metodi e che segue determinate linee guida si avvicina a quella che viene definita un'applicazione **RESTful.**

Come già accennato, il linguaggio utilizzato nello sviluppo del sistema è il Java. Il motivo di tale scelta è la presenza in questo linguaggio di due caratteristiche molto importanti ed utili allo scopo:

- **Object Oriented:** il paradigma di programmazione Object Oriented consiste nell'organizzazione del progetto in Classi, ovvero in strutture dati che descrivono una determinata entità. Un Oggetto è il risultato dell'instanziazione di una Classe e può interagire con altri Oggetti per mezzo dei Metodi, ovvero 'funzioni' che operano in modo diverso a seconda delle necessità.

- **Indipendenza dalla piattaforma:** per indipendenza dalla piataforma si intende la predisposizione del sistema ad essere trasferito su un qualsiasi

Sistema Operativo che supporti il Java, risultando perfettamente funzionante indipendentemente dal sistema che lo sta ospitando.

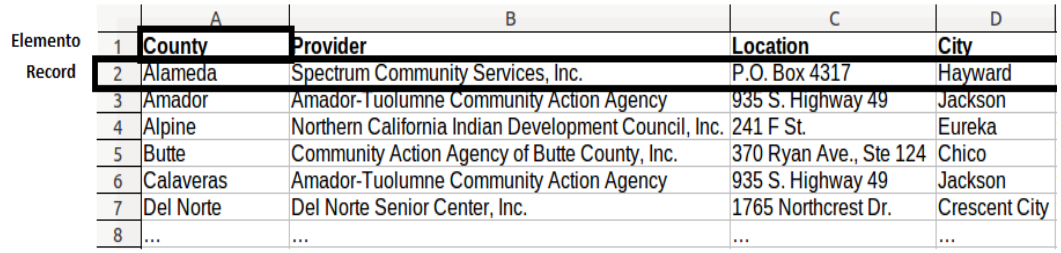
La struttura del progetto può essere al meglio descritta dai seguenti elementi:

1. Gestione dei Grafi (Alberi)
2. Interfaccia del Parser
3. Interfaccia dei Moduli
4. Il Core
5. Server Tomcat Embedded
6. Output

4.2 Gestione dei Grafi (Alberi)

Poichè i dataset presenti sul Web possono trovarsi in numerosi formati, tabellari e strutturati, si è cercato un modo per poter gestire sempre più formati senza modificare le logiche del sistema.

Innanzi tutto è necessario definire alcuni termini con i quali verranno identificate alcune entità a seguire; la Figura 3 mostra, su un generico dataset CSV, a cosa corrispondono le entità **Record**, **Elemento** e **Contenuto.**



	A	B	C	D
1	**County**	**Provider**	**Location**	**City**
2	Alameda	Spectrum Community Services, Inc.	P.O. Box 4317	Hayward
3	Amador	Amador-Tuolumne Community Action Agency	935 S. Highway 49	Jackson
4	Alpine	Northern California Indian Development Council, Inc.	241 F St.	Eureka
5	Butte	Community Action Agency of Butte County, Inc.	370 Ryan Ave., Ste 124	Chico
6	Calaveras	Amador-Tuolumne Community Action Agency	935 S. Highway 49	Jackson
7	Del Norte	Del Norte Senior Center, Inc.	1765 Northcrest Dr.	Crescent City
8	...	...	...	...

Figura 3: elemento e record CSV

L’Elemento è la singola cella della riga, il Record è l’insieme degli Elementi che compongono la riga, il Contenuto è il valore contenuto nella cella (quindi l’informazione da analizzare).

Nel caso di un dataset strutturato, prendiamo come esempio un dataset XML (Figura 4), ci si trova davanti ad una struttura non tabellare e il sistema deve dinamicamente e ricorsivamente esplorare tutti i livelli di annidamento che trova per arrivare a capire quali siano gli Elementi e quali i Record.

```
<?xml version="1.0" encoding="iso-8859-8" standalone="yes" ?>
<CURRENCIES>
  <LAST_UPDATE>2004-07-29</LAST_UPDATE>
  <CURRENCY>
    <NAME>dollar</NAME>
    <UNIT>1</UNIT>
    <CURRENCYCODE>USD</CURRENCYCODE>
    <COUNTRY>USA</COUNTRY>
    <RATE>4.527</RATE>
    <CHANGE>0.044</CHANGE>
  </CURRENCY>
  <CURRENCY>
    <NAME>euro</NAME>
    <UNIT>1</UNIT>
    <CURRENCYCODE>EUR</CURRENCYCODE>
    <COUNTRY>European Monetary Union</COUNTRY>
    <RATE>5.4417</RATE>
    <CHANGE>-0.013</CHANGE>
  </CURRENCY>
</CURRENCIES>
```

Figura 4: estratto XML

Analizzando le due struttura risulta evidente che la formattazione tabellare è un caso particolare di quella strutturata, ovvero un datasets tabellare può essere visto come un dataset strutturato lineare senza livelli di annidamento superiori al primo. Pertanto è stato elaborato un metodo che funzionasse sui dataset strutturali ottenendo di conseguenza lo stesso funzionamento sui dataset tabellari: ogni Record all'interno del dataset viene trattato come un Grafo, nello specifico come un Albero, ed ogni Elemento come Nodo di quest'Albero.

Così facendo il parsing e generazione degli Alberi su un dataset vengono delegati a delle Classi specifiche (le **Interfacce** di cui parleremo più avanti) che come standard di sistema devono restituire il Record richiesto convertito in Albero, i cui Nodi corrispondono agli **Elementi** del record e prendono il nome di **Semantic Element.** A livello implementativo, il *Semantic Element* è una classe avente come variabili private i riferimenti ad altri *Semantic Element* a lui collegati e come metodi pubblici i *Setters* e *Getters* di tali variabili, riferimenti che una volta impostati opportunamente creano quella rete ad albero che chiamiamo **Albero Semantico**.

Vediamo concretamente il risultato ottenuto dalla conversione in Albero Semantico di un Record Tabellare e di un Record Strutturato.

Documento Strutturato XML

```
<Persone>
  <Persona>
    <Nome>Mario</Nome>
    <Cognome>Rossi</Cognome>
    <NatoA>Bologna</NatoA>
  <Persona>
  ...
</Persone>
```

In questo estratto identifichiamo immediatamente le entità Record ed Elemento:

- I Record corrispondono ai Tag principali <Persona> che si ripetono all'interno del dataset
- Gli Elementi di ogni Record corrispondono ai Tag figli <Nome>, <Cognome> e <NatoA>

Documento Tabellare CSV

Persone

Nome	Cognome	NatoA
Mario	Rossi	Bologna
Francesco	Bianchi	Milano

Come per l'estratto XML, identifichiamo immediatamente le entità Record ed Elemento:

- I Record corrispondono alle Righe
- Gli Elementi corrispondono alle colonne che compongono le Righe

Le differenze principali tra i due estratti sono:

- Radice: nel documento strutturato è sempre presente una Radice ovvero un elemento che fa da contenitore padre a tutti i Record. Nel documento tabellare tale Radice non è esplicitata.
- Annidamenti: nel documento strutturato è evidente la presenza di annidamenti che genera una struttura gerarchica Padre-Figlio ben precisa. Nel documento tabellare invece viene mantenuta la stessa struttura per ogni Record del dataset.

Come anticipato precedentemente, adesso è evidente la relazione presente tra le due tipologie di struttura: *quella tabellare può essere vista come una strutturata con un solo livello di annidamento.*

Di conseguenza in entrambi i casi è possibile generare un albero che rispecchia fedelmente i dati e la loro struttura (Figura 5).

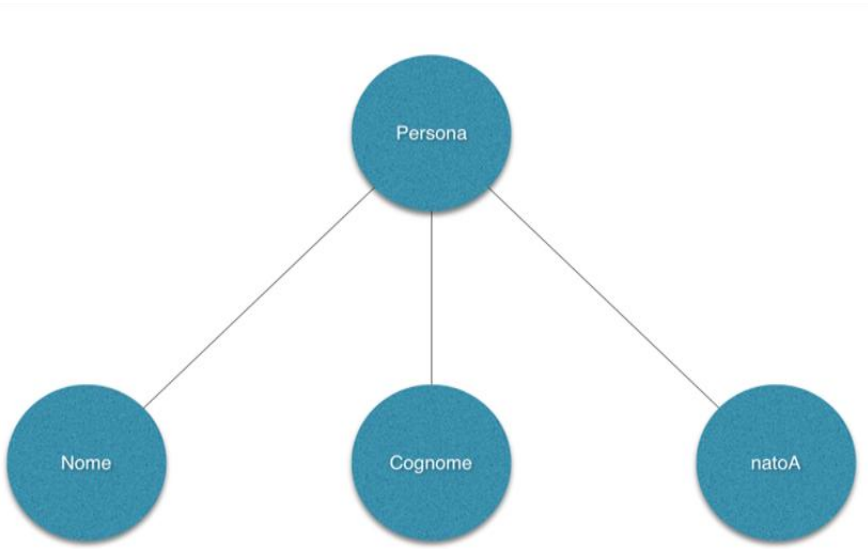


Figura 5: albero semantico

Il Padre, la Radice dell'albero, nel caso di un dataset tabellare non è esplicitato, quindi per ovviare a questa casistica viene creato un Nodo Radice chiamato ROOT da porre, in questo caso di esempio, al posto nel Nodo Persona (Nodo che nel caso del documento XML corrisponde al Tag principale che indentifica il Record). L'Albero Semantico ottenuto è l'unica struttura complessa accettata dal Core del sistema.

Come abbiamo visto gli Elementi di un dataset trattati come Nodi assumono nel sistema il nome di *Semantic Element*. Affinchè possa effettivamente essere analizzato, ogni Nodo deve contenere tutta una serie di informazioni circa la sua discendenza e circa il proprio contenuto; per rappresentarli è stata utilizzata una classe che assume la forma di una generica struttura dati con delle variabili private e dei metodi *Setter* e *Getter* per esse.

Vediamo nel dettaglio le proprietà principali di questi oggetti:

- **Type**: il Tipo identifica quello che è effettivamente il Nodo in questione, in particolar modo definisce se tal Nodo sia New, Standard o Xml-Attribute.
  New viene attribuito ad un Nodo appena creato, è quindi il valore di Default che viene assegnato all'oggetto appena instanziato.
  Standard viene attribuito a tutti quei Nodi che rappresentano un Elemento *normale* cioè che non ha particolari caratteristiche che lo contraddistinguono da un comune contenitore di dati (es. una classica cella CSV, un normale Tag XML con il suo contenuto).

Xml-Attribute viene infine attribuito a quei Nodi che rappresentato Elementi particolari del Record come ad esempio gli attributi XML, questo perchè nei dataset strutturati ogni Elemento può avere uno o più attributi contenenti informazioni possibilmente importanti che lo riguardano ed è pertanto necessario che anch'essi vengano processati.

- **Label:** la Label identifica la *targhetta* di ogni Elemento, ovvero corrisponde al nome del campo del dataset che si sta analizzando. Nel caso dei datasets tabellari, la Label di ogni cella equivale al nome della colonna alla quale essa appartiene; nel caso dei datasets strutturali corrisponde alla chiave associata ai valori degli Elementi, identificandosi quindi nei Tags nel caso di dataset XML e analogamente nelle vaire Key contenute nei datasets Json (composti da associazioni Chiave:Valore).

- **Content:** il Contenuto di un Nodo, preso in esame un Record ed un Elemento, è intuitivamente il Valore associato all'Elemento stesso ed è ciò a cui il sistema tenta di attribuire un valore Semantico significativo che risponda alla domanda "*che cos'è? Di che tipo di dato stiamo parlando?*".

- **Parent, Children, Attribute Children:** queste tre variabili servono a mantenere i riferimenti ad altri *Semantic Element* in modo da creare quella struttura gerarchica tipica dell'Albero Semantico.

- **Meaning Set:** strutturalmente il Meaning Set di un Elemento è una

HashMap atta a conservare progressivamente i possibili significati Semantici attribuiti dal sistema a tale Elemento. Le chiavi contengono una Stringa rappresentativa del significato (es. Integer, Address, Coordinate ecc.) mentre i valori associati ad ogni chiave sono dei contatori che tengono traccia del numero di occorrenze di ogni signicato Semantico attribuito all'Elemento.

4.2.1 Tree Utils

TreeUtils è la classe che implementa i meccanismi necessari ad operare correttamente sugli Alberi Semantici. In quanto tali, essi possono essere navigati ed esplorati utilizzando opportunamente tutti gli algoritmi noti che si desidera, intendendo con questo che possono essere riprodotte *le meccaniche e le strategie* di tali algoritmi. In particolare gli algoritmi utilizzati per l'esplorazione degli Alberi Semantici sono l'Algoritmo DFS e l'Algoritmo BFS, rispettivamente **Depth First Search** e **Breadth First Search.** Entrambi gli algoritmi possono essere utilizzati per esplorare un grafo (nel nostro caso un Albero) senza sapere a priori quanto esso sia grande e da quali Nodi sia composto, differenziandosi tra loro principalmente per la strategia adottata.
L'algoritmo più importante presente in questa Classe è il *combineTree*: questo algoritmo prende in input due Alberi Semantici, li esplora utilizzando l'algoritmo BFS e li combina in un unico Albero unendo le informazioni parziali che entrambi contengono. Questo deriva dal fatto che, come spiegato nei paragrafi successivi, l'analisi procede per record ognuno dei quali verrà convertito in un Albero con

struttura e informazioni sui contenuti riguardanti quel preciso record. Piuttosto che tenere in memoria un Albero Semantico per ogni record, ho preferito tenere sempre e solo una versione che si aggiorna al procedere dell'analisi.

4.3 Interfaccia del Parser e dei Moduli

Il Parser, nel contesto del sistema, è una Classe responsabile della lettura del dataset e della covnersione dei suoi record in Alberi Semantici. Poichè uno degli obbietti era la possibilità di leggere sempre più formati, è stata creata un'interfaccia Java che definisce la struttura che ogni nuovo parser deve avere per poter essere utilizzato dal sistema.
Questo vuol dire che per espandere le capacità di lettura del Semantic Detector, uno sviluppatore dovrà creare una nuova Classe Java che implementa l'interfaccia del Parser insieme ad alcuni metodi che utilizzerà il sistema ed inserire tale Classe all'interno del progetto. Il sistema da quel momento in poi sarà in grado di operare anche su archivi dati di quella tipologia.
Un Modulo, nel contesto del sistema, è una Classe utilizzata per effettuare una specifica tipologia di valutazione semantica sui dati. Come per il parser, il Modulo è un'interfaccia Java avente un solo metodo da dover implementare: il metodo *parse*.
Questo è l'unico metodo chiamato dal sistema per ottenere una risposta circa la semantica di un determinato input ma è assolutamente consentita l'implementazione di metodi aggiuntivi che ogni sviluppatore può ritenere

necessari per gli scopi del proprio modulo.
Questa interfaccia rappresenta il secondo grande punto di espansione del sistema in quanto consente sia di creare dal più semplice al più complesso meccanismo di riconoscimento sia in quanto consente un utilizzo personalizzato di tale meccanismo sfruttando, come spiegato meglio nel paragrafo 4.8, un file di configurazione utente. In altre parole uno sviluppatore ha la possibilità di creare nuovi moduli, implementandone l'interfaccia di sistema ed il relativo metodo *parse*, sia *embedded* , ovvero che non richiedono ne prevedono personalizzazioni utente ulteriori, sia configurabili, ovvero ne scrive il meccanismo ma lascia all'utente la possibilità di modificare dinamicamente i settaggi interni delle variabili del modulo con una o anche molteplici configurazioni differenti dello stesso modulo.
Per esempio un modulo configurabile che ho sviluppato è il modulo basato sulle Regular Expression: il metodo *parse* di questo modulo accetta come configurazione utente una o più espressioni regolari e le testa sull'input per verificarne il match.

4.4 Server Tomcat Embedded

Al fine di garantire la massima portabilità del sistema è stata scelta la versione Embedded di Apache Tomcat, lasciando quindi al sistema stesso il compito di configurare e rendere disponibili all'esterno tutti quei canali di comucazione HTTP che normalmente verebbero messi a disposizione dal server web (es. Apache) installato sulla macchina host. Con questa versione di Tomcat il sistema

può immediatamente risultare operativo in qualsiasi host che supporti Java 1.8 anche in assenza di server web installato.

Per l'interfaccia REST è stato utilizzato **Jersey**, "*un framework Open Source che implementa le specifiche JAX-RS, Java API for RESTful Web Services, utilizzando le annotazioni per mappare una classe Java ad una risorsa Web"* [html.it].
Con Jersey vengono gestite le Routes (le rotte, i percorsi) alle risorse e le azioni da effettuare in seguito ad una determinata richiesta, in particolare:

- in seguito ad una richiesta GET ad uno specifico URL, Jersey restituirà in output un file HTML contenente un Form. Tale Form accetta in input l'URL del dataset che si vuole analizzare e manderà questa informazione al Core del sistema per far partire effettivamente l'analisi, al termine della quale il sitema genererà l'output contenente le informazioni estratte.

- In seguito ad una richiesta POST ad uno specifico URL, specificando come parametri l'URL del dataset che si vuole analizzare e (opzionalmente) l'URL del proprio file di configurazione, Jersey contatterà direttamente il sistema per far partire l'analisi e generare l'output.

4.5 Il Core

Il Core, grazie alla modularità interna del sistema, si occupa di coordinare tutto il processo di analisi a cui è sottoposto il dataset preso in input. Nella Classe

principale non sono presenti meccanismi complessi bensì vengono sfruttate le singole funzionalità offerte dai vari componenti dell'applicazione.
Il processo di analisi completo si può adesso osservare nella sua interezza:

1. Il Web Server Tomcat riceve un dataset da analizzare e notifica il Core
2. Il Core rileva il tipo di dataset (CSV, XML ecc.) e instanzia, se presente, la Classe responsabile di quel formato che implementa l'interfaccia del Parser. Da ora in poi al Core non interessa di che formato si stia occupando in quanto il parser specializzato comunicherà con esso solo tramite Alberi Semantici.
3. Il Core instanzia tutte le Classi che implementato l'interfaccia del Modulo e le configura in base alle specifiche del file di configurazione. Da ora in poi il Core si preoccupera, per ogni input da processare semanticamente, di chiamare il metodo *parse* di ogni Modulo instanziato disinteressandosi di cosa farà e interessandosi solo del risultato derivato dalla sua valutazione.
4. Il Core richiede un record al Parser sotto forma di Albero Semantico.
5. Il Core naviga l'Albero sfruttando l'algoritmo BFS, lancia su ogni Nodo (in particolare sul contenuto del Nodo) il metodo *parse* di tutti i moduli e salva all'interno del Nodo stesso le risposte ottenute.
6. Il Core salva l'Albero appena ottenuto oppure se già presente in memoria un Albero derivante da record analizzati precedentemente (quindi se non sta analizzando per la prima volta un record del dataset in esame), combina i due Alberi arricchendo i risultati che progressivamente si ottengono esaminando sempre più record.
7. Il Core ripete i punti 4-5-6 finchè non ha esaminato la quantità di record

indicata nel file di configurazione.

8. Ad analisi finita il Core passa l'Albero defenitivo alla Classe responsabile della costruzione dell'output.

9. Il server Tomcat genera la risposta HTTP contenente l'output e la manda al Client.

5. Valutazione

Durante la fase finale della ricerca sono stati effettuati dei test per valutare come *Semantic Detector* migliora la qualità e la riusabilità degli *Open Data* riconoscendo effettivamente la tipologia dei dati.

Inizialmente sono stati selezionati 16 dataset significativi sia da portali di archivio dati italiani sia da portali americani. La tabella T1 mostra la lista completa dei dataset processati dal sistema ai fini della valutazione. Ogni riga della tabella si riferisce ad un dataset e ne riporta l'URI dal quale potervi accedere.

Topic	**Source URL**	Size (KBs)	Format
Doctors	http://gabriele.cigna.web.cs.unibo.it/dati/dottori.xml	70	XML
Nursery schools	http://gabriele.cigna.web.cs.unibo.it/dati/asili.csv	24	CSV
Dr. Who universe of doctors, actors, etc.	http://gabriele.cigna.web.cs.unibo.it/dati/drwho.csv	24	CSV
Drugs	http://gabriele.cigna.web.cs.unibo.it/dati/farmaci.xml	327	XML
Pharmacies	http://gabriele.cigna.web.cs.unibo.it/dati/farmacie.xml	127	XML
Employees of Emilia Romagna (italian administrative region)	http://gabriele.cigna.web.cs.unibo.it/dati/people.xml	45	XML
Plants	http://gabriele.cigna.web.cs.unibo.it/dati/plants.xml	9	XML
Music	http://gabriele.cigna.web.cs.unibo.it/dati/songs.xml	32	XML
Musical bands of Rome	http://85.18.173.117/opendata/ElencoBandeMusicali.csv	12	CSV
Libraries of Rome	http://85.18.173.117/mappe/Biblioteche.csv	20	CSV
Absestos licensed professionals of Illinois	https://data.illinois.gov/api/views/2gzr-9awy/rows.csv	230	CSV
Absestos licensed contractors of Illinois	https://data.illinois.gov/api/views/5vh3-wnad/rows.csv	18	CSV
Child health plus program enrollment of New York State	https://health.data.ny.gov/api/views/izdx-gtc9/rows.csv	8041	CSV
Contact information of the Dept. of Rehabilitation office of California	https://chhs.data.ca.gov/api/views/exxu-vffk/rows.csv	8	CSV
Service providers of California	https://chhs.data.ca.gov/api/views/jwfv-ersg/rows.csv	444	CSV
Public health services of the City of Chicago	https://data.cityofchicago.org/api/views/cjg8-dbka/rows.csv	19	CSV

Tabella T1: dataset selezionati per i test

Questi dataset sono stati selezionati secondo le seguenti logiche:

- una bilanciata quantità tra dataset di formato diverso (CSV e XML)
- varietà linguistica dei contenuti (non ci si è limitati a dataset italiani)
- differenti entità interessanti da riconoscere

Successivamente, per ogni elemento interessante die dataset (ad esempio per ogni colonna di una archivio CSV) è stata costruita la tabella T2 (di cui se ne riporta un estratto) contenente la corretta tipologia dei dati che il sistema dovrebbe essere in grado di riconoscere.

URL	FieldsLabel::Expected	
http://gabriele.cigna.web.cs.unibo.it/dati/dottori.xml	cognome::DBPedia_Person	nome::DBPedia_Person
http://gabriele.cigna.web.cs.unibo.it/dati/asili.csv	Address::TODO	Lat::Coordinate(Latitude)
http://gabriele.cigna.web.cs.unibo.it/dati/drwho.csv	Writer::DBPedia_Person	Director::DBPedia_Person
http://gabriele.cigna.web.cs.unibo.it/dati/farmaci.xml	href::UriPath	
http://gabriele.cigna.web.cs.unibo.it/dati/farmacie.xml	lat::Coordinate(Latitude)	long::Coordinate(Longitude)
http://gabriele.cigna.web.cs.unibo.it/dati/people.xml	cognome::DBPedia_Person	nome::DBPedia_Person
http://gabriele.cigna.web.cs.unibo.it/dati/plants.xml	availability::Date	
http://gabriele.cigna.web.cs.unibo.it/dati/songs.xml	Datereleased::Date	Datecertifiedplatinum::Date
http://85.18.173.117/opendata/ElencoBandeMusicali.csv	Indirizzo::TODO	

Tabella T2: settaggio delle entità presenti nei dataset

La Tabella T3 mostra come il sistema è stato configurato per effettuare i test utilizzando opportunamente il file di configurazione utente. Ad ogni tipologia è associato un metodo di riconoscimento e la soglia minima di accettazzione (compresa tra 0 e 1 ed impostata opportunamente per ottenere il giusto

compromesso tra qualità e tempo di esecuzione).

Data type	Threshold	Matching criterion
Email Address	0.1	RegEx
Tax Code	0.1	RegEx
Blood group	0.3	Matching wrt. to fixed set of elements
Gender	0.3	Matching wrt. to fixed set of elements
Location	0.1	Exact match on DBpedia labels of places
Person	0.1	Contain-based match on DBpedia labels of people
Number	0.1	RegEx
Geographic coordinates	0.1	RegEx
Date time	0.1	RegEx

Tabella T3: tipologia, soglia e metodo di matching per le entità

I meccanismi di riconoscimento utilizzati sono:

- **RegEx**: matching per mezzo di espressioni regolari
- **Set of elements**: matching basato sull'appartenenza ad una lista prefissata di elementi
- **DBPedia**: matching esatto o parziale con delle entità note riconosciute all'interno di archivi di Linked Data, in questo caso DBPedia.

A questo punto il sistema è stato utilizzato per analizzare i dataset presenti nella Tabella T1 aumentando progressivamente per ogni dataset la quantità di dati da analizzare rispetto alla totalità di quelli contenuti ed è stato confrontato l'output ottenuto con i risultati attesi in termini di Precision e di Recall.

Rispettivamente queste variabili indicano "*la probabilità che un documento recuperato (selezionato casualmente) sia attinente*" e "*la probabilità che un documento attinente (selezionato casualmente) sia recuperato in una ricerca*" [https://it.wikipedia.org/wiki/Precisione_e_recupero].
In questo caso la *precision* indica la probabilità che una determinata classificazione sia esatta e il *recall* la possibilità che tale classificazione venga assegnata alle entità corrette.

Per il calcolo di *precision* e *recall* (tenuta traccia dopo ogni test di veri positivi, veri negativi, falsi positivi e falsi negativi) sono state utilizzare le seguenti formule:

$$\text{Precisione} = \frac{|\{\text{documenti attinenti}\} \cap \{\text{documenti recuperati}\}|}{|\{\text{documenti recuperati}\}|}$$

$$\text{Recupero} = \frac{|\{\text{documenti attinenti}\} \cap \{\text{documenti recuperati}\}|}{|\{\text{documenti attinenti}\}|}$$

Con questa configurazione generale, le analisi dei dataset sono state effettuate su porzioni di essi sempre crescenti sia per misurare la precisione del sistema sia per identificare la quantità più ragionevole sotto o oltre la quale non è necessario andare. Nello specifico sono stati analizzati il 3%, 5%, 10% e 20% dei dati, ottenendo i risultati riportati nelle Tabelle T4, T5, T6 e T7, nelle quali TP indica i *True Positives*, FN i *False Negatives*, e FP i *False Positives.*

Type	Precision	Recall	TP	FN	FP
DBPedia_Person	0,64	0,70	7	3	4
DBPedia_Location	0,77	0,77	10	3	3
TaxCode	1,00	1,00	2	0	0
Coordinate(Latitude)	0,60	1,00	3	0	2
Coordinate(Longitude)	0,50	1,00	3	0	3
UriPath	1,00	1,00	4	0	0
BloodGroup	1,00	1,00	1	0	0
Gender	1,00	1,00	1	0	0
Date	1,00	0,83	5	1	0
EmailAddress	1,00	0,50	1	1	0

Tabella T4: test sul 3% del dataset

Type	Precision	Recall	TP	FN	FP
DBPedia_Person	0,55	0,60	6	4	5
DBPedia_Location	0,85	0,79	11	3	2
TaxCode	1,00	1,00	2	0	0
Coordinate(Latitude)	0,60	1,00	3	0	2
Coordinate(Longitude)	0,50	1,00	3	0	3
UriPath	1,00	1,00	4	0	0
BloodGroup	1,00	1,00	1	0	0
Gender	1,00	1,00	1	0	0
Date	1,00	0,83	5	1	0
EmailAddress	1,00	0,50	1	1	0

Tabella T5: test sul 5% del dataset

Type	Precision	Recall	TP	FN	FP
DBPedia_Person	0,58	0,78	7	2	5
DBPedia_Location	0,79	0,79	11	3	3
TaxCode	1,00	1,00	2	0	0
Coordinate(Latitude)	0,60	1,00	3	0	2
Coordinate(Longitude)	0,50	1,00	3	0	3
UriPath	1,00	1,00	4	0	0
BloodGroup	1,00	1,00	1	0	0
Gender	1,00	1,00	1	0	0
Date	1,00	0,83	5	1	0
EmailAddress	1,00	0,50	1	1	0

Tabella T6: test sul 10% del dataset

Type	Precision	Recall	TP	FN	FP
DBPedia_Person	0,57	0,89	8	1	6
DBPedia_Location	0,69	0,85	11	2	5
TaxCode	1,00	1,00	2	0	0
Coordinate(Latitude)	0,60	1,00	3	0	2
Coordinate(Longitude)	0,50	1,00	3	0	3
UriPath	1,00	1,00	4	0	0
BloodGroup	1,00	1,00	1	0	0
Gender	1,00	1,00	1	0	0
Date	1,00	0,83	5	1	0
EmailAddress	1,00	0,50	1	1	0

Tabella T7: test sul 20% del dataset

Osservando in particolare i valori relativi all'utilizzo di DBPedia, risultano evidenti tre aspetti molto importanti:

1. Aumentare la porzione di dati da analizzare non sempre porta a migliori risultati in termini di precisione. I test hanno provato che in media con questo sistema basta analizzare il 5-10% di tutto il dataset per ottenere dei risultati significativi.

2. *Precision* e *Recall* raggiungono dei valori importanti, circa 0.6 - 0.7 - 0.8 su 1, in termini di affidabilità dei risultati.

3. Le regole più statiche e sintattiche (come URI, email, date ecc.) se scritte opportunamente portano ad una precisione estremamente elevata, prossima a 1, da cui ne consegue un'elevata affidabilità.

A conferma delle affermazioni precedenti, il grafico G1 mostra i valori finali della precisione relativa alle tipologie riportate in Tabella T3 ottenuti, al variare della porzione di dataset analizzata, al termine dei test.

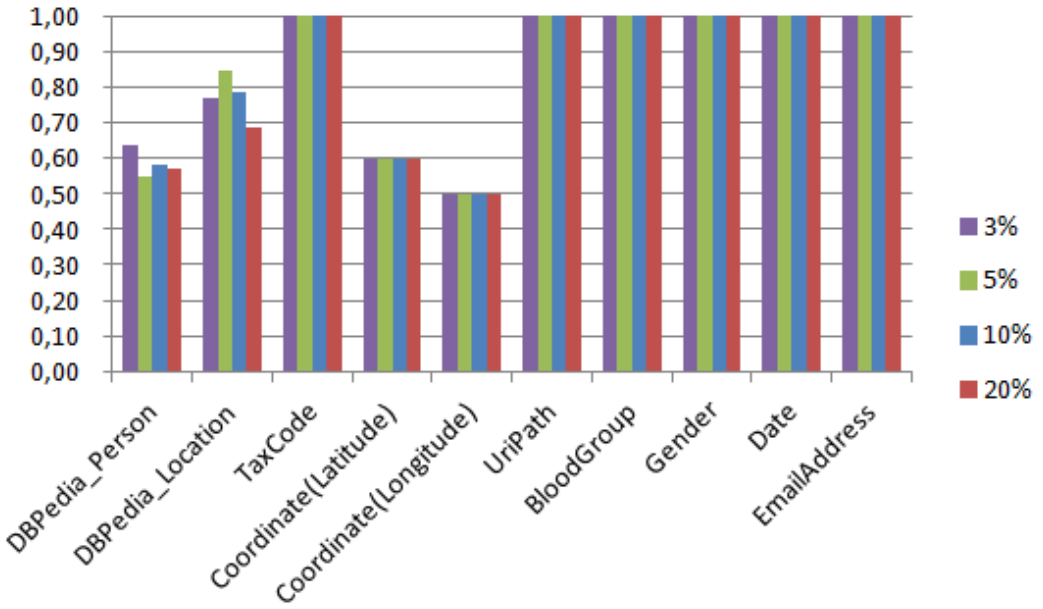


Grafico G1: precisione al variare della porzione di dataset analizzata

Questi test sono stati effettuati utilizzando delle meccaniche e regole (riutilizzabili tutte, in parte o nessuna) di riconoscimento (impostate nel file JSON di configurazione utente) per entità comuni come i luoghi, indirizzi email, date, persone e coordinate ottenendo dei risultati in termini di precisione ed acuratezza molto affidabili. Le potenzialità del sistema però non dipendono solo dalle regole che vengono impostate o dalla loro quantità bensì dall'espansione del sistema stesso: *Semantic Detector* è stato studiato e progettato affinchè a maggiori contributi nella creazione di nuovi meccanismi di riconoscimento corrispondano maggiori capacità semantiche.

6. Conclusioni

Con lo sviluppo di *Semantic Detector* questo lavoro di Tesi ha voluto dare un contributo ad un contesto in cui gli *Open Data* disponibili in rete presentano ancora oggi numerosi difetti. La presenza di tali difetti ha spesso limitato la possibilità di sfruttare al meglio le informazioni contenute in questi archivi, in particolare è spesso necessario un intervento manuale per comprenderne il contenuto e renderli processabili da un software cliente. I contributi dati dai software trattati nel paragrafo 2.4 non riescono ad eliminare questo problema poichè svolgono esclusivamente alcune mansioni di conversione e rifinitura richiedendo inoltre delle interazioni utente. Nonostante *Tableau* (presentato nel paragrafo 2.5, offra un servizio di analisi e manipolazione dei dati molto importante, non sembra configurarsi come una soluzione al problema in quanto limita ogni personalizzazione sia in fase di pre-analisi, impedendo di ampliare e personalizzare i riconoscimenti di entità, sia in fase di post-analisi, limitando le modalità di riutilizzo dei risultati.

Semantic Detector è stato studiato per essere più incisivo laddove altri software non sono riusciti ad esserlo, focalizzando lo sviluppo su estendibilità, personalizzazione e longevità. (esteso), il sistema non solo potrà processare un numero sempre maggiore di formati di archivi dati, ma potrà anche offrire un numero sempre maggiore di meccaniche per i riconoscimenti sintattici e, sfruttando al meglio DBPedia, effettuare valutazioni sempre più complesse circa la semantica dei dati. Potendo utilizzare un file di configurazione esterno, *Semantic Detector* offre inoltre la possibilità di predisporre una o più tipologie di

analisi per molteplici scopi, da poter succesivamente effettuare in modo automatico. Ciò permette di ottenere in risposta un *output* processabile altrettanto automaticamente da un software cliente ad alto livello. Dall'altra parte, un problema da non sottovalutare in futuro sarà la difficoltà che si è manifestata analizzando *Open Data* caratterizzati da una struttura interna non regionevole, l'analisi dei quali conduce a risultati scorretti nell'identificazione automatica di tale struttura. Un'importante miglioria deve pertanto essere l'ottimizzazione nei *Parser*, i responsabili di questa identificazione, delle meccaniche di riconoscimento strutturale per i *dataset* più imprevedibili.

I risultati evidenziati nel capitolo 5, in cui il sistema è stato testato su *Open Data* reali e con configurazioni sensate, provano che con *Semantic Detector* si riesce effettivamente ad ottenere una tipizzazione dei dati molto affidabile (con precisioni intorno al 70%) analizzando solo il 5 - 10% del contenuto dei *dataset*. Inoltre, potendo sfruttare DBPedia per riconoscere le entità e per classificarle utilizzando sintassi semanticamente note, *Semantic Detector* permette di predisporre i dati per un loro riutilizzo immediato o per una successiva conversione in *Linked Open Data,* escludendo gradualmente, se inserito in una catena di processi software, l'intervendo manuale.

7. Bibliografia

[BHAR07] Bizer, C., Heath, T., Ayers, D., Raimond, Y., *Interlinking Open Data on the Web*, in The 4th European Semantic Web Conference, 3-7th June 2007 in the Tyrol region of Innsbruck, Austria.

[BHT09] Bizer C. , Heath T. and Berners-Lee Tim, *Linked Data - The Story So Far*, in "The International Journal on Semantic Web and Information Systems", pp. 1-22, vol. 5, issue 3, 2009.

[AB04] G. Aichholzer, H. Burkert, *Public sector information in the digital age: between markets, public management and citizens' rights*. Edward Elgar Publishing, 2004.

[PB08] B. Ponti, *Il regime dei dati pubblici. Esperienze europee e ordinamento nazionale*, In Maggioli Editore, Sant'Arcangelo di Romagna, 2008.

[BC06] C. Bizer and R. Cyganiak. *D2R server - publishing relational databases on the semantic web*. Poster at the 5th International Semantic Web Conference. Vol. 175. Athens, GA, USA 2006 .

[HFHPRW09] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. *The WEKA data mining software: an update.* ACM SIGKDD explorations newsletter, 11(1):10–18, 2009.

[RBP15] P. Ristoski, C. Bizer, and H. Paulheim. *Mining the Web of Linked Data with Rapid-Miner*. Journal of Web Semantics, 35:142–151, 2015.

[KPHH11] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. *Wrangler: Interactive visual specification of data transformation scripts*. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pages 3363–3372. ACM, Vancouver BC 2011.

[VM12] M. Verlic. *Lodgrefine-lod-enabled google refine in action*. In I-SEMANTICS (Posters & Demos), pages 31–37, 2012.

[ABKLCI07] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, Zachary Ives - *DBpedia: A Nucleus for a Web of Open Data*, pp 722-735, book The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007.

[CSH03] C Chabot, C Stolte, P Hanrahan - *Tableau Software*, 2003 - medlibrary.org

[PA11] Peled, A. *When transparency and collaboration collide: The USA Open Data program*. Journal of the American society for information science and technology, 2011, 62.11:2085-2094