Caccia al tweet: un approccio per la geolocalizzazione di un utente sulla base dei suoi tweet

Valerio Gregori
Email: val.gregori2@stud.uniroma3.it

Mattia Iodice
Email: mat.iodice1@stud.uniroma3.it

Alessandro Oddi
Email: ale.oddi1@stud.uniroma3.it

Nella relazione viene presentata l'implementazione di un framework per la geolocalizzazione di un utente a partire dal corpus dei suoi tweet. L'obiettivo prefissato era appunto l'implementazione del processo di geolocalizzazione, basato sull'articolo You are where you tweet di Cheng, Caverlee e Lee, prendendo spunto dal lavoro dello studente F. Tanzi. Il documento é strutturato in sezioni, ciascuna focalizzata su un aspetto diverso del lavoro svolto. In particolare, dopo una breve descrizione delle caratteristiche fondamentali degli strumenti utilizzati, vengono descritti gli approcci seguiti nella fase di implementazione, insieme alla giustificazione delle scelte architetturali e alle motivazioni che hanno spinto alla reimplementazione totale del tool. Nella sezione finale vengono quindi presentati i risultati ottenuti, fornendo un confronto diretto con le metriche riportate nell'articolo precedentemente citato.

1 Dataset di riferimento

La prima parte del lavoro svolto ha riguardato lo studio dell'articolo *You are where you tweet* insieme all'analisi dei dati in esso utilizzati. Vengono quindi presentati i punti salienti emersi dall'indagine.
I dati presi in analisi nello studio condotto fanno riferimento a un dataset relativo a un insieme di tweet estrapolati e suddivisi in training set e test set. In particolare, le caratterisitche fondamentali sono le seguenti:

1. il processo di estrazione dei tweet é avvenuto tra il Settembre del 2009 al Gennaio del 2010
2. il training set contiene $115,886$ utenti di Twitter e $3,844,612$ aggiornamenti da parte degli utenti stessi.
3. ciascuna localitá degli utenti é automaticamente etichettata negli USA con livello di dettaglio relativo alla cittá.
4. il test set contiene $5,136$ utenti di Twitter e $5,156,047$ tweets
5. tutte le localizzazioni degli utenti sono ottenute dalla posizione dei loro telefoni e sono espresse nella forma *latitudine,longitudine*.

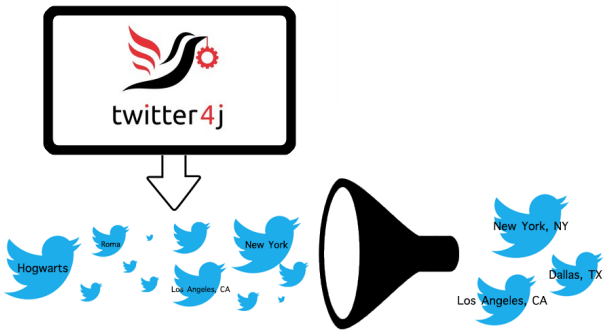

Fig. 1. Architettura del modulo per il retrieve di tweet dallo stream. Vengono scartate citt non USA e vengono associati agli stati alle citt non correttamente formattate.

Considerate le dimensioni piuttosto contenute dei dati in questione, in un primo momento abbiamo definito un modulo per arricchire (locupletare) il dataset. La scelta é stata guidata inoltre dall'esigenza di rispondere in modo efficiente ed efficace all'evoluzione della lingua. Infatti, le formule di espressione linguistiche, in contesti estremamente dinamici come quelli dei social network, sono soggette a continui cambiamenti: i.e. slang, neologismi, trand sociali ecc.

Il modulo implementato é essenzialmente un filtro che sfrutta lo stream offerto da **Twitter4j** effettuando una cernita tra tweet di interesse e non.
Per tweet di interesse si fa riferimento alla classe di aggiornamenti provenienti da utenti *ben geolocalizzati*: l'attributo *position* dell'utente deve comparire nella forma *cittá , stato* e la coppia in questione deve essere collocabile all'interno del territorio statunitense. Inoltre per far fronte all'eccessivo numero di utenti che dichiarano la cittá senza fornire lo Stato di appartenenza si é deciso di seguire un approccio gazetteer per l'individuazione dello Stato.
Considerando l'ingente tasso di omonimia fra i nomi delle cittá americane, data una cittá senzo uno stato, si associa ad essa lo Stato relativo alla cittá piú popolosa con quel nome.
Mediante l'utilizzo di questo approccio vengono scartati fino

all' 80% di tweet che non sono geotaggati.

2 Problematiche iniziali

L'obiettivo definito all'inizio del progetto era improntato all' *intention mining* degli utenti, diversificandoli sulla base della loro localizzazione.
Per il task di localizzazione si voleva utilizzare il tool realizzato dallo studente Tanzi e quindi si é proceduto con un'attenta analisi del codice e delle rispettive funzionalitá. Giá dalla prima esecuzione non é stato possibile ottenere l'output sperato.
In particolar modo, é emerso che il dataset in input prensentava tweet malformattati che non rispettavano la struttura necessaria per l'esecuzione del tool.
Escludendo i tweet in questione e testando il tool su un sottoinsieme del dataset iniziale é stato quindi possibile effettuare una stima del tempo d'esecuzione del processo di *parsing* sull'intero dataset. Il tempo richiesto stimato ammontava a circa una settimana a causa dell'utilizzo di strumenti non strettamente necessari, come spiegato nella Sezione successiva.
Risolti quindi i principali problemi legati ai tempi di computazione, si é proceduto con l'esecuzione del processo di geolocalizzazione. Anche qui, purtroppo, i costi d'esecuzione non sono stati soddisfacenti: l'implementazione aveva un costo pari a $O(n^3)$, con tempi di esecuzione stimati di circa 6 giorni.
Alla luce delle problematiche riscontrate si é quindi deciso di rinunciare al task legato all'intention mining degli utenti e piuttosto di focalizzarsi su un'implementazione funzionante e performante del processo di geolocalizzazione.

3 Descrizione dei flussi di esecuzione

Con l'intento di ridefinire l'intero processo di geolocalizzazione, sono stati definiti tre macro-procecessi:

1. Parsing
2. Calcolo del Tf-Idf
3. Validazione

Nel paper *You Are Where You Tweet* il fulcro del processo di geolocalizzazione dell'utente é l'utilizzo di un indice **tf-idf** sulle parole utilizzate all'interno dei tweet analizzati. Un requisito fondamentale per la costruzione di un tf-idf di qualitá risiede nella qualitá del training set. A tal fine, é stato creato un package dedicato al raffinamento e alla pulizia dei dati.

Nell'utilizzare YAGO viene consigliato di usare il core dello stesso knowledge graph, perció il processo é implementato sui file descritti in precedenza, che rappresentano proprio il nucleo del grafo. Abbiamo comunque ritenuto opportuno analizzare i dati presenti e abbiamo estratto diverse informazioni.
Dal file yagoFacts.tsv emerge che i fatti presenti sono **5.625.089**, mentre da yagoTransitiveTypes.tsv sono invece state contate **5.129.147** entitá. Il dato che emerge é che si tratta di un grafo poco denso, in cui i nodi che hanno almeno un arco entrante o uno uscente sono **2.632.727**.
Un ulteriore aspetto interessante che abbiamo notato nell'analisi dei documenti messi a disposizione é che delle **100** relazioni attestate, solo **37** compaiono nei fatti presenti in *yagofacts*.

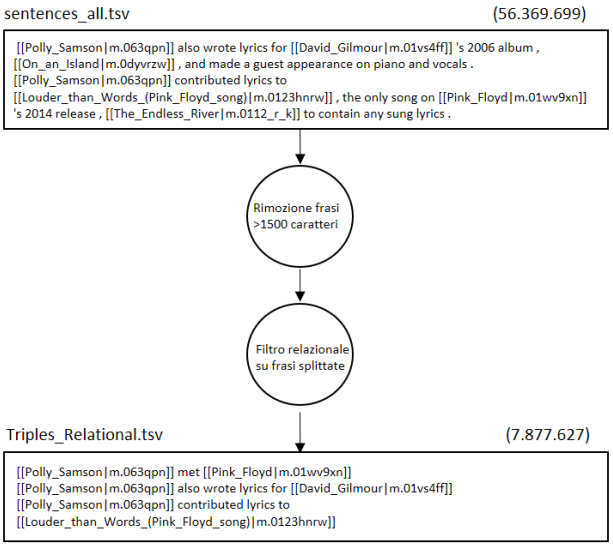

Fig. 2. Prefiltraggio del dump di Wikipedia

4 Implementazione di Lector

I dati riportati fanno riferimento all'esecuzione su un pc con le seguenti specifiche:

1. Intel Core i7-6700HQ
2. memoria fisica installata 8.00GB
3. SSD

4.1 Rimozione rumore e frasi non relazionali

Il processo di estrazione di fatti é stato implementato in Java, sebbene alcuni passaggi fondamentali siano stati sviluppati con l'ausilio di strumenti supplementari come MySQL e script bash.
Analizzando il file annotato *sentences_all.tsv* sono emerse alcune frasi particolari e problematiche per alcuni aspetti implementativi: si tratta di frasi spropositatamente lunghe e senza alcun significato, per lo piú costituite da sequenze di entitá. *Abbiamo quindi filtrato il file escludendo le frasi piú lunghe di 1500 caratteri*. Seguendo i passaggi fondamentali del processo seguito da Lector abbiamo poi trasformato ciascuna frase in una sequenza di triple, costituite da frasi racchiuse tra due entitá consecutive.
Ciascuna tripla é stata quindi data in pasto ad un filtro relazionale, con l'obiettivo di scartare tutte le frasi che non fossero relazionali, come mostrato in Figura 2. Il file ottenuto in output conteneva **7.631.515** triple distinte, e perció ne consegue che il filtro ha scartato circa **l'85%** delle triple. Di queste **7.631.515** abbiamo contato quelle con soggetto e oggetto entrambi presenti nelle entitá definite in yagoTypes: **5.480.597** triple. A questo punto abbiamo deciso di procedere attraverso quattro esecuzioni parallele:

1. la prima senza considerare né frasi lista né generalizzazione
2. la seconda effettuando la **generalizzazione** di alcune categorie di termini ricorrenti (lavori, numeri, stati, nazionalitá)
3. la terza effettuando la **generalizzazione** parallelamente ad un pesante processo di **stopping**
4. la quarta, piú che altro complementare alle precedenti, **normalizzando liste** di entitá

Tale scelta ci ha consentito di valutare singolarmente i quattro approcci descritti, permettendoci poi di trarre conclusioni sull'effettivo beneficio apportato.

4.2 Associazione di frasi e relazioni senza stopping e generalizzazione

Nella prima fase si é quindi direttamente proceduto con l'associazione di frasi alle relazioni. Dovendo confrontare soggetto e oggetto di 7 milioni di triple con soggetto e oggetto di 5 milioni di fatti, abbiamo intuito che una buona soluzione in termini di efficienza e soprattutto di flessibilitá potesse derivare dall'utilizzo di un DBMS. Infatti la logica SQL si presta particolarmente bene alle operazioni descritte nel paper: l'associazione di frasi alle relazioni non é altro che un join tra soggetto e oggetto di una tripla con soggetto e oggetto nei fatti presenti in YAGO. Abbiamo quindi innanzitutto definito la tabella *yagoFacts*(sub,relation,ob), nella quale abbiamo importato i fatti dal .tsv yagoFacts. A questo punto per ciascuna tripla presente é stato sufficiente eseguire una query del tipo:

$$SELECT * FROM\ yagoFacts\ WHERE\ sub=subjT\ AND\ ob=objT \tag{1}$$

In questa fase é stato interessante osservare che:

1. le query che hanno restituito almeno un match sono state **475.062**. Questo dato é risultato particolarmente rilevante perché, se sottratto al numero di triple che hanno soggetto e oggetto entrambi appartenenti alle entitá di YAGO[1], indica il massimo numero di fatti estraibili: poco piú di **5 milioni**.
2. ogni query restituisce un certo numero di relazioni, quelle appunto giá esistenti tra soggetto e oggetto all'interno di YAGO. É emerso che la somma delle relazioni restituite é **547.601**. Ció indica che, in media, pero ogni coppia soggetto-oggetto ci sono **1.15** fatti.
3. il numero di frasi univoche presenti in *relPhraseCount* [2] é circa **85.000**, mentre le tuple presenti sono **111.514**

L'esecuzione di questo join ha richiesto circa 120 minuti.

4.3 Misurazione della forza di un'associazione entitá-frase

La tabella *relPhraseCount* mette appunto in relazione ciascuna relazione con un certo numero di frasi e il numero

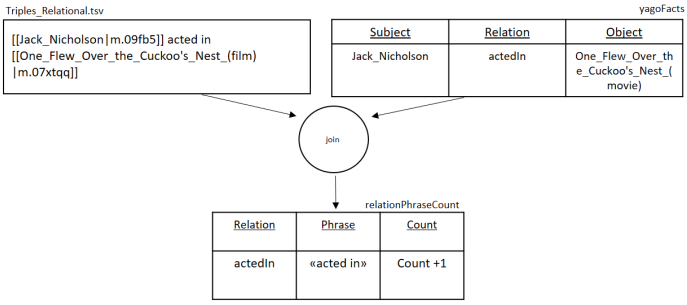

Fig. 3. Join tra le triple e i fatti

di occorrenze. Per misurare quindi la forza dell'associazione tra relazione e frase abbiamo riutilizzato la formula proposta in Lector:

$$P(r_i|p) = \frac{c(p,r_i)}{\sum_{j \in R} c(p,r_j)} \tag{2}$$

Abbiamo quindi definito la nuova tabella *relPhraseScore(rel,phrase,score,probability)*, in cui inserire, appunto i valori di probabilitá e score. Avendo giá a disposizione la tabella *relPhraseCount* é stato possibile riempire la nuova tabella andando ad inserire in probability il risultato della query:

$$\frac{SELECT\ \ countFROM\ \ relPhraseCount\ \ WHERErel = x\ \ AND\ \ phrase = y}{SELECT\ \ sum(count)\ \ FROM\ \ relPhraseCount\ \ WHEREphrase = y} \tag{3}$$

e nella colonna score:

$$score(p,r_i) = \log c(p,r_i) \cdot P(r_i|p). \tag{4}$$

Queste due operazioni sono risultate il collo di bottiglia dell'intero processo di estrazione di nuovi fatti. In particolare il tempo impiegato per popolare completamente la nuova tabella é stato di **4 h e 27 min**. Una considerazione su cui ci siamo soffermati é che si tratta peró di operazioni che si prestano abbastanza bene alla **scalabilitá orizzontale**.
É infatti sufficiente distribuire la tabella *relPhraseCount* su piú macchine e operare concorrentemente su frasi distinte, aggregando poi i risultati parziali. Questo approccio permetterebbe di scalare facilmente e abbattere i tempi di esecuzione.

4.4 Estrazione di nuovi fatti e validazione

Al fine di scegliere le K=20 frasi piú significative per ogni relazione é stato sufficiente eseguire la query che ci restituisse per ciascuna relazione le prime 20 frasi con probabilitá maggiore di 0.5, ordinate per score decrescente.
A quel punto abbiamo definito una mappa *HashMap <String>,<Set<String>> phraseToRelations* . Attraverso questa mappa é stato possibile scansionare il file iniziale, contenente le triple relazionali, per estrarre nuovi fatti: ogni volta che la frase contenuta nella tripla risulta contenuta

[1] 5.480.597
[2] Vedi 2.3 per definizione

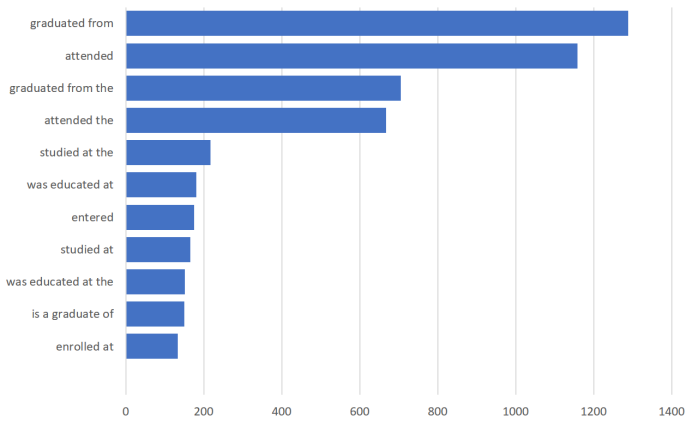

Fig. 4. Frasi pi ricorrenti per la relazione *graduatedFrom*

in *phraseToRelations* viene aggiunto un nuovo fatto tra soggetto e oggetto della frase, corrispondente alle relazioni associate alla frase stessa.
I vincoli che abbiamo specificato nell'aggiunta del fatto sono:

1. sia soggetto che oggetto devono essere entitá presenti in YAGO
2. non deve essere giá presente un fatto corrispondente alla relazione individuata tra le due entitá

Se per il secondo vincolo non c'é bisogno di particolari chiarimenti é bene spendere qualche riga per chiarire l'esigenza e le implicazioni del primo.
Al fine di validare i fatti estratti é infatti necessario che ciascuna entitá compaia nel file *yagoTypes*. Questo requisito é necessatio perché ciascuna relazione presente in YAGO lega un particolare tipo di soggetto ad un particolare tipo di oggetto. Vincoli che sono espressi in *yagoSchema*.
Proponendo un esempio concreto si ha che:
La relazione **actedIn** richiede un soggetto di tipo *actor* e un oggetto di tipo *movie*. Supponendo che sia stato estratto il fatto [[*Cristiano_Ronaldo...*]] *played in* [[*Real_Madrid..*]], associato alla relazione *actedIn*, la verifica dei tipi consentirebbe di scartare il fatto che, appunto, non rispetta il dominio della relazione. Ci siamo infatti accorti che in YAGO tutti i fatti presenti rispettano le imposizioni espresse in *yagoSchema* e perció la validazione dei fatti é un processo indispensabile prima di poter inserire nuovi fatti nel knowledge graph.
Nonostante questi vincoli abbiamo comunque provato ad analizzare le variazioni delle quantitá di fatti estratti e validati modificando i requisiti:

1. le entitá devono essere giá presenti in yagofacts
2. le entitá devono essere presenti in yagotypes
3. nessun vincolo sulle entitá

I risultati sono mostrati in Tabella 2 e come ci si poteva aspettare l'assenza di vincoli consente di estrarre un gran numero di fatti in piú.
In questa fase non c'é stato bisogno di caricare tutte le entitá in MySQL: é bastato leggere da tsv e memorizzare in un unico set le entitá univoche presenti (sia che provenissero dai

Table 1. Variazione del numero di fatti estratti al variare dei vincoli

Vincolo	# Fatti estratti	# Fatti validati
entitá in yagoFacts	340 K	190 K
entitá in yagoTypes	740 K	493 K
nessun vincolo	1.05 M	-

fatti giá presenti in YAGO sia che fossero stati presi da yagoType).
Per i tipi invece é stato necessario inserire i dati in MySQL, trattandosi di una quantitá di dati intorno ai 7 GB.
Abbiamo quindi definito la tabella *yagoTypes(entitá,tipo)*, e per ogni fatto estratto verificato che soggetto e oggetto avessero, tra i propri tipi, quelli richiesti dalla relazione in *yagoSchema*.
In un primo momento utilizzavamo un file *yagoTypes* semplificato, ma il contenuto non era per niente esaustivo, mancando spesso i tipi fondamentali delle entitá. Questo implicava un'analisi non banale dei tipi, dovendo andare a percorrerne l'albero delle dipendenze fino alla radice e costringendoci a fare assunzioni pericolose.
Abbiamo quindi analizzato i documenti disponibili, fino a trovare il file *yagoTransitiveTypes*, che garantisce la chiusura transitiva delle entitá rispetto ai tipi. In esso ciascuna entitá risulta associata a tutti i suoi supertipi, fino alla radice *owl:Thing*, ed é quindi stato sufficiente fare un confronto 1:1 tra ciascun tipo dell'entitá e il tipo richiesto dalla relazione.
In tutti e tre i casi il tempo di esecuzione per l'estrazione di fatti si é mantenuto al di sotto dei **18 minuti**, mentre quello per la validazione intorno ai **12**.

4.5 Generalizzazione e stopping

Dopo aver implementato questo primo processo di estrazione di nuovi fatti abbiamo ragionato sui modi di **generalizzazione delle frasi**, usando dei placeholder al posto di parole contenenti specifiche informazioni non essenziali per descrivere la relazione. Quindi, al di lá dei meccanismi proposti nel paper su Lector, abbiamo osservato i fatti estratti e cercato di individuare classi di termini in grado di generalizzare l'intero processo. Abbiamo quindi definito cinque dizionari:

1. nazionalitá, *NAT*
2. stati, *STA*
3. professioni, *JOB*
4. numeri , *NUM*
5. numeri cardinali, *CAR*

Un esempio di una frase generalizzata é il seguente:

input: *[[Sidney_Olcott ...]] married actress [[Vale...]]*
output: *[[Sidney_Olcott ...]] married JOB [[Vale...]]*

Abbiamo quindi ripetuto il processo di estrazione di nuovi fatti e di validazione basata su tipi. In Tabella 3 i

Table 2. Fatti estratti e fatti validati con il processo di generalizzazione e con il processo ibrido generalizzazione e stopping

Approccio	# Fatti estratti	# Fatti validati
Generalizzazione	777 K	499 K
Generalizzazione e stopping	2.19 M	985 K

risultati ottenuti.

Dopo aver rieseguito Lector con la generalizzazione delle frasi abbiamo voluto rieseguire l'intero processo introducendo anche un modulo che eseguisse lo **stopping** delle frasi, unitamente al processo di generalizzazione. Abbiamo fatto uso della libreria open source Exude. I risultati sono messi a confronto con quelli della generalizzazione in Tabella 3.
Abbiamo scelto di non ripetere dall'inizio il processo di calcolo della probabilitá e dello score, poiché il forte sospetto, confermato da diverse esecuzioni di testing, era che si rischiasse di introdurre troppo rumore. Essenzialmente quindi, dalla tabella *relPhraseScore* , costruita in precedenza sulle triple non processate in alcun modo viene effettuato il retrieve delle prime 30 frasi con probabilitá maggiore di 0.5, ordinate per score. A queste frasi abbiamo quindi applicato il modulo di generalizzazione (e stopping). Il risultato é stato che diverse frasi sono state aggregate in una sola, sia nel processo di generalizzazione che in quello di generalizzazione e stopping. Abbiamo poi trasformato ciascuna frase nelle triple applicando il medesimo processo di generalizzazione. La nostra ipotesi, testata su un campione di 150K triple, é che costruire i valori di score e probabilitá su frasi non processate assicuri una certa resistenza al rumore che altrimenti verrebbe introdotto.
In altre parole, in questo modo le frasi piú rappresentative per ciascuna relazione dovrebbero essere piú affidabili rispetto a quelle ottenute dal calcolo di score su frasi generalizzate o stoppate.

4.6 Normalizzazione di frasi lista

Ultimata la prima fase di implementazione di Lector abbiamo lavorato sul raffinamento del modulo realizzato nel precedente homework. Lo scopo del task era quello di migliorare l'estrazione di fatti da particolari tipologie di frasi, chiamate frasi lista.
Un esempio di questa tipologia di frasi é :

Obama is a supporter of Chicago White Sox, Chicago Bears, and Pittsburgh Steelers.

Da questo tipo ti frasi Lector ha un approccio abbastanza limitato: riesce ad operare solo in presenza di "*such as*", ricostruendo le frasi in modo parziale. Ad esempio, dalla frase precedente si vorrebbero estrarre tre fatti distinti:

Obama is a supporter of Chicago White Sox
Obama is a supporter of Chigago Bears
Obama is a supporter of Pittsburgh Steelers

Il nostro approccio si é basato quindi sull'utilizzo di *Stanford NLP Core*.
Questa libreria é in grado di analizzare una frase in termini di dipendenze sintattiche, consentendo di ricostruire l'analisi logica e i collegamenti logici tra parole all'interno della frase stessa.
Il principale svantaggio che deriva dall'utilizzo di questa libreria é il costo richiesto per l'analisi di ciascuna frase. Abbiamo infatti calcolato che in media il modulo riesce a calcolare le dipendenze di massimo 150 frasi al minuto. Parte del lavoro é stata quindi incentrata sulla realizzazione di un prefiltro che fosse in grado di individuare potenziali frasi lista, al fine di alleggerire il carico di lavoro a cui la libreria di Stanford viene sottoposta.

4.6.1 Prefiltraggio di frasi

La realizzazione del filtro é stata guidata da due esigenze particolari:

1. **efficienza**: tutte le 56 milioni di frasi presenti nel dump di Wikipedia devono essere passate al prefiltro. Non possono quindi essere tollerati tempi di esecuzione troppo lunghi per ciascuna singola frase.
2. **efficacia**: sostanzialmente si ambisce ad una precision molto alta, assicurandoci quindi di non andare a spendere esecuzioni inutili della libreria di Stanford.

Per sostenere l'efficienza abbiamo deciso di parallelizzare l'esecuzione del prefiltro attraverso l'utilizzo del Lightweight Executable Framework, utilizzando regex che richiedessero tempi di esecuzione ridotti. Lo speedup ottenuto é di circa 4 e il tempo di esecuzione inferiore ai 20 minuti.
Abbiamo quindi deciso di procedere secondo un approccio basato sull'attribuzione di un punteggio a ciascuna singola frase e sulla definizione di una soglia al di sotto della quale le frasi in input fossero giudicate come non-lista.
Il processo di attribuzione del punteggio si basa sulla ricerca di alcune caratteristiche strutturali all'interno della frase analizzata. In particolare:

+0.9 se contiene una sequenza di entitá dello stesso tipo (la verifica viene effettuata andando a verificare le classi di appartenenza di ciascuna entitá su *yagoTransitiveType*) .
+0.5 se contiene almeno un'entitá che segue un *list identifier*: una locuzione come *such as, like,both,...*
+0.35 se contiente un'entitá che segue un *quantificatore*: *some, many, plenty, few,..*
+0.3 se nel primo sesto della lunghezza della frase é contenuta almeno un'entitá
+0.2 se sono contenuti dei numeri (numerici o letterali) seguiti da un numero di virgole pari al numero -1.
+0.15 per ogni entitá contenuta tra virgole

+0.1 per ogni virgola presente

Prima di passare le frasi a questa componente peró, attraverso tre espressioni regolari, é stato effettuato un consistente processo di filtraggio del file originale *sentences_all.tsv*.
Innanzitutto sono state prese in considerazione soltanto le frasi composte da almeno 3 entitá:

$$egrep\ ".*|.*|.*|.*" \quad (5)$$

Da questa regex sono state scartate circa **40 milioni** di frasi del file originale.
Ci siamo peró resi conto che potrebbe non essere una scelta ottimale:

[[Obama...]] is a supporter of Chicago White Sox, Chicago Bears, and [[Pittsburgh Steelers..]].

Questa frase verrebbe scartata e non sarebbe trattata come frase lista, ragion per cui l'approccio semplice di estrazione di fatti non riuscirebbe ad estrarre nulla. Ad ogni modo, la maggior parte di frasi lista individuate non é soggetta a questo tipo di vulnerabilitá. Effettuato questo primo filtraggio abbiamo definito una nuova regex che, dalle **15.597.980** frasi rimanenti, individuasse quelle con una sequenza di entitá vicine.

$$egrep\ "]][a\text{-}zA\text{-}Z0\text{-}9\ \backslash",]\{0,15\}[[" \quad (6)$$

L'output é stato un file composto da **11.457.798** di frasi. Osservando la tipologia di frasi ottenute abbiamo quindi pensato di prendere soltanto quello che avessero una coppia di entitá separate da un *"and"*:

$$entitá="\backslash[\backslash[a\text{-}zA\text{-}Z0\text{-}9\_()/,\backslash.\&\text{-}]*\backslash|m.[a\text{-}zA\text{-}Z0\text{-}9\_]*\backslash]\backslash]" \quad (7)$$

$$egrep\ "\ '\$entitá'.\{0,7\}\ and\ .\{0,7\}'\$entitá'\ " \quad (8)$$

che ha restituito **3.009.416** di frasi.
A questo punto, quindi, **3.009.416** frasi sono state passate attraverso il modulo di attribuzione del punteggio, che ha scartato tutte quelle al di sotto della soglia **1.65**. Alla fine le frasi candidate a contenere liste di entitá, passate quindi allo Stanford NLP Core, sono state **302.533**. Vengono fornite, nell' Appendice A, alcuni esempi delle frasi che hanno passato l'intero processo di riconoscimento di frasi lista.

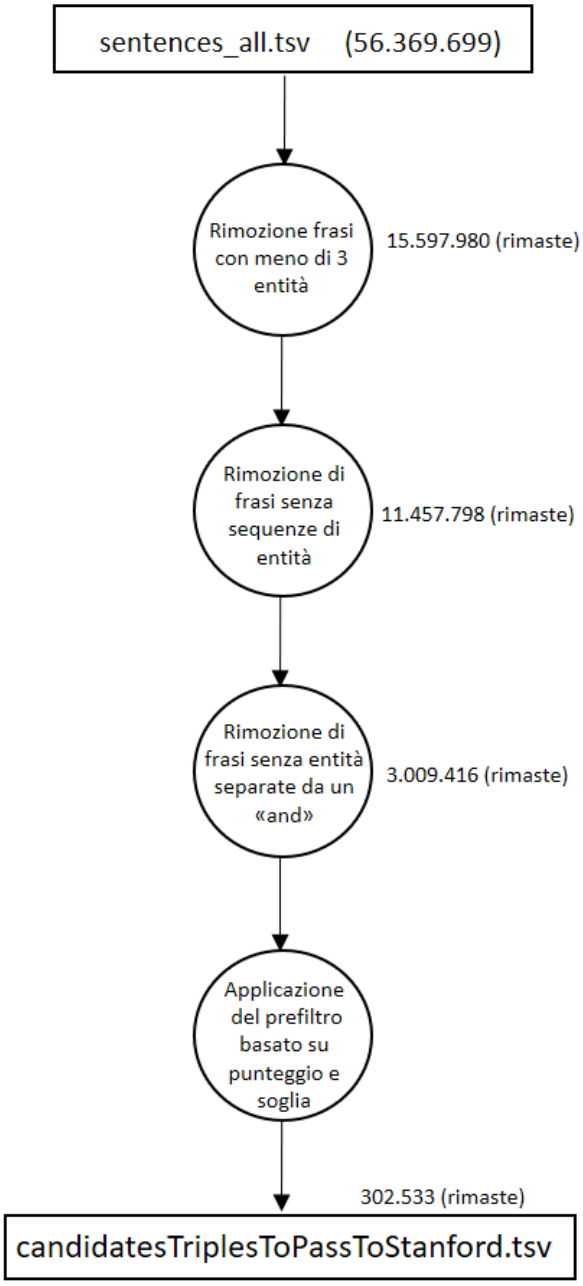

Fig. 5. Processo di prefiltraggio dellle frasi lista

4.6.2 Valutazione del prefiltro

Il prefiltro, per come é stato realizzato, in particolar modo per la scelta dell'attribuzione di un punteggio in base alle proprietá strutturali della frase e alla presenza di una soglia, si presta particolarmente bene all'uso di un classificatore.
Abbiamo peró scelto di procedere facendo un semplice tuning sui parametri e sull'attribuzione del punteggio in base ai risultati ottenuti. Sono quindi state etichettate 600 frasi, distinguendo le frasi lista da quelle non lista e dividendole quindi in training set e validation set. Il tuning sui parametri é stato effettuato manualmente e, nella migliore verifica sul validation set abbiamo ottenuto i seguenti risultati:

1. **Precision**:0.43
2. **Recall**:0.88
3. **Fallout**:0.18

4.6.3 Utilizzo di Stanford NLP Core

Una volta individuate le frasi candidate a contenere liste di fatti viene eseguita la libreria Stanford NLP Core su tale insieme di frasi. Ciascuna frase in input viene dunque analizzata dalla libreria, che é in grado di restituire la lista di dipendenze logiche all'interno delle frasi.
Le dipendenze rilevanti per le frasi lista sono risultate essere le *nmod*, che sono generalmente usate per modifica-

tori nominali e per predicati con clausole. [3] Nella maggior parte di frasi lista, si presenta infatti un numero ristretto di tipologie di nmod: **nmod:such_as**, **nmod:like**, **nmod:with** e **nmod:in** sono risultate le piú ricorrenti.

Individuate quindi le dipendenze nmod all'interno della frase il nostro approccio si é basato sul riconoscimento dei *governor* e dei *dependent* di ciascuna di esse.
Questi concetti si riferiscono alle estremitá della relazione e quindi alle parole coinvolte, come mostrato in Figura 6. A questo punto la frase viene suddivisa in sottoperiodi logici in base al numero di nmod rilevati, nella condizione in cui i dependent individuati siano almeno due. Essenzialmente si adotta questo processo perché le dependent di ciascun nmod rappresentano le possibili liste di entitá contenute nella sequenza di fatti che si vuole estrarre.
La fase successiva consiste nell'individuazione del soggetto del sottoperiodo in esame. Si procede quindi con la navigazione delle dipendenze a partire dal governor comune degli nmod, cercando il ramo che porta alla dipendenza *nsubj*. Il dependent di nsubj é proprio il soggetto del sottoperiodo.
A questo punto del processamento sostanzialmente sono state individuate le entitá coinvolte nel fatto da estrarre, il passo successivo consiste dunque nel riuscire a localizzare la frase caratterizzante il fatto stesso.
Vengono quindi richiesti gli indici corrispondenti al soggetto e alla prima entitá coinvolta nella lista. Tali indici corrispondono alla posizione della parola all'interno della frase.
Prendendo in considerazione l'esempio fornito nella Figura 6 avremmo che gli indici in questione sono 1, quello di "Obama", e 10, "Chicago".
Presa la frase delimitata dagli indici individuati si prosegue con il Pos Tagging, eseguito ancora dalla libreria di Stanford. L'obiettivo di questa fase é l'individuazione dei verbi e delle congiunzioni subordinate[4].
Queste operazioni sono risultate particolarmente rilevanti perché le frasi individuate vengono poi state utilizzate per la ricerca di nuovi fatti.
I principali problemi riscontrati sono i seguenti:

1. prendere tutta la frase compresa tra gli indici, escludendo magari soltanto i list identifiers, come ad esempio "such as" o "like", compromette la possibilitá di generalizzare la frase.
   Poiché le frasi estratte vengono successivamente passate al filtro relazionale si rischierebbe di perdere tutte le frasi eccessivamente lunghe.
2. prendere soltanto il verbo all'interno del periodo rischia invece di stravolgerne il significato, con la possibilitá concreta di inserire fatti sbagliati nel knowledge graph.

Abbiamo quindi cercato dei compromessi tra queste soluzioni opposte andando ad includere anche le preposizioni e le congiunzioni subordinate presenti tra il verbo e la prima entitá della lista. Con riferimento ancora alla Figura 6 otterremmo come output:

Obama is a supporter of Chicago White Sox
Obama is a supporter of Chigago Bears
Obama is a supporter of Pittsburgh Steelers

Su questo tipo di frasi elementari e ben definite i risultati sono piú che soddisfacenti, tuttavia all'aumentare della complessitá del periodo, l'output peggiora sensibilmente. In particolare, si verifica spesso che la frase estratta ha un significato diverso o parziale rispetto a quella originale. Al termine del processamento le frasi estratte vengono quindi passate al filtro relazionale, per poi seguire il processo di estrazione di fatti presentato in precedenza. I tempi di esecuzione di questa componente sono risultati estremamente lunghi e per questo é senza dubbio necessario scalare orizzontalmente, splittando il file ed eseguendo il calcolo di dipendenze su piú calcolatori. Anche la valutazione del processo di estrazione di fatti da frase lista, di conseguenza, é stata effettuata su un campione di **10.000** frasi passate alla libreria di Stanford.

5 Valutazione dei fatti estratti

Al termine del processo di implementazione abbiamo quindi valutato i fatti estratti mediante tecniche diverse:

1. fatti estratti senza generalizzazione di frasi e senza preprocessamento
2. fatti estratti mediante generalizzazione
3. fatti estratti mediante generalizzazione e stopping
4. fatti estratti dalle frasi lista

Abbiamo quindi preso in considerazione le medesime relazioni usate per la valutazione dei nuovi fatti estratti su FreeBase, impostando inoltre la probabilitá minima di *0.5* e *K=20*. Per la tecnica di estrazione senza preprocessamento abbiamo valutato 470 fatti, per quelle basate sul preprocessamento 240 ciascuna e per l'analisi delle frasi lista circa 280. Abbiamo inoltre adottato una soluzione di crowdsourcing, rivolgendoci a quattro nostri colleghi. Il costo di tale operazione é stato quantificato in un caffé ogni 80 fatti valutati. Sebbene, paragonando il carico di lavoro e il costo relativo alle offerte presenti su Amazon Mechanical Turk ci siamo accorti che la ricompensa era troppo elevata, abbiamo comunque accettato il compromesso, trattandosi di persone qualificate.

[3] http://universaldependencies.org/docs/en/dep/
[4] http://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

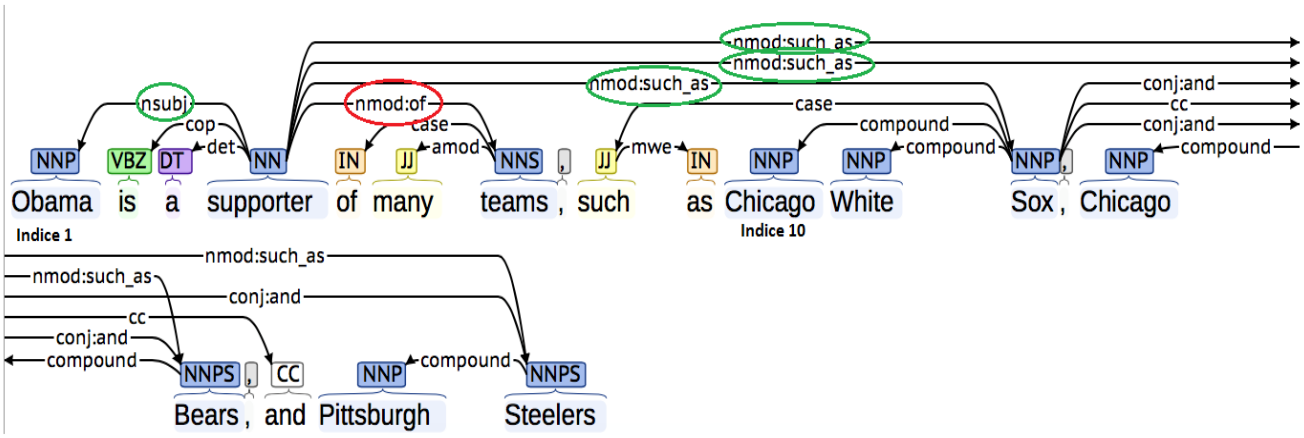

Fig. 6. Nella figura vengono mostrate le dipendenze individuate tramite la libreria di Stanford. Gli nmod rilevanti, cerchiati in verde, sono quelli che presentano diversi dependent. nmod:of viene quindi scartato in quanto presenta una sola dependent.

Table 3. In tabella i risultati ottenuti applicando Lector su **FreeBase**

Relazione	# Fatti in FreeBase	# nuovi fatti	# fatti valutati	# accuratezza
birthPlace	*662.192*	*57.140*	*347*	*88,9%*
deathPlace	*178.849*	*18.458*	*104*	*80,6%*
nationality	*584.792*	*50.234*	*290*	*95,6%*
team	*145.080*	*49.809*	*286*	*96,5%*
almaMater	*378.043*	*46.342*	*286*	*98,3%*
spouse	*130.425*	*14.939*	*97*	*31,6%*
child	*141.860*	*3.149*	*50*	*38,8%*
award	*98.625*	*1.934*	*50*	*96,4%*
party	*65.300*	*3.684*	*50*	*94,5%*

Table 4. In tabella i risultati ottenuti applicando Lector su **YAGO** senza generalizzazione e stopping

Relazione	# Fatti in YAGO	# nuovi fatti	# fatti valutati	# accuratezza
~~wasBorn~~	~~*280.942*~~	~~*57.837*~~	~~*40*~~	*90,4%*
diedIn	*94.728*	*22.203*	*40*	*81,6%*
isCitizenOf	*36.253*	*28.763*	*40*	*96,3%*
playsFor	*525.358*	*23.716*	*40*	*87,9%*
graduatedFrom	*51.478*	*50.229*	*40*	*96,3%*
isMarriedTo	*34.219*	*16.119*	*40*	*35,1%*
hasChild	*42.330*	*14.617*	*40*	*25,4%*
hasWonPrize	*120.465*	*25.736*	*50*	*93,1%*
isPoliticianOf	*32.774*	*630*	*50*	*82,5%*
actedIn	*126.236*	*2.385*	*50*	*96,7%*
Totale fatti estratti (contando tutte le relazioni)		*493.279*		