UNIVERSITÀ DEGLI STUDI DI FIRENZE
Facoltà di Ingegneria - Dipartimento di Sistemi e Informatica

Tesi di Laurea Specialistica in Ingegneria Informatica

Estensione e filtraggio di tag per annotazione di video basata su Semantic Web

Candidato
Mirco Nencioni

Relatori
Prof. Alberto Del Bimbo

Ing. Marco Bertini

Correlatori
Ing. Lamberto Ballan

Ing. Giuseppe Serra

Anno Accademico 2009/2010

Ringraziamenti

Desidero porgere sentiti ringraziamenti alle persone che hanno fornito il loro prezioso supporto durante lo svolgimento del mio lavoro di tesi. Ringrazio in particolare il Prof. Alberto Del Bimbo, per avermi permesso di affrontare un'esperienza di studio molto corposa su un tema di ricerca assai attuale. Un forte ringraziamento va anche all'Ing. Marco Bertini, all'Ing. Lamberto Ballan e all'Ing. Giuseppe Serra, ricercatori del Media Integration and Communication Center (MICC) di Firenze. Grazie al loro costante apporto ed al forte interesse dimostrato, essi hanno contribuito a risolvere di volta in volta le problematiche riscontrate durante questo lavoro.

Nel corso della mia esperienza universitaria, ho preparato gli esami assieme ad altri tre infaticabili compagni di corso: Salvatore Smeraldo, Alessandro Ferri e Tommaso Bruschi. Con essi ho condiviso gioie e dolori degli ultimi sette anni, e se sono arrivato al termine del mio percorso formativo posso affermare con certezza che è anche merito loro.

Ai miei genitori dedico infine un ringraziamento speciale per avermi permesso di completare gli studi.

Indice

Capitolo 1

Introduzione

Negli ultimi anni lo sviluppo deciso di Internet ha favorito la nascita di numerosi web sites volti alla condivisione di materiale multimediale. In particolare, hanno registrato una forte crescita i siti web destinati all'archiviazione di foto o video, come Flickr (immagini) e **YouTube** (filmati). In queste sorgenti di dati, il contenuto multimediale è corredato da informazioni aggiuntive. Molto spesso infatti, alle immagini ed ai video vengono affiancate note testuali quali il titolo, la descrizione, l'utente di provenienza ed alcune etichette (**tag**). I tag hanno un ruolo fondamentale all'interno del suddetto contesto, in quanto rendono possibile la ricerca dei contenuti.

Un aspetto di notevole importanza legato al "tagging" dei video consiste nel fatto che le etichette, spesse volte, non vengono inserite in modo corretto dagli autori dei filmati. Accade, infatti, che i tag siano pochi o al contrario troppo numerosi, e che quindi la percentuale di rumore da loro introdotta risulti alta.

Questo lavoro di tesi si inserisce nel contesto del *Tag Learning*, ed in particolare mira, come primo obiettivo, ad individuare una strategia di filtraggio e ranking dei tag. A partire da un video di YouTube, si estraggono le relative

etichette e si mette in pratica un algoritmo al termine del quale solo un sottoinsieme dei tag iniziali viene utilizzato nelle fasi di elaborazione successive. Tale operazione di filtraggio è necessaria dato che uno degli obiettivi di questo lavoro di tesi è quello di individuare un set finale di etichette *affidabili* per ogni filmato.

In tale quadro operativo si inserisce la seconda parte della tesi. L'idea di fondo è quella di stabilire un collegamento tra i tag validati di ogni video ed un'ontologia di riferimento. In altre parole, si vogliono derivare informazioni semantiche dal contenuto analizzato, collocandosi a tutti gli effetti nel contesto del **Semantic Web**. Questa parte del lavoro ha come obiettivo l'incremento dei metadati associati ad un filmato. I soli tag, infatti, non sono sufficienti a definire con certezza ciò che il video rappresenta oppure qual è il contesto nel quale si colloca. Dopo aver esplorato le due ontologie ad oggi più diffuse (*Wordnet*, *DBpedia*), si è deciso di realizzare l'espansione semantica dei video tramite la combinazione di *DBpedia* e *Wikipedia Miner*, un toolkit capace di derivare il grado di "parentela" tra termini messi a confronto.

Il processo di espansione semantica ha due scopi: l'arricchimento dello spazio iniziale dei tag e l'estrazione di ulteriore contenuto informativo da associare al video in analisi. In questa fase, infatti, si riesce ad incrementare il numero di etichette di ogni filmato grazie all'integrazione con il livello ontologico. Il risultato è un rafforzamento dei metadati legati a ciascun video; la relativa descrizione conterrà nuove informazioni derivanti dalle ontologie ed i tag risulteranno aumentati. Ognuno degli aspetti fin qui menzionato sarà ampiamente trattato nel Capitolo 4, dedicato alla descrizione del sistema sviluppato.

L'integrazione dei tag con il Semantic Web apre la strada al terzo ed ultimo obiettivo di questo lavoro di tesi. A partire dal set finale delle etichette, si vuole ottenere una serie di immagini ad esso legata. Il motivo di tale necessità risiede

nel lavoro svolto da Smeraldo [Smeraldo10], rispetto al tema della *localizzazione temporale dei concetti presenti in un video*. Il presente lavoro di tesi e quello di Smeraldo sono direttamente legati; la validazione dei tag ed il relativo arricchimento mira ad ottenere valide immagini da utilizzare per la localizzazione dei concetti all'interno di un filmato. In questa fase, un nodo cruciale consiste nella scelta della sorgente di dati. La decisione è stata quella di lavorare con Flickr, riconosciuto in letteratura come il punto di riferimento per quanto concerne gli archivi di foto. Flickr gestisce più di 2 miliardi di immagini e può registrare ogni giorno circa 3 milioni di nuovi upload. In questa tesi non saranno naturalmente trattati gli aspetti tecnici relativi alla localizzazione temporale dei concetti in un video.

Per concludere, un breve accenno circa la piattaforma software utilizzata per lo svolgimento del progetto. A tal proposito, il software è stato sviluppato in linguaggio Java, in combinazione al Sistema Operativo *Ubuntu 10.04 LTS* (basato su GNU/Linux, nella versione a 32 bit).

Nei prossimi capitoli vengono trattati gli aspetti teorici ed implementativi collegati al presente lavoro di tesi. Nell'ultima parte della trattazione sono stati inseriti i risultati numerici delle prove sperimentali, al fine di dare una valutazione quantitativa al metodo sviluppato.

Capitolo 2

Stato dell'arte

In questo capitolo saranno presentati gli approcci esistenti in letteratura riguardo al problema in studio, già descritto a grandi linee nel Capitolo Introduzione. Tra tutti i lavori attualmente disponibili sul tema del Tag Learning, sono stati scelti quelli che per vari aspetti riflettono maggiormente il nostro caso. Al termine del presente capitolo, identificheremo singolarmente il contributo di ogni metodo. In questo modo verranno evidenziate le idee che hanno rappresentato la base di partenza per l'intero lavoro di tesi. Ogni metodo descritto è corredato da un paper di riferimento, le cui coordinate sono naturalmente disponibili nella sezione *Bibliografia*. Per ogni approccio, viene fornita una descrizione relativamente sintetica delle sue idee cardine, mettendone in luce, al termine, le rilevanze sperimentali.

2.1 Learning Social Tag Relevance by Neighbor Voting

2.1.1 Introduzione

L'analisi delle immagini ed il loro ritrovamento è importante per aiutare le persone ad organizzare ed accedere il contenuto multimediale corredato da tag. Dal momento che i tag ins eriti dagli utenti sono incontrollati, ambigui ed altamente "personalizzati", un problema fondamentale è capire come interpretare la loro rilevanza, in accordo al contenuto visuale che stanno descrivendo. Intuitivamente, se persone differenti etichettano con gli stessi tag immagini visualmente simili, allora i tag dovrebbero riflettere fedelmente gli aspetti oggettivi del contenuto visuale. Partendo da questa intuizione, gli autori [Snoek09] propongono un *Neighbor Voting Algorithm* che apprende con grande accuratezza la rilevanza di ogni etichetta (l'idea di base è quella di sommare le presenze di ogni tag all'interno del set di immagini simili). Gli esperimenti sono stati effettuati su 3.5 milioni di immagini estratte da Flickr[1], ed hanno dimostrato la generale applicabilità del metodo nel caso di *Image Retrieval* e di *Image Tag Suggestion*, con ottime prospettive per le applicazioni del mondo reale.

2.1.2 Learning Tag Relevance by Neighbor Voting

L'idea di base di questo metodo suggerisce che data un'immagine, la rilevanza di ognuno dei suoi tag può essere inferita verificando il numero di volte che il tag in questione compare all'interno del set di immagini simili. In base a questo, maggiore è la frequenza di un tag all'interno del set, maggiore sarà la sua rilevanza. Una visione schematica del metodo è visibile attraverso la Figura 2.1. Non sempre le etichette che hanno una frequenza molto elevata

[1] www.flickr.com

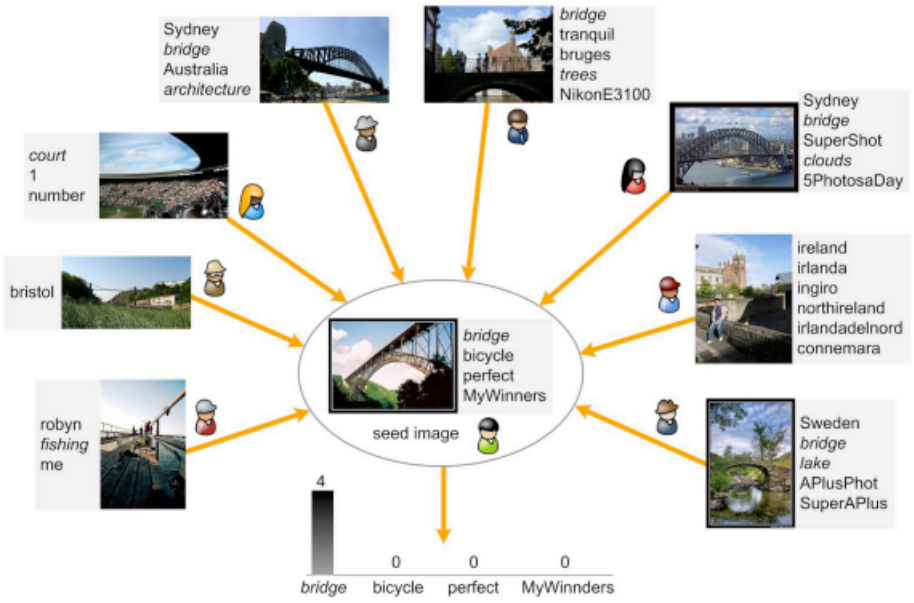

Figura 2.1: La rilevanza di un tag (*seed image*) è stimata attraverso i "voti" dei suoi visual neighbors.

sono buone; il rischio infatti è quello di attribuire una rilevanza alta a quei tag che in realtà sono troppo generali. Una buona misura della rilevanza deve perciò considerare la distribuzione dei tag nel set di immagini similari ma anche all'interno dell'intera collezione.

2.1.2.1 L'apprendimento della rilevanza dei tag

E' necessario introdurre una notazione per trattare il caso in esame. Si denota con Φ una collezione di immagini dotate di *user-tag* e con W un vocabolario di tag utilizzato in Φ. Per un'immagine $I \in \Phi$ ed un tag $w \in W$, sia $r^*(w, I)$: $\{W, \Phi\} \longmapsto \mathbb{R}$ una misura della rilevanza del tag. Tale misura soddisfa i seguenti 2 criteri:

- **Image Ranking**: date due immagini $I_1, I_2 \in \Phi$ ed un tag $w \in W$, se w è

rilevante rispetto a I_1 ma irrilevante rispetto a I_2, allora

$$r^*(w, I_1) > r^*(w, I_2) \tag{2.1}$$

- **Tag Ranking**: dati due tag $w_1, w_2 \in W$ ed un immagine $I \in \Phi$, se I è rilevante rispetto a w_1 ma irrilevante rispetto a w_2, allora

$$r^*(w_1, I) > r^*(w_2, I) \tag{2.2}$$

Lo scopo dell'algoritmo è trovare una misura di rilevanza dei tag che soddisfa i due criteri.

2.1.2.2 L'utilizzo dei visual neighbors per il calcolo della rilevanza

Si deve studiare il modo nel quale sono etichettate le immagini rilevanti ed irrilevanti per un tag. In un database di foto annotate, è possibile che per un tag specifico w, il numero di immagini irrilevanti rispetto a w sia significativamente maggiore del numero di immagini rilevanti, ovvero $\mid R_w^c \mid \gg \mid R_w \mid$, dove $\mid \bullet \mid$ è l'operatore di cardinalità sul set di immagini.

Sia $L_w \subset \Phi$ il set di foto della collezione etichettate con w. Può essere fatta un'assunzione riguardo al comportamento del tagging: "in un database di grande dimensione, la probabilità di tagging corretto è superiore rispetto a quella di tagging errato". Si analizza la distribuzione delle immagini rilevanti ed irrilevanti rispetto a w all'interno del set dei k *nearest neighbors*. Nel primo scenario, la ricerca visuale si comporta come il campionamento casuale ed il numero di immagini rilevanti nel neighbor set coincide con quello delle immagini rilevanti all'interno del set estratto casualmente. Nel caso in cui la ricerca visuale sia migliore rispetto al campionamento casuale, date due immagini I_1 (rilevante per w) ed I_2 (irrilevante per w), ci si aspetta di ottenere:

$$| N_f(I_1, k) \cap R_w | > | N_{rand}(k) \cap R_w | > | N_f(I_2, k) \cap R_w | \tag{2.3}$$

dove f rappresenta una funzione di similarità tra due immagini, basata su features di basso livello. Il numero di immagini rilevanti per il tag w all'interno del neighbor set è espresso come:

$$| N_f(I, k) \cap R_w | = k \cdot (P(R_w) + \epsilon_{I,w}) \tag{2.4}$$

dove $(P(R_w) + \epsilon_{I,w})$ è la probabilità che un immagine selezionata casualmente dal neighbor set sia rilevante rispetto a w. A questo punto è necessario fare una seconda assunzione: "una ricerca visuale basata sul contenuto è migliore rispetto ad un campionamento casuale". Dividendo il neighbor set in due distinti sottoset $N_f(I, k) \cap R_w$ e $N_f(I, k) \cap R_w^c$, si conta il numero di occorenze di w all'interno dei due sottoset:

$$n_w[N_f(I, k)] = n_w[N_f(I, k) \cap R_w] + n_w[N_f(I, k) \cap R_w^c] =$$

$$= k \cdot (P(R_w) + \epsilon_{I,w}) P(w \mid R_w) + k \cdot (P(R_w^c) - \epsilon_{I,w}) P(w \mid R_w^c) \tag{2.5}$$

In modo simile si può derivare:

$$n_w[N_{rand}(k)] = k \cdot (P(R_w) P(w \mid R_w) + P(R_w^c) P(w \mid R_w^c)) \tag{2.6}$$

La quantità $n_w[N_{rand}(k)]$ riflette la frequenza di occorrenza di w nell'intera collezione di immagini, e può quindi essere denotata con $Prior(w, k)$. Sostituendo (2.6) in (2.5) si ottiene:

$$n_w[N_f(I, k)] - Prior(w, k) := k \cdot (P(w \mid R_w) - P(w \mid R_w^c)) \epsilon_{I,w} \tag{2.7}$$

In definitiva, la rilevanza del tag w rispetto all'immagine I ed al set dei suoi k neighbors si definisce come:

$$tagRelevance\,(w, I, k) := n_w\,[N_f\,(I, k)] - Prior\,(w, k) \tag{2.8}$$

2.1.2.3 Neighbor Voting Algorithm

L'espressione (2.8) esprime come la rilevanza del tag w dipenda dal numero delle sue occorrenze nei k nearest neighbors dell'immagine I e dalla frequenza a priori del tag stesso. Considerando che ogni nearest neighbors vota circa la presenza/assenza del tag w, $n_w\,[N_f\,(I, k)]$ indica il numero di voti dei "vicini" sul tag w. Si introduce quindi un *Neighbor Voting Algorithm*: in primo luogo viene effettuata una ricerca visuale per identificare i neighbors di un'immagine corredata di tag, dopo di che vengono usati i tag di ogni vicino per votare sui tag dell'immagine di riferimento. Si approssima la frequenza a priori del tag w come:

$$Prior\,(w, k) \approx k\frac{\mid L_w \mid}{\mid \Phi \mid} \tag{2.9}$$

dove k è il numero di *visual neighbors*. $\mid L_w \mid$ indica il numero di immagini etichettate col tag w, mentre $\mid \Phi \mid$ è la dimensione dell'intera collezione. Si può notare che l'espressione (2.8) non restituisce necessariamente valori di rilevanza positivi; per questa ragione il valore minimo di $tagRelevance\,(w, I, k)$ è impostato a 1. In altre parole, se la rilevanza di un user-tag è minore della sua frequenza originale all'interno dell'immagine alla quale appartiene, si "rigetta" tale quantità.

In aggiunta, si noti come i risultati della procedura fin qui descritta siano influenzati dal numero di foto che ogni utente possiede all'interno del neighbor set. Per rendere più oggettivo il *Neighbor Voting Algorithm*, quindi, si impone il vincolo di *unique-user*. Esso impone che un utente abbia al massimo 1 immagine

all'interno del neighbor set. I risultati sperimentali dimostrano che tale vincolo è capace di ridurre il bias dell'algoritmo di voting.

2.1.3 Gli esperimenti

L'algoritmo proposto è stato valutato nel caso di *Image Ranking* e *Tag Ranking*. Nel primo caso, vengono comparati 3 metodi di Image Retrieval basati sui tag, rispetto alla presenza/assenza dell'algoritmo basato sulla rilevanza. Nel secondo caso, vengono valutate le potenzialità dell'algoritmo nell'ambito dell' ausilio al "tagging" svolto dagli utenti. In definitiva, gli autori hanno proposto tre tipi di esperimenti, descritti nelle sezioni (2.1.3.1), (2.1.3.2), (2.1.3.3). Facciamo un cenno al dataset attraverso il quale gli autori hanno valutato il metodo fin qui descritto.

Nel caso di *Image Ranking*, vengono selezionati come query 20 concetti visuali. Per ogni interrogazione si prendono casualmente 1000 esempi (immagini etichettate) all'interno di una collezione composta da 3.5 milioni di foto estratte da Flickr. Tali esempi vengono annotati nuovamente secondo il criterio di rilevanza descritto nelle precedenti sezioni, dopo di che vengono classificate le 1000 immagini di test con 2 metodi baseline e con l'algoritmo proposto. Nel caso di *Tag Ranking*, invece, viene adottato il dataset proposto in [VanZowl08]. Il set di foto consiste di 331 immagini estratte da Flickr, senza sovrapposizioni all'interno della collezione di 3.5 milioni di foto. Tutte le 331 immagini vengono utilizzate come test.

2.1.3.1 Tag-based Image Retrieval

Data una query q contenente keywords $\{w_1, ..., w_n\}$, la rilevanza di un'immagine I è computata come:

$$score\,(q, I) = \sum_{w \in q} qtf\,(w)\, idf\,(w)\, \frac{tf\,(w) \cdot (k_1 + 1)}{tf\,(w) + k_1 \cdot \left(1 - b + b \frac{l_I}{l_{avg}}\right)} \tag{2.10}$$

dove $qtf\,(w)$ è la frequenza del tag w in q, $tf\,(w)$ è la frequenza di w nei tag di I, l_I il numero totale di tag di I mentre l_{avg} è il valore medio di l_I nell'intera collezione. La funzione $idf\,(w)$ è calcolata come $log \frac{N - |L_w| + 0.5}{|L_w| + 0.5}$, dove N è il numero di immagini nella collezione e $\mid L_w \mid$è il numero di immagini etichettate con w.

Gli esperimenti sono stati condotti comparando il metodo baseline con l'algoritmo descritto in questa sezione, al variare di tre parametri. Il primo, k , indica il numero di neighbors per una data immagine, il secondo, $b\,(0 \leq b \leq 1)$ controlla il numero di tag in un'immagine mentre il terzo, k_1, è il parametro di regolarizzazione della frequenza dei tag.

2.1.3.2 Tag suggestion for labeled images

Si valuta come l'algoritmo proposto migliora le prestazioni di [VanZowl08] introducendo l'informazione sul contenuto visuale nel processo di suggerimento dei tag. Similarmente a [VanZowl08], si cercano innanzitutto x candidati tag aventi la più alta co-occorrenza con i tag iniziali. Per ogni candidato tag, si calcola il relativo punteggio di *rilevanza* rispetto all'immagine secondo l'espressione:

$$score\,(c, I) = score\,(c, w_I) \cdot \frac{\lambda}{\lambda + (rank_c - 1)} \tag{2.11}$$

dove c è il candidato tag, I rappresenta l'immagine e w_I il set di tag iniziali. La funzione $score\,(c, w_I)$ computa un punteggio di rilevanza tra il candidato

tag ed i tag iniziali. Gli autori hanno adottato il $Vote^{+}$, il miglior metodo utilizzato in [VanZowl08] per implementare la funzione *score*. Il valore $rank_c$ è la posizione del tag c nella lista dei candidati, secondo il valore discendente di *tagRelevance*. La variabile λ, infine, è un parametro per regolarizzare il calcolo della rilevanza dei tag.

2.1.3.3 Tag suggestion for unlabeled images

Il metodo viene comparato con gli approcci [Torralba08] e [Wang08]. Si considerano come candidati tutti i tag presenti nel vocabolario. Si stima il valore di *tagRelevance* per ogni candidato rispetto all'immagine non etichettata, quindi si stila una classifica basata sull'ordinamento descrescente della *tagRelevance* stessa. Il vincolo di *unique-user* viene rimosso, in quanto i metodi baseline non considerano le informazioni degli utenti. Per tutti i metodi in analisi si fissa a 500 il numero di visual neighbors, come suggerito in [Wang08]. Infine, per ogni metodo, si selezionano i migliori 5 tag come suggerimento finale per ogni immagine di test.

2.2 Flickr Tag Recommendation based on Collective Knowledge

2.2.1 Introduzione

I servizi di gestione delle foto on line come Flickr e Zooomr permettono agli utenti di condividere le loro immagini con la famiglia, gli amici e l'intera comunità del Web. Un aspetto importante di questi servizi è che gli utenti annotano manualmente le immagini con dei *tag*, legati al contenuto visuale della foto e capaci di aggiungere ad essa informazione semantica. Gli autori di questo approccio [VanZowl08] si sono prefissi di studiare in che modo è possibile assistere

gli utenti nel processo di tagging. L'articolo di riferimento offre un duplice contributo alla ricerca.

In primo luogo viene analizzato uno snapshot rappresentativo di Flickr e vengono tratte conclusioni circa il modo nel quale gli utenti annotano le foto. Basandosi sulle conclusioni relative a questo primo aspetto, gli autori propongono una strategia di raccomandazione delle etichette. Il risultato consiste in un set di tag raccomandati, da aggiungere alla foto di riferimento. I risultati sperimentali dimostrano la bontà del metodo, che riesce realmente a suggerire annotazioni valide per le immagini.

2.2.2 Il comportamento dei tag in Flickr

In questa sezione si descrive la collezione di foto Flickr e si entra nel dettaglio del comportamento degli utenti per quanto concerne le operazioni di annotazione. In particolar modo, ci interessa sapere: "come gli utenti effettuano il tagging delle foto?" e "cosa annotano gli utenti?".

Flickr è un servizio online di condivisione foto che contiene centinaia di milioni di immagini, inserite da più di 8.5 milioni di utenti registrati. Durante le ore di punta, viene effettuato l'upload di più di 12,000 foto al secondo, mentre il record sul numero di immagini inserite in un giorno ha superato le 2 milioni di unità. Vediamo di analizzare le caratteristiche basilari riguardo a come le foto vengono annotate su Flickr. La collezione a cui fanno riferimento gli autori consiste di circa 52 milioni di foto che contengono in totale 188 milioni di tag, dai quali sono identificabili 3.7 milioni di *unique-tag*. La Figura 2.2 mostra la distribuzione della frequenza dei tag su scala logaritmica per entrambi gli assi. L'asse x rappresenta i 3.7 milioni di unique-tag, ordinati per frequenza discendente. L'asse y, invece, indica il valore di frequenza di ogni tag. La Figura 2.3 mostra la distribuzione del numero di tag per foto, che segue, come

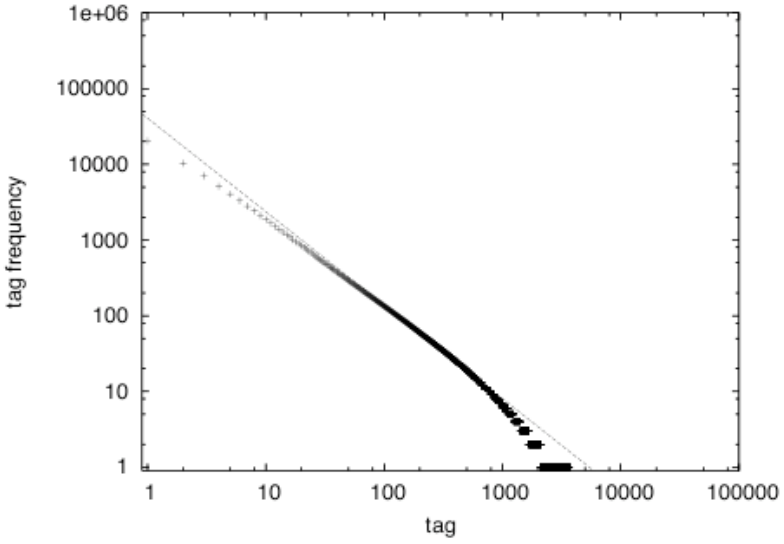

Figura 2.2: Distribuzione della frequenza dei tag in Flickr.

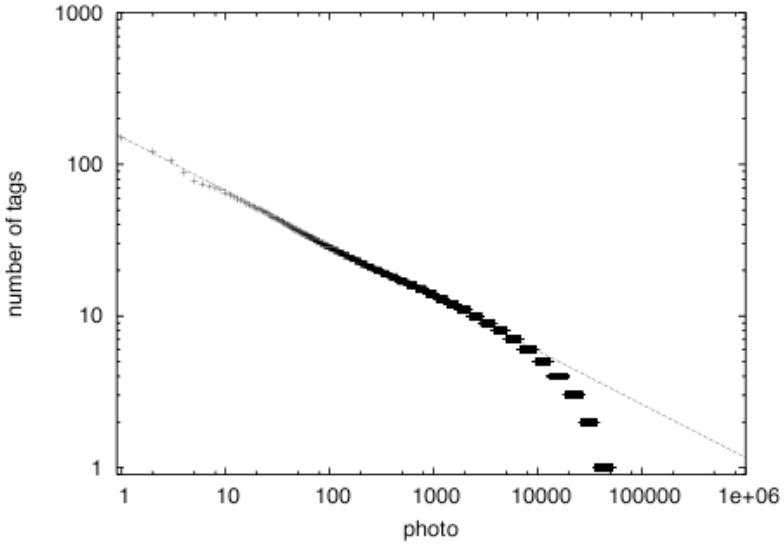

Figura 2.3: Distribuzione del numero di tag per foto in Flickr.

	Tags per photo	Photos
Class I	1	$\approx$ 15,500,000
Class II	$2-3$	$\approx$ 17,500,000
Class III	$4-6$	$\approx$ 12,000,000
Class IV	>6	$\approx$ 7,000,000

Figura 2.4: La definizione delle classi ed il numero di foto per ogni classe.

nel precedente caso, una legge di potenza. L'asse x rappresenta i 52 milioni di foto, ordinate per numero discendente di tag, mentre l'asse y si riferisce al numero di tag per ogni immagine.

Il sistema di raccomandazione delle etichette è stato analizzato a differenti livelli di esaustività; per far questo vengono definite quattro classi di immagini. Le classi differiscono per il numero di tag associati alle foto ivi contenute. La tabella in Figura 2.4 chiarisce quanto appena detto.

2.2.3 Le strategie di raccomandazione dei tag

In questa sezione viene data una descrizione dettagliata del sistema di raccomandazione dei tag. Iniziamo con una vista generale della sua architettura. A tal propostito, può essere utile la Figura 2.5. Data una lista di tag definiti dagli utenti, per ognuno di essi si crea una lista di m candidati tag, basandosi sulla *co-occorrenza*. Tale lista viene fornita come input ad un modulo di aggregazione e ranking, che produce la lista finale degli n tag raccomandati. Il numero attuale di tag raccomandati dipende naturalmente dalla rilevanza di quest'ultimi, e varia a seconda dell'applicazione.

2.2.3.1 La misura di co-occorrenza

La misura della *co-occorrenza* tra tag è la chiave dell'intero sistema di raccomandazione dei termini proposto in questo approccio. Esso lavora bene quando è

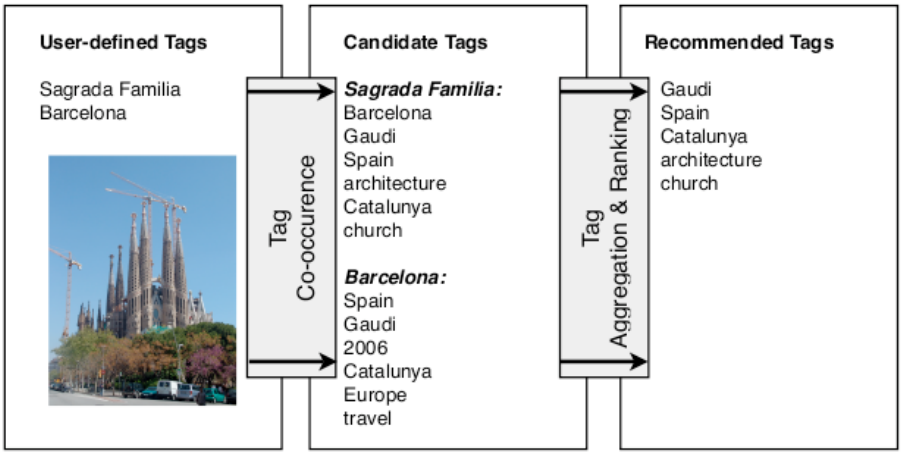

Figura 2.5: Overview del processo di raccomandazione dei tag.

disponibile una larga quantità di dati. Ovviamente, il numero di foto disponibili su Flickr soddisfa questo requisito e fornisce la base per rendere funzionante il sistema nelle applicazioni del mondo reale. La co-occorrenza tra due tag consiste nel numero di foto (all'interno della collezione) dove entrambi i tag sono presenti. Normalmente, tale grandezza è soggetta ad una normalizzazione basata sulla frequenza globale dei tag. Tale operazione ha come obiettivo rendere la misura maggiormente "informativa". Per chiarire questo concetto, è fondamentale introdurre le due misure tipiche di co-occorrenza:

- Misura **simmetrica**. In accordo al coefficiente di Jaccard, è possibile normalizzare la co-occorrenza tra due tag t_i e t_j calcolando:

$$J(t_i, t_j) := \frac{|\, t_i \cap t_j \,|}{|\, t_i \cup t_j \,|} \tag{2.12}$$

  In genarale, si può usare la misura simmetrica di Jaccard per capire se due tag hanno un significato simile.

- Misura **asimmetrica**. Alternativamente, la co-occorrenza può essere nor-

malizzata usando la frequenza di uno dei tag. Per esempio, tramite l'equazione:

$$P(t_j \mid t_i) := \frac{\mid t_i \cap t_j \mid}{\mid t_i \mid} \tag{2.13}$$

si esprime in che modo il tag t_i occorre col tag t_j, normalizzando rispetto alla frequenza totale di t_i. Questa quantità può essere interpretata come la probabilità che una foto venga annotata con t_j dato che è annotata con t_i.

Per evidenziare la differenza tra i due approcci, facciamo un semplice esempio, considerando il tag *Eiffel Tower*. Con la misura simmetrica, si trova che i tag maggiormente co-occorrenti sono (in ordine): *Tour Eiffel*, *Eiffel*, *Seine*, *La Tour Eiffel, Paris*. Utilizzando il criterio asimmetrico, invece, troviamo (in ordine): *Paris*, *France*, *Tour Eiffel*, *Eiffel*, *Europe*. Si può notare come il coefficiente di Jaccard in (2.12) sia valido quando si vuole identificare tag "equivalenti", come *Tour Eiffel, Eiffel, La Tour Eiffel*. Con la misura in (2.13), invece, si riesce ad ottenere una maggiore diversità nei tag proposti.

2.2.3.2 L'aggregazione e la promozione dei tag

Nel momento in cui è nota, per ogni *user-tag*, la lista dei tag candidati, è necessario uno step di aggregazione per ottenere una loro classifica globale. In questa sezione ci si riferisce a tre differenti tipi di tag:

- **User-defined tag** (U): è il set di tag che l'utente assegna ad una foto.
- **Candidate tag** (C_u): è la lista ordinata con gli m tag maggiormente co-occorrenti, per un tag $u \in U$. Si denota con C l'unione di tutti i candidati tag per ogni $u \in U$.
- **Recommended tag** (R): è la lista ordinata degli n tag più rilevanti prodotti dal sistema di raccomandazione.

A tal proposito, considerando la Figura 2.5, è possibile notare come, dato un set di tag candidati (C), venga attuata una procedura di ranking che porta alla formazione di una lista unica finale (R). In questa sezione vengono trattate due strategie di aggregazione: *vote* e *sum*. Descriviamo brevemente queste due diverse procedure:

- *Vote*. Questa strategia computa un punteggio per ogni candidato tag $c \in C$, dove un voto per c è dato, ogni volta che $c \in C_u$:

$$vote\,(u, c) = \begin{cases} 1 & c \in C_u \\ 0 & altrimenti \end{cases} \tag{2.14}$$

  La lista dei tag raccomandati è ottenuta ordinando i candidati tag sul numero di voti. Il punteggio è perciò computato come:

$$score\,(c) := \sum_{u \in U} vote\,(u, c) \tag{2.15}$$

- *Sum*. Questa strategia considera l'unione di tutti i candidati tag (C), e somma sui valori di co-occorrenza dei tag, perciò il punteggio di un candidato tag $c \in C$ è calcolato come:

$$score\,(c) := \sum_{u \in U} \left(P\,(c \mid u)\,, c \in C_u\right)$$

  La funzione $P\,(c \mid u)$ calcola il valore di co-occorrenza asimmetrica, come definito in (2.13).

L'obiettivo a questo punto è definire una sorta di "funzione di promozione", per promuovere i tag maggiormente rappresentativi. Guardando la Figura 2.2 si nota come gli estremi della legge di potenza non corrispondono in genere a buoni tag. I tag situati nella coda della curva, infatti, corrispondono a descrittori

	vote	**sum**
no-promotion	vote	sum
promotion	vote^+	sum^+

Figura 2.6: Le quattro strategie di raccomandazione dei tag esplorate in questo articolo.

instabili (infrequenti), mentre quelli in testa sono troppo generici per essere utili. L'idea è quella di creare una funzione di promozione basata su tre aspetti: *stabilità*, *descrittività*, *ranking*. Per il primo, si prende in considerazione la frequenza di un user-defined tag e la si inserisce in una funzione la cui forma esula dagli obiettivi di questo elaborato. Per il secondo, si cerca di diminuire il contributo dei tag candidati con alta frequenza, mentre per il terzo aspetto si considera essenzialmente la posizione r del tag candidato $c \in C_u$ per un user-defined tag u. Il risultato di questo processo è la creazione di una funzione di promozione della forma:

$$\mathit{promotion}\,(u,c) := \mathit{rank}\,(u,c) \cdot \mathit{stability}\,(u) \cdot \mathit{descriptive}\,(c) \tag{2.16}$$

Il punteggio di ogni candidato tag c viene quindi corretto in accordo a:

$$\mathit{score}\,(c) := \sum_{u \in U} \mathit{vote}\,(u,c) \cdot \mathit{promotion}\,(u,c) \tag{2.17}$$

In definitiva, come mostrato nel prossimo paragrafo, vengono valutate le prestazioni della funzione di promozione secondo quattro configurazioni. Per chiarire meglio questo aspetto, si consideri la Figura 2.6.

2.2.4 Gli esperimenti

Per i rilievi sperimentali gli autori hanno selezionato 331 foto estratte attraverso le API di Flickr. Le foto selezionate sono basate su una serie di concetti di alto

livello, del tipo “basketball”, “Iceland”, “sailing”, etc, e rispettano la tabella in Figura 2.4. Le immagini vengono quindi divise in *training set* e *test set* (rispettivamente 131 foto e 200 foto). Il ground thruth è stato creato manualmente da un pool di persone che hanno annotato le 331 immagini prese in considerazione. Per ognuna di esse, e per ognuna delle 10 raccomandazioni, vengono inseriti i giudizi di rappresentatività rispetto ad una scala a quattro livelli: *very good*, *good*, *not good*, *don't know*.

Per le valutazioni sperimentali sono state adottate tre metriche:

- **Mean Reciprocal Rank (MRR)**. Misura in che punto della classifica viene inserito il tag più rappresentativo, facendo la media su tutte le foto. Questa metrica viene utilizzata per capire quanto il sistema riesce ad inserire i tag migliori in testa alla classifica.

- **Success at rank k (S@k)**. Si utilizzano S@1 e S@5. Questa metrica è definita come la probabilità di trovare un tag descrittivo tra i migliori k tag raccomandati.

- **Precision at rank k (P@k)**. Si utlizza P@5. E'definita come la proporzione rilevante di tag ritrovati, facendo la media su tutte le foto.

2.3 Tag Suggestion and localization in user generated videos based on social knowledge

Da questo articolo [MICC10] sono state tratte idee molto interessanti per lo svolgimento del mio progetto di tesi. Viene presentato un sistema per il suggerimento dei tag di un video e per la localizzazione temporale degli stessi all'interno delle sequenze video. L'algoritmo suggerisce nuovi tag che possono essere

associati ad un keyframe analizzando i tag associati al video e alle immagini che si possono carpire da siti web come YouTube e Flickr.

2.3.1 Studio della rilevanza dei tag per l'annotazione di video

L'approccio proposto in questo articolo ha due scopi: l'estensione del numero di tag associati ad ogni video e l'associazione di tag agli shot rilevanti che compongono il filmato stesso.

L'annotazione di video viene realizzata in due passi. Nella prima fase viene calcolata la rilevanza dei tag per ogni shot andando però ad eliminare i termini non rilevanti, quindi ad ogni shot vengono aggiunti nuovi tag. Dopo aver estratto gli shot dal video, vengono estratti tre keyframe per ognuno di essi. I tag associati al video vengono utilizzati per selezionare e scaricare da Flickr un set di immagini che sono state annotate con i tag di partenza del video. L'unione di tutti i tag delle immagini scaricate da Flickr costituisce il dizionario con il quale possono essere annotati gli shot del filmato. Dato che le immagini di Flickr sono state taggate da principianti, è fondamentale valutare la rilevanza dei termini che compongono il lessico, per evitare che siano aggiunte annotazioni non corrette. A questo scopo è stato adattato l'algoritmo per la valutazione della rilevanza dei tag presente in [Snoek09] per poter funzionare con l'annotazione degli shot dei video. Il calcolo della rilevanza di un tag si basa sul numero di volte in cui esso compare nei k-nearest neighbors dell'immagine sottratto per la frequenza a priori del tag stesso. Questo richiede un calcolo efficiente dei k-nearest neighbors, dal punto di vista visuale, per il keyframe. Le immagini di cui viene valutata la similitudine rispetto al keyframe sono quelle scaricate da Flickr in base ai tag del video di partenza.

Per rappresentare l'informazione visuale dei keyframe e delle immagini, viene

calcolata una feature visuale a 72 dimensioni contenente informazioni globali relative a colore e tessitura. Il vettore è composto da un correlogramma a 48 dimensioni calcolato nello spazio di colore HSV, da 6 dimensioni date dal momento di colore calcolato nel dominio RGB e da un vettore a 18 dimensioni per tre feature di Tamura basate sulla tessitura (le feature considerate sono la *granularità*, il *contrasto* e la *direzionalità*). Alle feature delle immagini scaricate da Flickr viene applicato un algoritmo di k-mean clustering. Per ciascuno dei keyframe viene identificato il cluster rappresentato dalla parola visuale più vicina e le immagini che appartengono a quel cluster vengono selezionate come vicini.

Per ottenere risultati migliori, un tag viene mantenuto nella lista associata ad un'immagine solo se presente tra i tag delle immagini più vicine. Inoltre viene condotta un'espansione tramite WordNet mirata ad espandere i tag da associare a ciascuno shot. Il modo di calcolare la rilevanza di un tag è quello dato dall'algoritmo $Vote^{+}$proposto in [Snoek09]. Questo punteggio viene utilizzato per ordinare i termini da associare ad uno shot; i cinque tag più rilevanti vengono utilizzati in tal senso.

2.3.2 Risultati sperimentali

Le prestazioni sono state valutate utilizzando un dataset progettato per rappresentare la varietà dei contenuti di YouTube. Il dataset è stato creato scegliendo 4 video di YouTube da ognuna delle 14 relative categorie. In questo modo il numero di shot rilevati è 1135, per un totale di 3405 keyframe analizzati. Per ogni tag associato al video scaricato da YouTube, sono state scaricate le prime 15 immagini da Flickr in base al criterio della rilevanza fornito dalle relative API. Inoltre, il sistema scarica 5 immagini aggiuntive per ognuno dei sinonimi aggiunti grazie all'espansione realizzata con WordNet. La valutazione delle prestazioni è stata realizzata considerando la misura di *Accuracy*, calcolata come

la proporzione dei veri positivi rispetto al numero totale di veri e falsi positivi. Negli esperimenti le prestazioni sono state valutate in termini di:

STL (Shot level tag identification): valutazione delle prestazioni della localizzazione dei tag a livello shot. Questa misura mostra l'accuratezza della localizzazione dei tag negli shot considerando solo le annotazioni iniziali del video di YouTube. Il valor medio di *Accuracy* ottenuto è 0.63.

STSL (Shot level tag suggestion and localization): valutazione delle prestazioni della localizzazione dei tag a livello shot sia per i tag generati dagli utenti (originali) sia per quelli suggeriti. In questo caso il valore medio di *Accuracy* è 0.35.

STSL-WN (STSL with WordNet query expansion): misura delle prestazioni di STSL con l'espansione dei tag grazie ai sinonimi ottenuti da WordNet. Il valore medio di *Accuracy* è 0.36.

2.4 Dual Linkage Refinement For YouTube Video Topic Discovery

2.4.1 Introduzione

L'apprendimendo di video realistici ha assunto di recente un notevole interesse scientifico. Negli ultimi anni, sono nate svariate collezioni di user-generated video, tra le quali la più importante è senza dubbio quella gestita da YouTube[2]. Questa collezione è nata nel 2005 ed è rapidamente cresciuta; Wikipedia riporta che vengono caricate ogni minuto più di 20 ore di filmati. A causa delle dimensioni di questa collezione, non è semplice etichettare efficientemente i video. Si

[2]www.youtube.com

rende necessario quindi un meccanismo di *labeling automatico*. L'articolo di riferimento per questo approccio (del quale non abbiamo i riferimenti bibliografici) fornisce tre contributi:

- arricchimento del contesto testuale di ogni video: per ogni video, vengono esplorati i suoi *YouTube Related Videos*. La rappresentazione del video originale è basata sui termini che compaiono in quest'ultimo e nei video correlati.

- definizione di un metodo per scoprire i *topic* all'interno dei video: per topic si intende un oggetto o un evento che il video analizzato descrive.

- analisi dei video di risposta di YouTube per raffinare i risultati del video clustering.

2.4.2 L'arrichimento del contesto dei video

I video di YouTube presentano tre tipi di informazioni testuali: *titolo*, *descrizione*, *tag*. E'comune che alcuni termini importanti appaiono solo una o due volte all'interno dei metadati del filmato, e che una funzione di pesatura come ad esempio *tfidf* perda il loro contributo. Per innalzare le frequenza dei termini rappresentativi (per un topic), si utilizzano i Related Videos forniti da YouTube.

Ogni video V_i di YouTube ha un numero di video correlati minore o uguale a 60, indicati con $V_{r_i} = \left\{V_{r_{ij}} \mid j = 1, 2, ..., n, n \leq 60\right\}$. Si definisce con D_i la documentazione di V_i, che consiste di *titolo*, *descrizione* e *tag*. Si calcola la frequenza del termine t_i in D_i come:

$$tf^{'}(t_i) = tf(t_i) + df_r(t_i) \tag{2.18}$$

dove $df_r(t_i)$ si riferisce alla frequenza del termine t_i nei metadati dei Related Videos. Dopo questo step, viene attuata la procedura di Feature Selection in [Chang05] per eliminare i termini con poco significato.

2.4.3 L'estrazione dei topic

Dopo che il contesto dei video è stato arricchito, si costruisce un grafo G_t basato sulle relazioni di co-occorrenza tra video originale e video correlati. Il termine di correlazione è definito come:

$$r_{ij} = \frac{df(t_i \cap t_j)}{max\{df(t_i), df(t_j)\}} \tag{2.19}$$

dove $df(t_i \cap t_j)$ è la frequenza nella quale i termini t_i e t_j appaiono insieme. Il grafo G_t è costruito quindi con r_{ij} e t_i: ogni termine t_i forma un vertice v_i ed un link tra v_i e v_j esiste se $r_{ij} > mean(r)$. Il peso del link tra v_i e v_j è definito con r_{ij}. Per valutare l'importanza dei termini ed estrarre correttamente keywords da G_t, si definiscono tre pesi per ogni termine: il peso ($tfidf$) ω_t, la proiezione KLT ω_p ed il termine di co-occorrenza ω_r. Il peso ω_t è definito come:

$$\omega_{t_i} = \sum_{j}^{N_d} tf'(t_{ij}) * idf(t_i) \tag{2.20}$$

dove N_d è il numero di video documents e $tf'(t_{ij})$ è la frequenza del termine t_i in D_j. Il peso ω_r è invece definito come:

$$\omega_{r_i} = \sum_{j}^{m} r_{ij}/m \tag{2.21}$$

e considera le correlazioni medie tra v_i e gli m vertici connessi ad esso. Il peso ω_p rappresenta l'energia di ogni termine nella matrice di proiezione KLT. Si costruisce innanzitutto una matrice T di tipo $tfidf$, di dimensione $N_t * N_d$,

dove N_t è il numero di termini. Si computano quindi gli autovalori e gli autovettori di T. I più grandi N_e autovalori ed i loro corrispondenti autovettori $E_k\,(k = 1, 2, ..., N_e)$ sono scelti in modo che gli autovalori possano coprire il 95% dell'energia totale. ω_p è quindi definito come:

$$\omega_{p_i} = \sum_{k}^{N_e} E_{k_i}^2$$

Il peso di ogni vertice in G_t è definito come:

$$W_i = \alpha\omega_{t_i}^{'} + \beta\omega_{p_i}^{'} + (1 - \alpha - \beta)\,\omega_{r_i}^{'}$$

dove $\omega^{'}$indica il peso normalizzato. α e β sono parametri usati per bilanciare ogni parte della formula. Attraverso W_i si possono "potare" i vertici non importanti di G_t, ovvero quelli il cui peso è minore di $mean\,(W_i)$.

Tralasciamo il dettaglio della procedura di clustering e del relativo raffinamento, per non entrare troppo nei particolari ed esulare dagli obiettivi del presente elaborato. L'idea di fondo è quella di effettuare un clustering sull'insieme di *termini* a disposizione, e calcolare una *cosine-similarity* tra il video in questione V_i ed ogni cluster creato. Al termine, ogni video è associato ad un cluster (il più vicino), e quindi ad una serie di termini. Tramite i video di risposta (*response video link*), infine, si attua un rifinimento dei pesi atto a migliorare la rappresentazione del video stesso.

2.4.4 Gli esperimenti

Facciamo un brevissimo resoconto circa la parte sperimentale di questo approccio. I rilievi sono stati effettuati sulla base di un data set estratto da YouTube. Il dataset include 92,640 *video documents*. Per prima cosa, vengono scaricati tutti i video dalla lista dei "Recently Featured", "Most Viewed", "Top Rated",

"Most Discussed", "Today", "This Week", "This Month", "All Time". In secondo luogo, tutti i tag di questi filmati vengono raggruppati insieme, ed i 10 più frequenti sono usati per interrogare YouTube. Per ogni termine query, YouTube fornisce 4 liste di video organizzate per "Relevance", "View Count", "Rating", "Published Time", ognuna delle quali contiene 1000 video. Se i video di ogni lista hanno dei filmati di risposta, vengono anch'essi inclusi nel dataset. I rilievi sperimentali includono valutazioni per i tre diversi contributi dati da questo articolo: *document enhancement*, *topic discovery*, *clustering refinement*.

2.5 Enrichment and Ranking of the YouTube Tag Space and Integration with the Linked Data Cloud

2.5.1 Introduzione

L'incremento del numero di videocamere digitali e di telefoni capaci di fruire contenuto multimediale, ha favorito l'aumento della quantità di *user generatd videos* nel Web. I siti dedicati alla condivisione di materiale multimediale permettono agli utenti di assegnare dei tag ai video archiviati, generando con poco sforzo una serie di annotazioni per il contenuto stesso. Il "tagging" permette alla comunità di organizzare le risorse e presentarle attraverso browser. Il problema consiste nel fatto che questi termini potrebbero essere scelti male e poco correlati tra loro. Le tecniche correnti per ritrovare, integrare e presentare questi contenuti agli utenti, non sono molto svoluppate e necessitano di miglioramenti.

In questo articolo [Choudhury], gli autori descrivono un framework per l'arricchimento semantico, il ranking e l'integrazione dei tag usando le tecnologie del *Semantic Web*. L'accrescimento dello spazio dei tag con la semantica è stato

compiuto attraverso due fasi:

- un'espansione dello spazio dei tag ed uno step di ranking;
- il matching dei concetti e l'integrazione con il *Linked Data Cloud*.

Per arricchire lo spazio dei tag pre-esistente, vengono esplorati i contesti sociali, spaziali e temporali. Lo spazio dei tag semantico risultante è modellato attraverso un grafo locale basato sulle distanze di co-occorrenza, usate ai fini del ranking. Successivamente, una lista di tag pesata viene mappata ed integrata con il Linked Data Cloud attraverso il repository di risorse DBpedia[3].

2.5.2 L'architettura del sistema

In questa sezione, viene data una descrizione dettagliata del sistema di espansione e valutazione dei tag. Si inizia con una vista generale dei differenti moduli, seguita da una spiegazione dello step di filtraggio ed espansione delle annotazioni, dopo di che si descrive il processo di creazione del relativo grafo, per finire con la trattazione dei metodi di tag ranking tramite *spreading activation* sul grafo appena citato. L'intera architettura del sistema è visibile tramite la Figura 2.7.

Lo scopo di questo lavoro è arricchire lo spazio degli *user-generated tag*, classificare e quindi collegare i tag stessi ai concetti DBpedia per una maggiore integrazione con altri dataset. Ci sono tre moduli principali nel sistema, ognuno dei quali è diviso in svariati sottomoduli. La Figura 2.7 mostra il normale flusso di lavoro del sistema: (1) analisi del contesto ed espansione, (2) classificazione dei tag, (3) mapping dei concetti e collegamento col Semantic Web.

[3] http://dbpedia.org

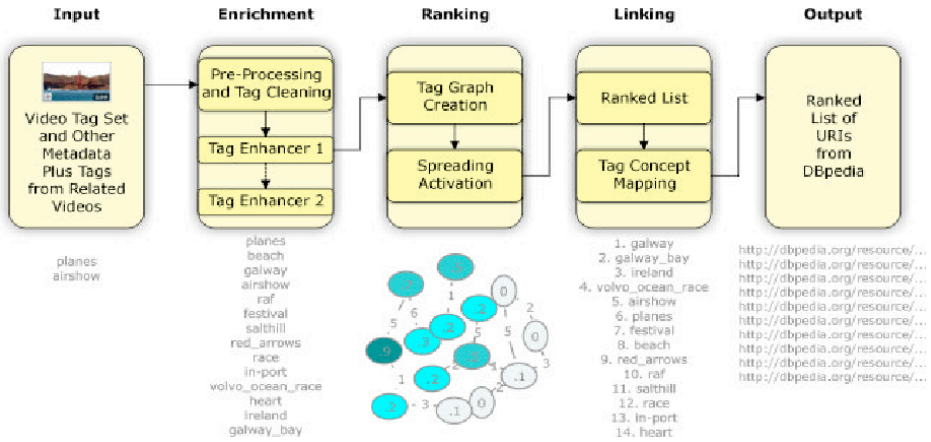

Figura 2.7: Work flow dei processi di accrescimento, ranking e collegamento dei tag.

2.5.2.1 Analisi del contesto ed espansione dei tag

A causa della sparsità dei tag di un video, è necessario effettuare una loro espansione andando a considerare ulteriori contesti: sociali, temporali e geografici. Gli *user-generated-tag* consistono di tre categorie: funzionali (carichi di significato e consistenti in parole singole), rumorosi, composti (o emergenti). I tag composti o emergenti sono quei tag che consistono di due o più keywords spazi bianchi come ad esempio “friendsoftheearth”, “iswc09”, etc. Nel sistema in uso, vengono esclusi i tag di lunghezza inferiore a tre caratteri, i tag soggettivi e quelli non Inglesi. In ogni caso, ci possono essere alcune difficoltà con i tag composti, qualora siano parole non comuni. In questo scenario, infatti, è necessario un lavoro ulteriore per identificare i tag carichi di significato, tra tutti quelli presenti nel set. Il contenuto testuale preso dal titolo del video e dalla descrizione è soggetto allo stesso tipo di pre-processing descritto finora, incluso la rimozione delle stop-words.

2.5.2.2 Arricchimento dello spazio semantico dei tag

Il task consiste nell'arricchimento dello spazio dei tag considerando sorgenti contestuali multiple. Le sorgenti di contesto possono essere di varia natura:

- *testuali*, come titolo e descrizione del video;
- *geospaziali*, come il luogo nel quale il video è stato registrato;
- *temporali*, come il tempo di registrazione del video;
- *sociali*, come gruppi o playlist che includono il video annotato come un'istanza;
- *Related Videos*, ovvero filmati che condividono alcune caratteristiche specifiche come tag, tempo e spazio;
- contesti inerenti l'utente, cioè il tipo di utente che include il video nella lista dei suoi segnalibri o dei preferiti;
- contesti estrapolati dal Web, ovvero siti che portano informazione rispetto ad un tag.

Gli autori, per incrementare lo spazio dei tag, hanno considerato soltanto le prime cinque sorgenti contestuali. Per l'incremento dello spazio dei tag, è stata utilizzata anche la misura di co-occorrenza, nel caso in cui il tag set sia minore di cinque elementi dopo il primo step di espansione. La co-occorrenza utilizzata è di tipo asimmetrico, come specificato in (2.13). Quando si ottiene una lista di tag co-occorrenti per ognuno dei termini della lista di partenza, è necessario un meccanismo per la loro aggregazione. Quest'ultima può essere un semplice meccanismo di voting dove i tag candidati frequenti occupano posizioni elevate della classifica.

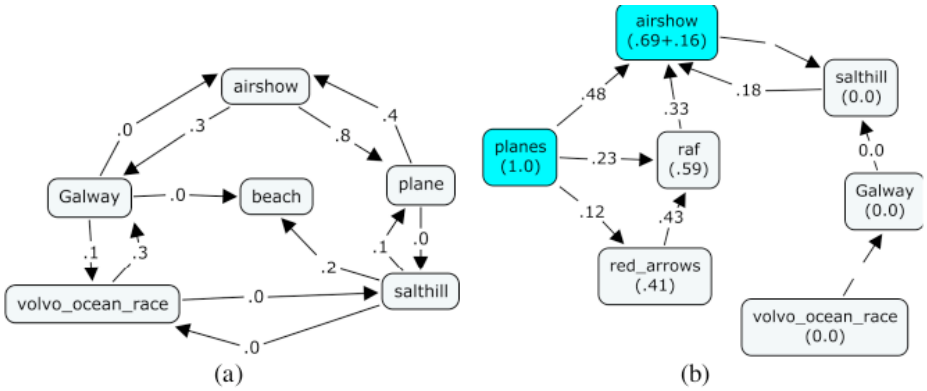

Figura 2.8: (a) Grafo dei tag per un video e (b) Processo di *spreading activation* per il nodo "planes".

2.5.2.3 Tag Ranking

In questa sezione si descrive la classificazione dei tag tramite la creazione di un grafo e l'attuazione di un processo su di esso che porta alla valutazione dei tag stessi. Entriamo nel dettaglio della procedura. Dato un video $v \in V$ ed un set esteso di tag $ET = \{t_1, t_2, ..., t_n\}$, si crea un grafo locale dei tag. Si tratta di un grafo diretto pesato dove i nodi sono proprio i tag ed i collegamenti sono archi pesati. Il peso di un arco dipende da una misura di correlazione asimmetrica basata sulla co-occorrenza. Se il valore di correlazione è minore di una soglia (τ), i tag non sono connessi. Per avere idea di un esempio di grafo, si consideri la Figura 2.8.

La creazione del grafo, e quindi il processo di *spreading activation* su di esso, include i seguenti passi:

1. Ai nodi del grafo viene assegnato il valore iniziale 0 eccetto il nodo di *firing* che assume il valore 1.0.

2. Esso diffonde la sua attivazione a tutti i nodi nelle sue immediate vicinanze, che sono connessi alla sorgente.

3. L'attivazione in uscita è una funzione di $(initialActivation + (initialActivation * edgeStrenght) * d)$ dove d è un fattore di decadimento settato sperimentalmente. Tipicamente è uguale a 0.85.

4. Se il valore del nodo eccede una soglia, si "accende" (fire) nuovamente.

5. Il valore di attivazione del nodo è la somma pesata dei suoi nodi contribuenti.

6. Ogni nodo si attiva una volta sola durante il processo.

Nel momento in cui inizia la propagazione dell'energia, il nodo diffonde quest'ultima ai nodi connessi e l'ammontare di energia ricevuta è una funzione di *strenght* e del fattore d. La Figura 2.8(b) mostra il processo di attivazione che inizia col nodo "plane" e diffonde ai tre nodi "air show", "raf" e "red arrows" (questo nodo contribuisce poi al nodo "air show").

2.5.2.4 La creazione del linked data

Dopo che i tag sono stati classificati, possono essere connessi a risorse similari. In pratica si crea un meccanismo di mapping dagli *user tag* alle risorse ontologiche presenti all'interno dell'universo definito dal *Linked Open Data* (*LOD*). Il LOD consiste attualmente di più di cento dataset e bilioni di entità interlacciate tra loro. Una visione schematica del Linked Open Data è visibile in Figura 2.9.

La questione è come integrare gli *user-generated videos* con l'esistente struttura LOD. L'obiettivo più ambizioso consiste nel mapping automatico tra tag e concetti, che si rivela un compito gravoso a causa dei contesti multipli di un concetto. In questo articolo, viene utilizzato il motore di indirizzamento semantico Sindice[4] per effettuare query a DBpedia e selezionare le entità più appropriate.

[4] http://sindice.com/

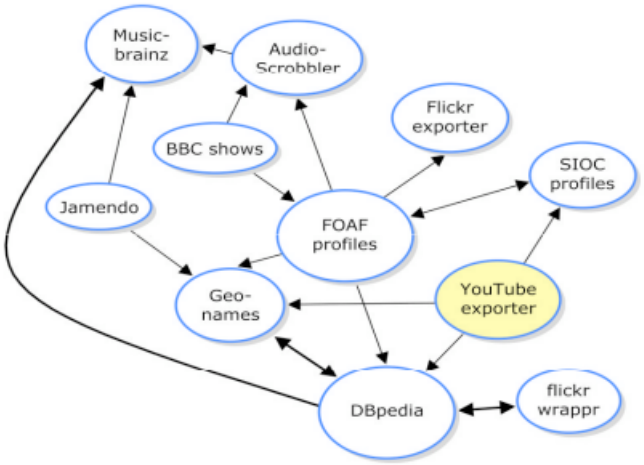

Figura 2.9: I dataset attualmente connessi all'interno del Linked Data Cloud.

2.5.2.5 Il mapping tra tag e concetto

Il matching tag-to-concept non è stato ancora completamente implementato e sarà parte dei prossimi studi in questo ambito. Parte degli esperimenti sono stati fatti usando le risorse DBpedia. Dopo che i tag sono stati finalizzati, si usa un processo in due passi per assegnare loro gli *identificatori di concetto* (*URI*). I tag sono messi in un modulo locale WordNet e vengono seguite alcune semplici euristiche:

1. Se il tag corrisponde ad un nome WordNet, e se esiste solo un synset di matching, viene selezionato il corrispondente URI in DBpedia. Queste operazioni sono rappresentate all'interno della Figura 2.10.

2. Se esiste più di un synset WordNet, si invia il tag ed i suoi termini di contesto ad un modulo di similarità che computa la *cosine similarity* tra quest'ultimi e gli URI (Uniform Resource Identifier) già esistenti.

3. Per quei tag che non fanno parte di WordNet, si utilizza il motore di indirizzamento semantico Sindice per cercare le risorse corrispondenti. Può

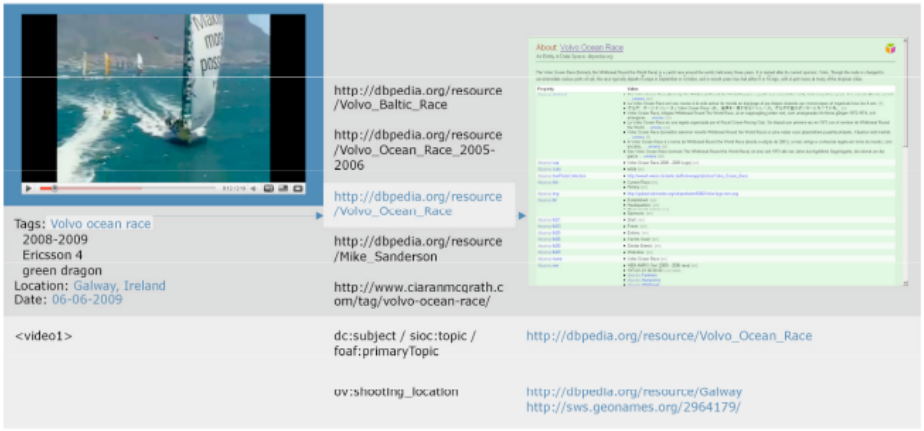

Figura 2.10: Matching dei tag YouTube con le risorse presenti in DBpedia.

essere presente un modulo di disambiguamento che sceglie il miglior URI tra i k disponibili.

2.5.3 Gli esperimenti

Gli autori hanno collezionato 3,990 video da YouTube. Di tutti questi filmati sono stati salvati i metadati, presi attraverso le YouTube API. I video sono stati catalogati in specifiche categorie come “skiing”, “sailing” e “cricket”. I metadati includono tag, date, luoghi (se disponibili), titoli e descrizioni. Il numero di unique tag è maggiore di 11,900 e comprende tag numerici, soggettivi e con scarso significato. Gli utenti distinti presenti nel dataset sono 2,261; in media un utente possiede meno di due video.

Per effettuare le rilevazioni, è stato effettuato un primo filtraggio sui tag eliminando le *stop-words*, i tag con meno di tre caratteri ed quelli numerici. E’ stata quindi fatta una valutazione per misurare la qualità del metodo di classificazione ed arricchimento dei tag. Gli autori hanno selezionato in modo casuale 100 video dal loro dataset; tre utenti sono stati chiamati a valutare la “bontà” del tag più rilevante di ogni video su una scala a quattro livelli:

Figura 2.11: Il numero di volte che il tag più rilevante è stato annotato nelle differenti posizioni.

"most relevant", "relevant", "partially relevant", "irrelevant". Il risultato è un istogramma di frequenza del tag più rilevante nelle quattro posizioni. Questo esperimento permette di capire la robustezza della procedura di tag ranking. I risultati sono evidenziati in Figura 2.11. Dalla figura suddetta si può notare che quasi l'80% delle volte, il *top-ranked tag* viene collocato nelle prime due posizioni.

In modo similare, gli autori hanno valutato l'aspetto relativo all'espanzione delle annotazioni e la sua efficacia nella descrizione del contenuto dei video. Questa valutazione è stata fatta in due momenti: prima dell'espansione e dopo l'espansione. I tre annotatori hanno valutato la lista dei tag (nei due momenti) sulla base di una scala a tre livelli. Il dettaglio di questa procedura è visibile in Figura 2.12. I risultati mostrano come il processo di arricchimento dei tag migliori in modo considerevole la descrizione del contenuto dei video. Nonostante ciò, circa il 28% dei filmati necessita di ulteriori improvements. La cause di tale situazione sono molteplici: necessità di un filtraggio più specifico dei tag, lista dei tag insufficiente, propagazione del rumore da differenti sorgenti contestuali.

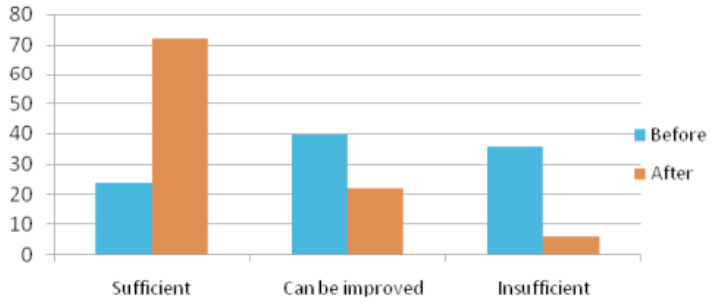

Figura 2.12: Valutazione comparativa della qualità dello spazio dei tag.

2.6 Il contributo fornito dai vari approcci

In questa sezione mettiamo in evidenza i motivi che hanno portato a descrivere gli approcci di cui sopra, che per aspetti diversi riflettono tuttte le problematiche che sono state affrontate in questo lavoro di tesi. A tal proposito, analiziamo singolarmente ogni metodo.

Il metodo [Snoek09] rappresenta la vera e propria base di partenza per il mio lavoro sul Tag Learning. Si tratta infatti dell'articolo di riferimento per questa tematica, in quanto assai recente e scritto da uno dei ricercatori più attivi in questo campo. Questo paper ha introdotto il tema delle annotazioni (tag) per i contenuti multimediali (in questo caso foto), ed ha fornito una base di partenza per la valutazione della loro *rilevanza*. In altre parole, tramite questo articolo, sono nate le prime idee atte a creare un metodo per valutare l'importanza dei singoli tag di un video YouTube. Nel Capitolo 4, a tal proposito, vedremo come l'idea dei *visual neighbors* sia stata adattata al nostro caso.

L'articolo di [VanZowl08] è stato importante per due aspetti. In primo luogo, esso affronta le problematiche già descritte in [Snoek09], ma entra ancor più nel dettaglio dei metodi di calcolo della rilevanza di un tag. In particolare, questo metodo ha suggerito alcune misure di rilevanza (tra le quali la *co-occorrenza* tra tag) che poi sono state adattate ed utilizzate all'interno del nostro approccio.

In secondo luogo, l'articolo affronta la tematica del *tag enrichment*. Come già detto in precedenza, uno degli obiettivi del lavoro di tesi è quello di "arricchire" lo spazio delle annoatazioni di un user-generated video. Con questo articolo, quindi, è stato possibile ragionare su alcune strategie volte ad aumentare il numero di tag dei video che vengono analizzati.

L'approccio [MICC10] ha dato un duplice contributo al presente lavoro di tesi. In primo luogo, ha contribuito ad accrescere la conoscenza per quanto concerne il *tag ranking*. Grazie a questo articolo, infatti, sono nati ulteriori spunti per arrivare a formulare l'algoritmo finale di calcolo della rilevanza dei tag. In secondo luogo, questo metodo ha fornito suggerimenti importanti relativi a come valutare sperimentalmente l'algoritmo proposto. In particolare, l'articolo è stato utile per capire come creare un primo dataset di riferimento. L'utilizzo delle 15 categorie di YouTube è stato importante per differenziare tra loro i video, e al tempo stesso ha permesso di creare risultati sperimentali da confrontare direttamente con quelli già esistenti in altri paper (in questo caso con [MICC10]).

L'articolo descritto nel paragrafo 2.4 ha consentito di acquisire ulteriori conoscenze nell'ambito della rappresentazione dei video e del relativo accrescimento. In altre parole, grazie a questo approccio ho acquisito conoscenze rispetto alla gestione dei filmati da parte di YouTube, ed al tempo stesso capito come sfruttare le potenzialità delle relative API. Il risultato di questo ragionamento è il potenziamento dello spazio dei tag di ogni video analizzato.

Il paper [Choudhury], invece, è stato molto importante per questo lavoro di tesi in quanto ha permesso di entrare all'interno di aspetti non trattati nei precedenti articoli. Quanto scritto in questo paper mi ha consentito di avvicinare i concetti basilari del *Semantic Web*, ed anche capire come poterne sfruttare le potenzialità rispetto agli scopi del presente elaborato. In particolare, questo

paper è stato utile nel processo di affiancamento di informazioni a più alto livello ai tag.

Il risultato è stato il mapping dei tag con le risorse di alcuni repository semantici (es. DBpedia) e quindi la creazione di una vera e propria descrizione testuale del contenuto semantico di ogni video. Questa fase è progettata con lo scopo di accrescere la documentazione di un video, una tematica già trattata in alcuni dei precedenti articoli. Nei prossimi Capitoli e nella parte dedicata ai risultati sperimentali, questi aspetti del lavoro verranno evidenziati ancora più nel dettaglio.

Capitolo 3

Presentazione degli strumenti utilizzati

In questo capitolo ci soffermeremo sulla descrizione degli strumenti che sono stati utilizzati nella sviluppo del metodo descritto in questa tesi. L'obiettivo è quello di presentare al lettore il "contesto operativo" all'interno del quale è stato sviluppato il metodo trattato in questo elaborato.

Faremo riferimento, innanzitutto, al linguaggio di programmazione (Java) ed in particolare alle *Application Programming Interface* (*API*) che ci permettono di interagire con YouTube. Esse rappresentano il punto di partenza di tutto il lavoro; sono fondamentali per estrarre i metadati (titolo, autore, descrizione, tag,...) legati ai video e forniscono uno strumento imprescindibile per lo sviluppo di applicazioni *on line*.

Nella seconda parte, tratteremo con sufficiente dettaglio il concetto di *Semantic Web* e ci soffermeremo sulla descrizione della *Knowledge Base* DBpedia[1]. Questo repository di risorse, come già anticipato nel paragrafo 2.5, fornisce un

[1] http://dbpedia.org/

notevole ausilio nell'ambito del "rinforzamento" della documentazione associata ad un filmato. Il capitolo prosegue con la trattazione dei concetti fondamentali del linguaggio SPARQL, necessario per la scrittura di query verso il repository DBpedia. In questa sezione saranno descritti in forma sintetica gli strumenti sintattici del linguaggio, senza entrare troppo nei particolari in quanto non rientra negli obiettivi di questa tesi.

La parte successiva è dedicata alla trattazione della libreria *Wikipedia Miner*, utilizzata per la gestione delle informazioni strutturate presenti in Wikipedia. Tramite questo toolkit sarà possibile stabilire una misura di *relatedness* tra termini, assolutamente indispensabile, come vedremo in seguito, per i nostri scopi. L'ultima parte del capitolo è dedicata ad una breve descrizione delle API di Flickr[2], il sito web di gestione delle foto che si è deciso di utilizzare per l'ultima parte del lavoro, come premesso e trattato nel Capitolo Introduzione.

3.1 YouTube Java Application Programming Interface

Le YouTube Data API[3] permettono alle client applications di ritrovare e modificare il contenuto YouTube nella forma di *Google Data API feeds*. Ogni applicazione client può utilizzare le API di YouTube per catturare video feeds, commenti, risposte e playlists, effettuando delle query sui video che soddisfano particolari criteri. Un ulteriore potenzialità è quella di fare richieste autenticate atte a modificare queste informazioni e magari eseguire l'upload sul sito di nuovi video. Per poter utilizzare le funzionalità messe a disposizione dalle API di YouTube, è necessario fare il download dei file contenuti all'interno della *Google Data Java Client Library*[4]. Passiamo adesso alla descrizione delle operazioni

[2] http://www.flickr.com/
[3] http://code.google.com/intl/it-IT/apis/youtube/2.0/developers_guide_java.html
[4] http://code.google.com/p/gdata-java-client/downloads/list

che si possono effettuare con questa libreria.

3.1.1 Autenticazione e login per applicazioni residenti

La Java Client Library può essere utilizzata per ritrovare *public feeds* ed eseguire operazioni autenticate. Tutti i public feeds sono read-only e non richiedono alcuna autenticazione. Le operazioni con autenticazione, di contro, includono il ritrovamento di *private feeds* ed in generale la scrittura, l'upload, la modifica e la cancellazione. Per poter eseguire le operazioni autenticate, è necessario ottenere una *developer key* ed un *clientID*[5]. La richiesta di autenticazione può essere fatta in due modi: *AuthSub proxy autentication* o *ClientLogin username/password*. Descriviamo la seconda procedura, che è quella utilizzata in questo lavoro di tesi.

Il primo passo da compiere è l'istanziazione di un oggetto YouTubeService nel modo seguente:

```
YouTubeService service = new YouTubeService(clientID,
    developer_key);
```

A questo punto è possibile effettuare la *ClientLogin Authentication* per la nostra applicazione residente, che tiene traccia dell'indirizzo e-mail dell'utente (username) e della password. Per effettuare l'autenticazione, è sufficiente invocare il metodo setUserCredentials della classe YouTubeService nel modo seguente:

```
service.setUserCredentials("indirizzo_mail", "password");
```

3.1.2 Il prelievo dei metadati associati ai video

In questa sezione ci occuperemo di specificare come è possibile estarre i metadati associati ad un video: titolo, autore, descrizione, tag, etc. Questa operazione è

[5] http://code.google.com/apis/youtube/dashboard/

basilare nel nostro caso, in quanto come sappiamo il compito da assolvere è la classificazione e l'espansione dello spazio dei tag di un video. Molti feeds all'interno delle API di YouTube consistono di *video entries*. Questi feeds possono essere modellati molto semplicemente sotto forma di oggetti VideoFeed, ognuno dei quali contiene un certo numero di oggetti VideoEntry. Ogni *video entry* corrisponde esattamente ad un video presente in YouTube e contiene informazioni circa il video. La riga di codice seguente ritrova un video feed:

```
VideoFeed videoFeed = service.getFeed(new URL(feedUrl),
    VideoFeed.class);
```

Molto spesso può essere utile ricercare uno specifico video presente in YouTube, ovvero una particolare istanza di tipo VideoEntry. Per fare questo, sono necessarie le seguenti righe di codice:

```
String videoEntryUrl = "http://gdata.youtube.com/feeds/
    api/videos/ADos_xW4_J0";
VideoEntry videoEntry = service.getEntry(new URL(
    videoEntryUrl), VideoEntry.class);
```

Nell'esempio fatto, viene ritrovato il video con *videoID = 'ADos_xW4_J0'*. L'ultima parte di un URL, infatti, corrisponde all'identificativo (ID) del video in questione.

A questo punto vediamo come è possibile estrarre i metadati di un video presente in YouTube. Partendo dal video in questione, tramite un'istanza videoEntry, possono essere estratti il *titolo*, l'*autore*, l'*identificativo* (*ID*), la *descrizione* ed i *tag*. Il codice Java che ci permette tale operazione è indicato di seguito:

```
System.out.println("Title: " + videoEntry.getTitle().
    getPlainText());
```

```
YouTubeMediaGroup mediaGroup = videoEntry.getMediaGroup()
    ;
System.out.println("Uploaded by: " + mediaGroup.
    getUploader());
System.out.println("Video ID: " + mediaGroup.getVideoId()
    );
System.out.println("Description: " + mediaGroup.
    getDescription().getPlainTextContent());
MediaKeywords keywords = mediaGroup.getKeywords();
for(String keyword : keywords.getKeywords()) {System.out.
    print(keyword + ",");}
```

3.1.3 La ricerca dei video in YouTube

In questa sezione tratteremo come poter sfruttare le funzionalità delle API di YouTube per effettuare delle query on line e ritrovare i video che soddisfano un certo criterio di ricerca. Tra tutte le operazioni che possono essere fatte tramite la Java Client Library, poniamo l'attenzione solo su quelle che si sono rivelate utili nel presente lavoro di tesi. In primo luogo, vediamo come estrarre, a partire da un videoEntry, la lista dei suoi *Related Videos* (video correlati). Le righe di codice che permettono di eseguire questa operazione sono le seguenti:

```
if (videoEntry.getRelatedVideosLink() != null) {
    String feedUrl = videoEntry.getRelatedVideosLink().
        getHref();
    VideoFeed videoFeed = service.getFeed(new URL(feedUrl
        ), VideoFeed.class);
}
```

A questo punto vediamo in che modo è possibile fare la ricerca dei video in YouTube attraverso le Java API. Quest'ultime permettono al programmatore di richiedere al servizio un set di video che produce una corrispondenza con un certo termine di ricerca. Le API supportano una varietà di *Google Data query parameters* e parametri consuetudinali legati alla ricerca di video. In particolare, si possono ricercare video caricati da un preciso utente oppure video che appartengono ad una certa categoria di YouTube. Per eseguire una richiesta di ricerca, si deve creare un oggetto di tipo YouTubeQuery che specifica il criterio di ricerca e passare tale oggetto al metodo YouTubeService.query. Le righe di codice seguenti mostrano come sviluppare una query per ricercare video che sono legati al termine di ricerca "puppy":

```
YouTubeQuery query = new YouTubeQuery(new URL("http://
    gdata.youtube.com/feeds/api/videos"));
query.setOrderBy(YouTubeQuery.OrderBy.VIEW_COUNT);
query.setFullTextQuery("puppy");
query.setSafeSearch(YouTubeQuery.SafeSearch.NONE);
VideoFeed videoFeed = service.query(query, VideoFeed.
    class);
```

La query sovrastante specifica che i risultati devono essere ordinati in base al numero di visualizzazioni e che il risultato può includere anche del contenuto soggetto a restrizioni. Nell'esempio precedente, la ricerca è stata fatta tramite il parametro identificato da setFullTextQuery. Le API di YouTube includono numerosi altri criteri di ricerca, come visibile nella documentazione disponibile on line[6].

Vediamo adesso in che modo si possono ricercare filmati utilizzando le *categorie* di YouTube e le *keywords* (*tag*). In generale, ogni video può avere associati

[6]http://code.google.com/intl/it-IT/apis/youtube/2.0/developers_guide_java.html

molti tag ma può appartenere ad una sola categoria. Si deve notare che alcune parole, come "comedy" possono rappresentare sia una categoria che una keyword. Per risolvere questa anomalia, le API utilizzano la seguente convenzione: i termini con iniziale maiuscola (es. "Comedy") denotano una categoria mentre quelli con iniziale minuscola (es. "comedy") stanno ad indicare un tag. Il codice di seguito dimostra come ricercare video all'interno della categoria "News" che contengono i tag "sports" e "football":

```
YouTubeQuery query = new YouTubeQuery(new URL("http://
    gdata.youtube.com/feeds/api/videos"));
Query.CategoryFilter categoryFilter1 = new Query.
    CategoryFilter();
Query.CategoryFilter categoryFilter2 = new Query.
    CategoryFilter();
Query.CategoryFilter categoryFilter3 = new Query.
    CategoryFilter();
categoryFilter1.addCategory(new Category(YouTubeNamespace
    .KEYWORD_SCHEME, "sports"));
categoryFilter2.addCategory(new Category(YouTubeNamespace
    .KEYWORD_SCHEME, "football"));
categoryFilter3.addCategory(new Category(YouTubeNamespace
    .CATEGORY_SCHEME, "News"));
query.addCategoryFilter(categoryFilter1);
query.addCategoryFilter(categoryFilter2);
query.addCategoryFilter(categoryFilter3);
VideoFeed videoFeed = service.query(query, VideoFeed.
    class);
```

Se si vogliono ricercare video che contengono soltanto uno dei due tag precedenti ("sports" e "football") e che appartengono alla categoria "News", sarà sufficiente utilizzare solo due oggetti di tipo Query.CategoryFilter, inglobando nel primo entrambi i tag ed associando al secondo la categoria YouTube voluta ("News"). Per una descrizione completa dei modi in cui è possibile scrivere una query, consultare la documentazione delle API già menzionata.

In questa sezione sono state trattate le funzionalità della *Google Data Java Client Library* che si sono rivelate utili al presente lavoro di tesi. Tutte le operazioni descritte troveranno spazio nel prossimo capitolo, quando verrà descritto nel dettaglio il sistema che è stato implementato.

3.2 Il progetto DBpedia

In questa sezione descriveremo la struttura e le potenzialità messe a disposizione dal progetto DBpedia[7]. Questo strumento è di fondamentale importanza nel mio lavoro di tesi in quanto permette di realizzare l'espansione semantica dei video YouTube. Come vedremo nei prossimi paragrafi, infatti, DBpedia permette di manipolare informazioni strutturate presenti in Wikipedia. Grazie a questo aspetto, è possibile incrementare il contenuto semantico-descrittivo di ogni video. Questa parte del lavoro, naturalmente, sarà descritta nel dettaglio nel prossimo capitolo.

Partiremo con una overview del progetto DBpedia, mettendo in evidenza quelli che sono i suoi aspetti architetturali. Nella seconda parte della descrizione, invece, tratteremo le tecniche di interazione con questo framework tramite Java, ed in particolare descriveremo brevemente il linguaggio attraverso il quale è possibile scrivere query: *Sparql*.

[7] http://dbpedia.org/

3.2.1 Introduzione

Il progetto DBpedia [Bizer09] è uno sforzo della comunità scientifica per estrarre informazioni strutturate da Wikipedia e rendere tali informazioni disponibili nel Web. La base di conoscenza risultante descrive più di 2.6 milioni di entità, tra le quali sono incluse 198,000 persone, 328,000 luoghi, 101,000 lavori musicali, 34,000 film e 20,000 compagnie. Per ciascuna di queste entità, DBpedia definisce un identificatore unico globale che può essere dereferenziato nel Web in una ricca descrizione RDF[8] (*Resource Description Framework*) dell'entità. Quest'ultima include definizioni in trenta lingue diverse, relazioni con altre risorse, classificazioni in quattro gerarchie di concetti e collegamenti con altre sorgenti web di dati.

Nell'ultimo anno, un sempre crescente numero di editori ha iniziato a stabilire collegamenti con le risorse DBpedia, rendendo essa il fulcro dell'emergente *Web of Data*. La base di conoscenza DBpedia consiste di 3.1 milioni di collegamenti a pagine web esterne, nonché di 4.9 milioni di collegamenti RDF con altre sorgenti di dati. Essa presenta dei vantaggi rispetto ai repository semantici alternativi: copre molti domini, evolve automaticamente con i cambiamenti di Wikipedia, è multilingua ed è accessibile dal Web.

Il progetto DBpedia, in definitiva, fornisce i seguenti contributi allo sviluppo del *Web of Data*:

- sviluppa un framework di estrazione delle informazioni che converte il contenuto Wikipedia in una base di conoscenza multi-dominio;
- definisce un identificatore dereferenziabile per ogni entità;

[8]Il Resource Description Framework (RDF) è lo strumento base proposto da W3C per la codifica, lo scambio ed il riutilizzo di metadati strutturati. Consente l'interoperabilità tra applicazioni che si scambiano informazioni sul Web. E' costituito da due componenti: *RDF Model and Syntax* e *RDF Schema* (*RDFS*). Il primo espone la struttura del modello RDF, descrivendone una possibile sintassi. Il secondo espone la sintassi per definire schemi e vocabolari per i metadati. Per maggiori informazioni, consultare http://it.wikipedia.org/wiki/Resource_Description_Framework

- stabilisce dei collegamenti RDF verso altre sorgenti di dati e supporta gli editori nello stabilire collegamenti tra le loro fonti e DBpedia.

3.2.2 L'architettura del DBpedia Knowledge Extraction Framework

Gli articoli di Wikipedia consistono per la maggior parte di testo libero, ma contengono anche vari tipi di informazione strutturata nella forma di *wiki markup*. Tale informazione include modelli infobox, categorizzazione di informazione, immagini, coordinate geografiche, collegamenti a pagine web esterne, pagine di disambiguazione, redirects tra pagine e collegamenti attraverso differenti edizioni di Wikipedia. In questa sezione, verrà descritto il framework di estrazione della conoscenza in DBpedia.

Per avere una overview dell'architettura del sistema, si consideri la Figura 3.1. Diamo una breve descrizione dei componenti principali del framework. *PageCollections* è un estrattore delle sorgenti locali o remote degli articoli Wikipedia, *Destinations* memorizza o serializza le triple RDF estratte, *Extractors* trasforma un tipo specifico di wiki markup in triple mentre *Parsers* supporta gli estrattori determinando i tipi di dato, convertendo valori tra unità e scindendo i markup in liste. Il componente *ExtractionJobs*, invece, raggruppa una collezione di pagine, gli estrattori e la destinazione all'interno di un workflow.

Il nucleo di questo framework è l'*Extraction Manager*, che gestisce il processo di passaggio degli articoli Wikipedia agli estrattori e consegna il loro output alla destinazione. Questo componente, inoltre, si occupa della gestione degli URI e risolve i redirects tra gli articoli.

Il framework consiste attualmente di 11 estrattori che processano i seguenti tipi di contenuto Wikipedia:

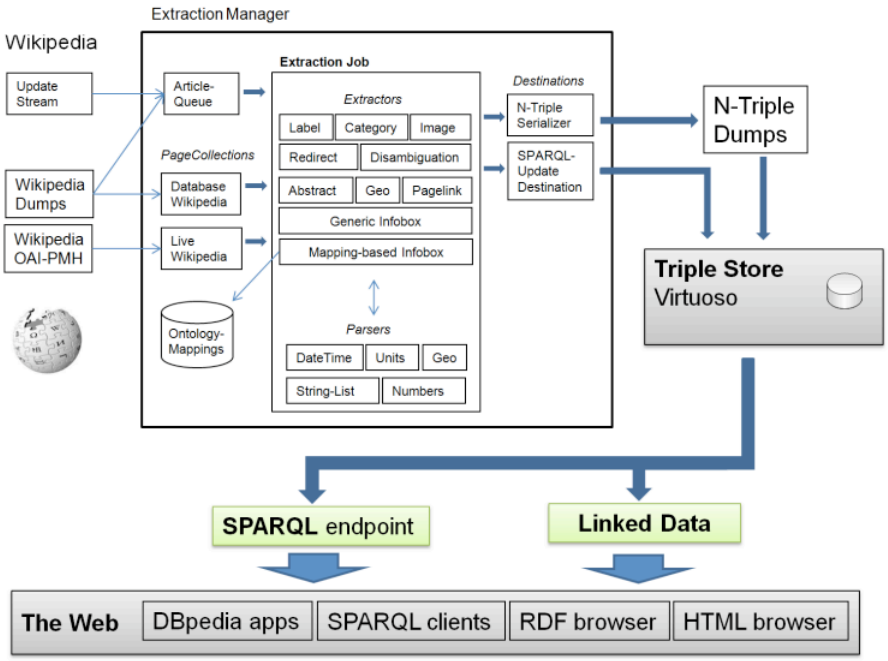

Figura 3.1: Overview dei componenti di DBpedia.

- *Labels.* Tutti gli articoli Wikipedia hanno un titolo, che è visto come un `rdfs:label` per la corrispondente risorsa DBpedia.

- *Abstracts.* Viene estratto un abstract breve (*first paragraph*, rappresentato con `rdfs:comment`) ed un abstract lungo (al massimo 500 parole, usando la proprietà `dbpedia:abstract`) da ogni articolo.

- *Interlanguage links.* Si estraggono collegamenti che connettono gli articoli dello stesso argomento a differenti edizioni di Wikipedia e si utilizzano per assegnare labels e abstracts in linguaggi diversi alle risorse DBpedia.

- *Images.* Vengono estratti collegamenti che puntano ad immagini Wikipedia raffiguanti una risorsa. I collegamenti sono rappresentati tramite la proprietà `foaf:depiction`[9].

- *Redirects.* Per identificare termini sinonimi, gli articoli di Wikipedia possono essere reindirizzati verso altri articoli. Si estraggono questi redirects e si utilizzano per risolvere le referenze tra le risorse DBpedia.

- *Disambiguation.* Le pagine di disambiguamento di Wikipedia spiegano le differenze di significato dei termini omonimi. Si estraggono e si rappresentano collegamenti di disambiguamento usando il predicato `dbpedia:disambiguates`.

- *External links.* Gli articoli contengono referenze a risorse web esterne che si rappresentano usando la proprietà DBpedia `dbpedia:reference`.

- *Pagelinks.* Si estraggono tutti i links tra gli articoli Wikipedia rappresentandoli tramite la proprietà `dbpedia:wikilink`.

[9] Con `foaf` (Friend of a Friend) ci riferiamo ad un dataset che assieme a DBpedia contribuisce a formare il *Web of Data*. Nei prossimi paragrafi verrà data maggiore chiarezza su questo aspetto. Per il momento, limitiamoci ad evidenziare che si tratta di una tecnologia decentralizzata volta alla descrizione di persone ed alla connessione di *social web sites*.

- *Homepages*. Questo estrattore ottiene i collegamenti verso le homepage di entità come compagnie ed organizzazioni cercando i termini *homepage* o *website* all'interno dei collegamenti (le homepage sono rappresentate con `foaf:homepage`).

- *Categories*. Gli articoli Wikipedia sono organizzati in categorie, rappresentate tramite il vocabolario SKOS[10]. Le categorie diventano `skos:category`; le relazioni tra categorie sono rappresentate usando `skos:broader`.

- *Geo-coordinates*. L'estrattore geografico esprime le coordinate usando il vocabolario Basic Geo[11] (WGS84 lat/long) e la codifica GeoRSS Simple[12] del vocabolario W3C Geospatial. Il primo esprime la latitudine e la longitudine come fatti distinti, permettendo di attuare una sorta di filtraggio nelle query SPARQL.

3.2.3 La base di conoscenza DBpedia

La base di conoscenza DBpedia consiste attualmente di 274 milioni di triple RDF, estratte da 35 diverse versioni di Wikipedia. Essa descrive più di 2.6 milioni di entità ed è dotata di etichette e brevi abstracts in 30 lingue diverse. Dispone di 609,000 collegamenti ad immagini, 3,150,000 collegamenti a pagine web esterne, 415,000 categorie Wikipedia e 286,000 categorie YAGO.

La tabella in Figura 3.2 fornisce una vista delle comuni classi DBpedia, e mostra il numero di istanze nonché alcune proprietà di esempio per ogni classe. Nel seguito della descrizione, si spiega come vengono costruiti gli identificatori e si comparano i quattro schemi di classificazione messi a disposizione da DBpedia.

[10] http://www.w3.org/2004/02/skos/
[11] http://www.w3.org/2003/01/geo/
[12] http://www.w3.org/2005/Incubator/geo/XGR-geo/

Ontology Class	Instances	Example Properties
Person	198,056	name, birthdate, birthplace, employer, spouse
Artist	54,262	activeyears, awards, occupation, genre
Actor	26,009	academyaward, goldenglobeaward, activeyears
MusicalArtist	19,535	genre, instrument, label, voiceType
Athlete	74,832	currentTeam, currentPosition, currentNumber
Politician	12,874	predecessor, successor, party
Place	247,507	lat, long
Building	23,304	architect, location, openingdate, style
Airport	7,971	location, owner, IATA, lat, long
Bridge	1,420	crosses, mainspan, openingdate, length
Skyscraper	2,028	developer, engineer, height, architect, cost
PopulatedPlace	181,847	foundingdate, language, area, population
River	10,797	sourceMountain, length, mouth, maxDepth
Organisation	91,275	location, foundationdate, keyperson
Band	14,952	currentMembers, foundation, homeTown, label
Company	20,173	industry, products, netincome, revenue
Educ.Institution	21,052	dean, director, graduates, staff, students
Work	189,620	author, genre, language
Book	15,677	isbn, publisher, pages, author, mediatype
Film	34,680	director, producer, starring, budget, released
MusicalWork	101,985	runtime, artist, label, producer
Album	74,055	artist, label, genre, runtime, producer, cover
Single	24,597	album, format, releaseDate, band, runtime
Software	5,652	developer, language, platform, license
TelevisionShow	10,169	network, producer, episodenumber, theme

Figura 3.2: Le comuni classi di DBpedia, col numero di istanze ed alcune proprietà di esempio.

3.2.3.1 L'identificazione delle entità

Per la creazione degli identificatori, DBpedia utilizza i nomi degli articoli in lingua Inglese. L'informazione proveniente da altre versioni di Wikipedia viene mappata in questi identificatori valutando in modo bidirezionale i collegamenti inter-linguaggio tra gli articoli di Wikipedia. Alle risorse vengono assegnati degli identificatori (URI) in accordo al pattern `http://dbpedia.org/resource/`*`Name`*, dove *`Name`* è preso dall'URL dell'articolo Wikipedia di provenienza, che ha la forma `http://en.wikipedia.org/wiki/`*`Name`*. Questo produce alcune benefici certi:

- gli URI coprono un range largo di topic enciclopedici;
- sono definiti dal consenso della comunità;
- esistono politiche chiare per la loro gestione;
- è disponibile una definizione testuale ampia dell'entità ad una locazione web ben conosciuta (la pagina Wikipedia).

3.2.3.2 La classificazione delle entità

Le entità DBpedia sono classificate all'interno di quattro schemi allo scopo di soddisfare i requisiti delle differenti applicazioni. Come primo schema abbiamo le **Wikipedia Categories**. DBpedia contiene una rappresentazione SKOS di categorie di Wikipedia. Il sistema di categorie consiste di 415,000 categorie. Il vantaggio principale di questo schema consiste nell'essere esteso ed aggiornato in modo collaborativo da migliaia di editori di Wikipedia. Uno svantaggio, invece, è che le categorie non formano una corretta gerarchia di attualità.

Il secondo schema consiste in **YAGO**. Esso contiene 286,000 classi che formano una vera e propria gerarchia. Questo schema fu creato mappando le categorie foglie di Wikipedia (quelle che non hanno sottocategorie) nei synset

di WordNet[13]. Le caratteristiche della gerarchia YAGO sono la sua profondità e la codifica di molta informazione in una sola classe.

Il terzo schema è **UMBEL**. La sigla sta per *Upper Mapping and Binding Exchange Layer*[14]; si tratta di un'ontologia leggera creata per l'interlacciamento del contenuto web e dei dati. Questa ontologia è stata derivata da *OpenCyc* e consiste di 20,000 classi. Le classi *OpenCyc* sono a loro volta derivate dalle collezioni *Cyc*, basate sui synset di WordNet.

Il quarto schema è quello che ci interessa maggiormente in quanto è la rappresentazione utilizzata nel presente lavoro di tesi. Si tratta della **DBpedia Ontology**. L'ontologia di DBpedia consiste di 170 classi che formano una gerarchia superficiale ed include 720 proprietà con definizioni di dominio e range. L'ontologia fu creata manualmente dai templates più comunemente usati all'interno dell'edizione inglese di Wikipedia. La colonna di sinistra nella tabella di Figura 3.2 mostra una parte della gerarchia di classe dell'ontologia DBpedia. Ulteriori informazioni sulle classi ontologiche saranno fornite nel prossimo capitolo, dove verrà descritto il sistema implementato.

3.2.3.3 La descrizione delle entità

Ogni entità in DBpedia è descritta tramite un set di proprietà generali (*general properties*) ed un set di proprietà specifiche (*infobox specific properties*), se il corrispondente articolo Inglese di Wikipedia contiene una infobox. Le proprietà generali includono un'etichetta (label), un breve ed un lungo abstract in lingua Inglese, un link all'articolo Wikipedia corrispondente, le coordinate geografiche (se disponibili), un link ad un'immagine raffigurante l'entità, collegamenti a pagine web esterne e ad entità DBpedia correlate. Se un'entità esiste in più

[13]http://wordnet.princeton.edu/
[14]http://www.umbel.org/

versioni di Wikipedia, i due abstract (breve e lungo) all'interno di ogni lingua e collegati alle restanti versioni vengono aggiunti alla descrizione.

Le proprietà specifiche (*infobox specific properties*) sono definite all'interno del namespace `http://dbpedia.org/property/` oppure in alternativa in `http://dbpedia.org/ontology/` . La differenza tra i due namespace riguarda essenzialmente i due differenti approcci di estrazione in DBpedia. Per non dilungarsi troppo nella trattazione, lasciamo questi aspetti ad approfondimenti facoltativi del lettore. In questo lavoro di tesi, comunque, si fa riferimento al secondo namespace.

3.2.4 L'accesso alla DBpedia Knowledge Base tramite Web

DBpedia è presente sul Web sotto i termini della *GNU Free Documentation Licence*. Allo scopo di adempiere i requisiti delle diverse applicazioni client, i realizzatori forniscono l'accesso alla DBpedia Konowledge Base attraverso quattro meccanismi:

- **Linked Data**. E' un metodo di pubblicazione dei dati RDF nel Web che si affida agli URI HTTP come identificatori di risorse ed al protocollo HTTP per le relative descrizioni. Gli identificatori di risorse sono impostati per ritornare descrizioni RDF (quando sono acceduti da *Semantic Web Agents*) oppure una vista HTML verso i tradizionali web browser.

- **SPARQL Endpoint**. Le applicazioni client possono inviare query rispettando il protocollo SPARQL verso l'endpoint presente all'indirizzo `http://dbpedia.org/sparql` . In aggiunta allo standard SPARQL, questo endpoint supporta diverse estensioni del linguaggio per le query che sono utili per lo sviluppo di client applications: tra queste ricordiamo la

ricerca testuale sui predicati RDF e le funzioni aggregate (es. COUNT()). Lo SPARQL Endpoint è gestito utilizzando *Virtuoso Universal Server*[15].

- **RDF Dumps**. La DBpedia Knowledge Base è stata divisa in più parti e resa disponibile come download, sotto forma di *N-Triple serialisations*, al sito[16] web di DBpedia.

- **Lookup Index**. E'stato creato per facilitare la connessione tra i dati pubblicati dagli editori e gli URI di DBpedia. Si tratta di una Web Service che combina la similarità tra stringhe ad una classifica di rilevanza, al fine di trovare la corrispondenza DBpedia corretta per un certo termine.

Nel presente lavoro di tesi, la connessione alla base di conoscenza di DBpedia è stata effettuata tramite SPARQL Endpoint. Nel prossimo capitolo verrà dato naturalmente ampio spazio alla trattazione di questo aspetto.

3.3 Gli strumenti utilizzati per le interrogazioni a DBpedia

In questa sezione introduciamo due strumenti che si sono resi necessari per la scrittura di interrogazioni verso la *DBpedia Knowledge Base*. In primo luogo descriveremo, in forma sintetica, le caratteristiche fondamentali del linguaggio **SPARQL**, attraverso il quale è possibile scrivere le query appena menzionate. Tratteremo soltanto gli aspetti sintattici fondamentali del linguaggio; per una descrizione più approfondita (che esula dagli obiettivi di questo lavoro), si può consultare la pagina web del linguaggio SPARQL[17]. Nella seconda parte di questa sezione, invece, elencheremo gli aspetti principali del **Jena Toolkit**[18],

[15] http://virtuoso.openlinksw.com
[16] http://wiki.dbpedia.org/Downloads32
[17] http://www.w3.org/TR/rdf-sparql-query/
[18] http://jena.sourceforge.net/index.html

un *Semantic Web Framework* per Java attraverso il quale si possono inserire le interrogazioni all'interno delle client applications.

3.3.1 Il linguaggio SPARQL

SPARQL, linguaggio di interrogazione per RDF, asceso ormai da tempo al rango di *W3C Candidate Recommendation*, è stato accolto entusiasticamente come l'agognato ultimo tassello per l'edificazione del web semantico. Nel seguito verranno introdotte, senza pretese di esaustività, le caratteristiche essenziali di questo linguaggio.

Prima di entrare nel dettaglio del linguaggio, è utile riflettere su un aspetto: per quale motivo si è creato un linguaggio ad hoc per le interrogazioni verso RDF quanto già esistevano altri linguaggi, come SQL e XQuery? In sintesi, gli scopi di SQL e di SPARQL sono abbastanza diversi tra loro da giustificare la creazione di un linguaggio specifico ex-novo. I due linguaggi sono comunque legati, in quanto è possibile tradurre espressioni SPARQL in espressioni SQL. Tale peculiarità permette agli sviluppatori di immagazzinare i propri dati RDF in database relazionali e di scrivere le query, a seconda dei casi, usando SQL oppure SPARQL.

A questo punto introduciamo gli aspetti sintattivi del linguaggio. SPARQL adotta la sintassi *Turtle*, un'estensione di *N-Triples*, alternativa estremamente sintetica ed intuitiva al tradizionale RDF/XML. Si considerino le seguenti triple RDF, utilizzate nel seguito come punto di partenza per le query d'esempio:

```
@prefix cd: <http://example.org/cd/>
@prefix: <http://example.org/esempio/>
:Permutation cd:autore "Amon Tobin".
:Bricolage cd:autore "Amon Tobin".
:Amber cd:autore "Autechre".
```

```
:Amber cd:anno 1994.
```

Le asserzioni sono espresse in concise sequenze soggetto-predicato-oggetto e delimitate da un punto finale. @prefix introduce prefissi e namespace; i due punti senza prefisso (seconda riga) definiscono il namespace di default. Gli URI sono inclusi tra parentesi uncitate. I letterali di tipo stringa sono contrassegnati da virgolette.

Le query SPARQL si basano sul meccanismo del "pattern matching" e in particolare su un costrutto, il "triple pattern", che ricalca la configurazione a triple delle asserzioni RDF fornendo un modello flessibile per la ricerca delle corrispondenze. Si consideri a questo proposito la seguente riga di esempio:

```
?titolo cd:autore ?autore.
```

In luogo del soggetto e dell'oggetto questo "triple pattern" prevede due variabili, contrassegnate con `?`. Le variabili funzionano in un certo senso da incognite dell'interrogazione, `cd:autore` ha invece il ruolo di costante: le triple RDF che trovano riscontro nel modello associeranno i propri termini alle variabili corrispondenti. Per chiarire, ecco una semplice query di selezione SPARQL:

```
PREFIX cd: <http://example.org/cd/>
SELECT ?titolo ?autore ?anno
FROM <http://cd.com/listacd.ttl>
WHERE {?titolo cd:autore ?autore.
       ?titolo cd:anno ?ann .
}
```

Come si può facilmente notare, l'analogia con il linguaggio SQL è lampante. Identifichiamo, a questo punto, il significato di ogni singola riga della query di cui sopra. `PREFIX` dichiara prefissi e namespace, `SELECT` definisce le variabili di ricerca da prendere in considerazione nel risultato (nell'e-

titolo	autore	anno
"Amber"	"Autechre"	1994

Figura 3.3: Risultato delle query SPARQL di esempio.

sempio: `titolo`, `autore` ed `anno`), `FROM` specifica il set di dati su cui dovrà operare la query (si suppone che le triple siano immagazzinate presso l'indirizzo fittizio `http://cd.com/listacd.ttl`). È inoltre possibile ricorrere alla clausola `FROM NAMED` e alla parola chiave `GRAPH` per specificare più set di dati: per dettagli circa questa utile funzionalità, si vedano le sezioni "7. RDF Dataset" e "8. Querying Dataset" della specifica presente all'indirizzo web `http://www.w3.org/TR/rdf-sparql-query/` .

La clausola `WHERE` , infine, definisce il criterio di selezione specificando tra parentesi graffe uno o più *triple patterns* separati da punto fermo. Applicando la query al set di triple precedente, si ottiene il risultato in Figura 3.3.

Vediamo adesso come è possibile effettuare delle ricerche per opzioni ed alternative. La query precedente, infatti, ha catturato esclusivamente le triple dotate di tutti e tre i termini richiesti (`titolo`, `autore`, `anno`), escludendo quelle che ne possedevano soltanto due (`titolo`, `autore`). È possibile riformulare la query in modo più elastico, prevedendo l'eventuale assenza di alcuni termini; a tal proposito si consideri la porzione di codice seguente:

```
PREFIX cd: <http://example.org/cd/>
SELECT ?titolo ?autore ?anno
FROM <http://cd.com/listacd.ttl>
WHERE {?titolo cd:autore ?autore.
       OPTIONAL {?titolo cd:anno ?anno}
}
```

titolo	autore	anno
"Permutation"	"Amon Tobin"	
"Bricolage"	"Amon Tobin"	
"Amber"	"Autechre"	1994

Figura 3.4: Il risultato della seconda query di esempio.

Nell'esempio, il secondo pattern è dichiarato opzionale: l'informazione è aggiunta al risultato solo se disponibile, altrimenti le variabili compariranno prive di valore (unbound). Il risultato della query è visibile in Figura 3.4. Un altro modo per assicurare una certa elasticità nel reperimento dei dati è il seguente:

```
PREFIX cd: <http://example.org/cd/>
SELECT ?titolo ?autore ?anno
FROM <http://cd.com/listacd.ttl>
WHERE{
    {?titolo cd:autore ?autore}
    UNION
    {?titolo cd:anno ?anno}
}
```

La parola chiave `UNION` esprime un OR logico: la query non si limita pertanto alle triple che soddisfano entrambi i triple patterns, ma cattura sia quelle che soddisfano il primo, sia quelle che soddisfano il secondo. Tramite il linguaggio SPARQL, inoltre, è possibile stabilire delle restrizioni sui valori da associare alle variabili. Si consideri a questo proposito il seguente esempio:

```
PREFIX cd: <http://example.org/cd/>
SELECT ?titolo ?anno
FROM <http://cd.com/listacd.ttl>
```

```
WHERE {?titolo cd:anno ?anno.
       FILTER (?anno > 2000).
}
```

In questo caso, la restrizione è effettuata mediante l'operatore di confronto `>`: il filtro esclude i termini che non soddisfano la condizione definita tra le parentesi tonde: il risultato in questo caso è nullo (il dataset di riferimento non prevede valori maggiori di `2000` per la proprietà anno). Per l'elenco completo degli operatori supportati da SPARQL si veda la sezione "11.3 Operator Mapping" della specifica descritta nell'apposito sito web[19].

Nella sezione successiva, "11.4 Operators Definitions", sono descritti gli operatori specifici del linguaggio: tra questi, trattiamo solatanto *regexp*, valido corrispettivo dei criteri di ricerca LIKE dell'SQL. Esso permette di adoperare espressioni regolari per il matching dei letterali. Come verrà descritto nel prossimo capitolo, questo costrutto ha una forte importanza nel contesto del matching tra tag e risorse DBpedia. Si consideri il seguente esempio:

```
PREFIX cd: <http://example.org/cd/>
SELECT ?titolo ?autore
FROM   <http://cd.com/listacd.ttl>
WHERE  { ?titolo cd:autore ?autore .
         FILTER regex(?autore, '^au', 'i')
}
```

Il filtro seleziona, senza riguardo per maiuscole o minuscole, solo gli autori che iniziano per "au". Il risultato di questa query è mostrato in Figura 3.5. Come in SQL, è possibile escludere dal risultato i valori duplicati mediante la parola chiave `DISTINCT`, ad esempio:

```
SELECT DISTINCT ?titolo ?autore
```

[19] http://www.w3.org/TR/rdf-sparql-query/

titolo	autore
"Amber"	"Autechre"

Figura 3.5: Il risultato della terza query di esempio.

Altri costrutti supportati da SPARQL per la manipolazione del risultato sono:

```
ORDER BY DESC(?autore)
LIMIT 10
OFFSET 10
```

L'espressione `ORDER BY` imposta l'ordine dei risultati della query: stando all'esempio, quest'ultimi verranno presentati in ordine decrescente (`DESC`) in base alla variabile `?autore`. `LIMIT` pone restrizioni al numero dei risultati: nell'esempio soprastante si prendono soltanto i primi `10`. La clausola `OFFSET` permette invece di escludere un certo numero di risultati, scartando, stando all'esempio, i primi `10`. Per ulteriori approfondimenti relativi al linguaggio di interrogazione SPARQL, si faccia riferimento, come già anticipato, al sito web `http://www.w3.org/TR/rdf-sparql-query/` .

3.3.2 Il Jena Toolkit per la scrittura di query SPARQL in Java

Il Resource Description Framework (RDF) è uno standard per la descrizione delle risorse. Con risorsa si intende tutto quello che si può identificare: una pagina web, un tutorial, una persona, etc. Supponiamo di voler fare un esempio relativo alle persone. In questo caso si utilizza la rappresentazione RDF di VCARDS[20], basata su nodi e diagrammi ad archi. Si consideri l'esempio mostrato in Figura

[20] http://www.w3.org/TR/vcard-rdf/

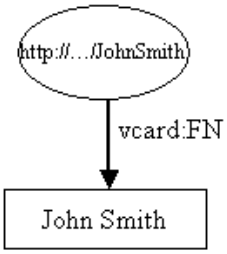

Figura 3.6: Esempio di semplice *vcard*.

3.6. La risorsa *John Smith* è identificata con un'ellisse, contenente il suo *Uniforme Resource Identifier* (*URI*). Le risorse sono dotate di proprietà: nel caso di esempio viene mostrata solo la proprietà *Full Name* (*FN*). Il nome di una proprietà è a sua volta un URI; la parte precedente ai ":" è chiamata namespace prefix e rappresenta un namespace. La parte successiva ai ":" è chiamata local name e rappresenta un nome in un namespace.

Ogni proprietà ha un valore; nel caso specifico si tratta di un valore di tipo *literal* (presente all'interno di un rettangolo) che può essere pensato come una stringa di caratteri. Il toolkit Jena è una Java API da utilizzare nella creazione e nella manipolazione di grafi RDF come quello mostrato nell'esempio. Jena dispone di classi per rappresentare *grafi*, *risorse*, *proprietà* e *literals*.

Il codice per creare il grafo di Figura 3.6 è il seguente:

```
static String personURI = "http://somewhere/JohnSmith";
static String fullName = "John Smith";
Model model = ModelFactory.createDefaultModel();
Resource johnSmith = model.createResource(personURI);
johnSmith.addProperty(VCARD.FN, fullName);
```

Ogni arco in un modello RDF è chiamato *statement*. Ogni statement asserisce un fatto circa una risorsa ed è composto da tre parti:

- *subject*: è la risorsa dalla quale l'arco parte;

- *predicate*: è la proprietà che etichetta l'arco;
- *object*: è la risorsa o il literal puntato dall'arco.

Uno statement è chiamato anche tripla, in quanto composto da tre parti. Supponendo di voler leggere gli statements che compongono un modello, si consideri il seguente codice:

```
StmtIterator iter = model.listStatements();
while (iter.hasNext()) {
    Statement stmt = iter.nextStatement();
    Resource subject = stmt.getSubject();
    Property predicate = stmt.getPredicate();
    RDFNode object = stmt.getObject();
    System.out.print(subject.toString());
    System.out.print(" " + predicate.toString() + " ");
    if (object instanceof Resource) {
        System.out.print(object.toString());
    } else {
        System.out.print(" \"" + object.toString() +
            "\"");
    }
    System.out.println(" .");
}
```

Dal momento che l'oggetto di uno statement può essere sia una risorsa che un literal, il metodo `getObject()` ritorna un oggetto di tipo `RDFNode`, ovvero una comune superclasse di `Resource` e `Literal`. Il risultato prodotto dall'esempio è mostrato in Figura 3.7. Per concludere questa sezione, vediamo come poter effettuare una "connessione semantica" tra Jena e Sparql, tramite la creazione di query Sparql. Come primo passo si crea il modello ontologico di riferimento,

```
http://somewhere/JohnSmith http://www.w3.org/2001/vcard-rdf/3.0#N anon:14df86:ecc3dee17b:-7fff .
anon:14df86:ecc3dee17b:-7fff http://www.w3.org/2001/vcard-rdf/3.0#Family  "Smith" .
anon:14df86:ecc3dee17b:-7fff http://www.w3.org/2001/vcard-rdf/3.0#Given  "John" .
http://somewhere/JohnSmith http://www.w3.org/2001/vcard-rdf/3.0#FN  "John Smith" .
```

Figura 3.7: Output del listato Java di esempio. Ogni riga consiste di tre campi: `soggetto`, `predicato` ed `oggetto`.

partendo dall'ontologia costruita in precedenza e salvata in locale. Il codice a tal proposito è il seguente:

```
public static void main(String[] args) {
      OntModel model = ModelFactory.createOntologyModel(
          OntModelSpec.OWL_MEM, null);
model.read("file:src/ontologies/OSDoap.owl");
```

A questo punto, si scive la query SPARQL che richiede la lista di tutti i progetti Open Source e per ognuno di essi la relativa Maturity Note:

```
String queryString = "PREFIX doap:     <http://usefulinc.
    com/ns/doap#>"+
    "PREFIX osDoap:  <http://imolinfo.it/ontologies/osdoap
        #>" +
    "PREFIX rdfs:     <http://www.w3.org/2000/01/rdf-schema
        #> "+
    "SELECT ?name ?note_comment "+
    "WHERE { "  +
    "     ?proj a osDoap:OpenSourceProject ." +
    "     ?proj doap:name ?name ." +
    "     ?proj osDoap:hasMaturityNote ?note ." +
    "     ?note rdfs:comment ?note_comment" +
    "}";
```

Ovviamente scriviamo la query SPARQL tramite il `QueryFactory` e la pas-

siamo a un `QueryExecution` insieme al modello che rappresenterà il dataset di default per la query:

```
Query query = QueryFactory.create(queryString);
QueryExecution qexec = QueryExecutionFactory.create(query
    , model);
```

A questo punto si esegue la query tramite il metodo `execSelect()` e si effettua un'iterazione sui risultati:

```
try {
ResultSet results = qexec.execSelect();
    System.out.println("Elenco dei risultati della query
        SPARQL: ");
    System.out.println
        ("=============================================");
    while (results.hasNext()) {
        QuerySolution querySolution = (QuerySolution)
            results.next();
        Literal nameLiteral = querySolution.getLiteral("
            name");
        Literal noteLiteral = querySolution.getLiteral("
            note_comment");
    System.out.println("Progetto Open Source: "+
        nameLiteral.getString()
          +" ==> Maturity Note: "+noteLiteral.getString()
              );
    }
    } finally {
        qexec.close();
```

```
    }
}
```

L'oggetto `QuerySolution` che otteniamo permette di carpire il valore di una variabile come `Literal`, `Resource` o anche come `RDFNode`. Nel nostro caso i valori sono dei `Literal` e come tali li salviamo nelle due variabili `nomeLiteral` e `noteLiteral`. Per gli scopi di questo lavoro di tesi, è sufficiente terminare qui la trattazione riguardo al toolkit Jena. Eventuali approfondimenti sono disponibili all'indirizzo web `http://jena.sourceforge.net/`.

3.4 Il toolkit Wikipedia Miner

L'enciclopedia online Wikipedia è un vasto repository di informazioni e rappresenta per i ricercatori e gli sviluppatori un database multilingua di concetti e relazioni semantiche. L'obiettivo di questa sezione è la descrizione del toolkit **Wikipedia Miner** (WM); l'articolo scientifico a cui si fa riferimento è [Milne08]. Come prima presentazione, possiamo dire che si tratta di una collezione open source di codice che permette agli sviluppatori di integrare la semantica di Wikipedia all'interno delle loro applicazioni. Il toolkit Wikipedia Miner è ormai un prodotto maturo, che fornisce l'accesso alla struttura ed al contenuto di Wikipedia tramite un approccio object-oriented.

Il problema che affronta WM è l'estrazione di informazione utile da Wikipedia, in modo scalabile e tempestivo. La versione Inglese corrente di Wikipedia include approssimativamente sei milioni di pagine; le sue caratteristiche semantiche utili sono racchiuse in 20 GB di markup criptati.

3.4.1 La modellazione di Wikipedia

In questa sezione vengono descritte alcune delle classi più importanti per la modellazione della struttura e del contenuto di Wikipedia. In particolare, si spiega come esse corrispondono agli elementi di un dizionario tradizionale. Le classi sono mostrate in Figura 3.8, contenente la loro gerarchia nonché le proprietà ed i metodi. Descriviamo nel dettaglio le singole parti che compongono la modellazione di Wikipedia:

Pages. Tutto il contenuto di Wikipedia è presentato sotto forma di pagine. Il toolkit WM modella ogni pagina con un identificatore unico (id), un titolo e del contenuto sotto forma di markup MediaWiki.

Articles. Forniscono la maggior parte del contenuto informativo di Wikipedia. Ogni articolo descrive un singolo concetto o argomento; i loro titoli sono succinti, ben formati e possono essere utilizzati all'interno di ontologie e dizionari. Il toolkit WM fornisce due definizioni per ogni articolo: una breve ed una media. Quest'ultime consistono rispettivamente nella prima frase ed nel primo paragrafo del contenuto dell'articolo. Quando un certo articolo è stato identificato, possono essere estratti i concetti correlati in base agli articoli a cui esso punta, oppure a quelli che vi sono collegati. Sfortunatamente, molti di questi collegamenti non corrispondono a relazioni semantiche; è perciò assai difficile separare i collegamenti utili da quelli irrilevanti. Nel seguito della trattazione, vedremo come WM affronta questo problema. Un'informazione utile è il numero di articoli che sono collegati ad un articolo di riferimento: i concetti poco conosciuti, infatti, hanno pochi collegamenti.

Redirects. Sono pagine il cui unico scopo è connettere un articolo a titoli

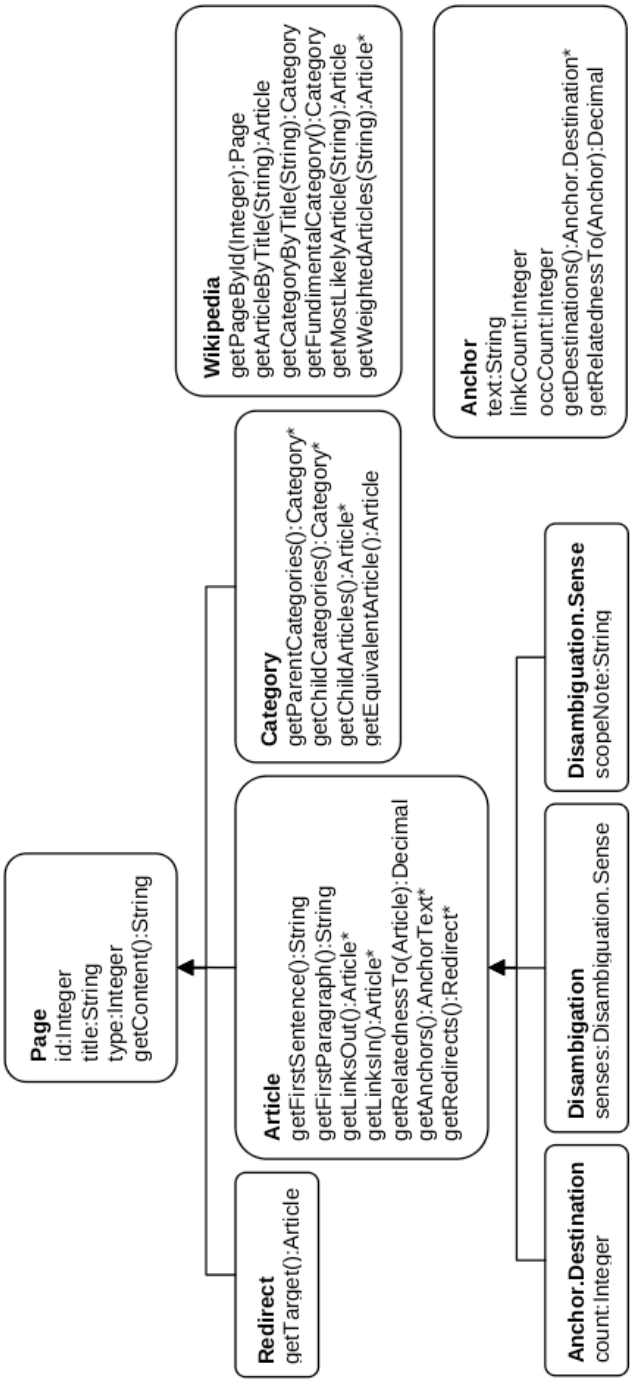

Figura 3.8: Un esempio di classi, proprietà e metodi disponibili nel toolkit Wikipedia-Miner.

alternativi. I redirects possono anche rappresentare argomenti più especifici che non giustificano articoli separati.

Categories. Quasi tutti gli articoli di Wikipedia sono organizzati all'interno di una o più categorie, che possono essere estratte per hyponims, holonyms ed argomenti più generali. Se un argomento è sufficientemente ampio, l'articolo centrale potrebbe essere accoppiato con una categoria dello stesso nome: l'articolo *dog*, ad esempio, è accoppiato alla categoria *dogs*. Tutte le categorie di Wikipedia discendono da una singola radice chiamata *Fundamental*.

Disambiguation. Quando a più articoli viene assegnato lo stesso nome, un tipo specifico di articolo, il disambiguation, è usato per separarli. Ognuna di queste pagine dispone di una nota esemplificativa addizionale: una piccola frase che spiega perché è differente da altri potenziali significati.

Anchors. Rappresentano sinonimi ed altre variazioni di forma per un certo articolo. Gli anchors codificano inoltre la polisemia; il termine *dog*, ad esempio, è utilizzato per collegarsi ad articoli differenti quali animali domestici, segni zodiacali o icone americane di fast food. Nel seguito della trattazione, vedremo che questa parte del toolkit avrà un ruolo fondamentale nell'algoritmo di espansione dei tag progettato nel mio lavoro di tesi.

Wikipedia. E' esso stesso uno degli oggetti più importanti da modellare. Fornisce un punto centrale di accesso alla maggior parte delle funzionalità di Wikipedia Miner.

3.4.2 La ricerca in Wikipedia

Il toolkit Wikipedia Miner indicizza le pagine così che queste possano essere ricercate efficientemente. Lo scenario più comune nella ricerca è restituire un articolo o un set di articoli che si riferiscono al termine dato. Quando viene ricercato ad esempio il termine *dog*, il risultato atteso è il set di articoli a cui potrebbe essere ragionevolmente dato quel titolo. Un approccio comune in letteratura è cercare rispetto ai titoli delle pagine e risolvere i diversi tipi di pagine seguendo i collegamenti da esse implicate. In accordo a questo schema, i *matching articles* sono usati direttamente, i redirects sono risolti verso i loro articoli target, e le pagine di disambiguamento sono estratte per i differenti sensi.

In pratica, Wikipedia non è molto machine-readable: le pagine di disambiguamento ed i titoli non sono facilmente processabili in modalità automatica.

3.4.3 La computazione della correlazione semantica

Il toolkit Wikipedia Miner include algoritmi per la generazione di misure di parentela semantica, che quantificano il grado di relazione tra i concetti. In accordo al toolkit, ad esempio, *dog* è relazionato al 100% con *canis familiaris*, al 47% con *domestic animal* ed al 19% con *animal*. Queste misure hanno una vasta gamma di applicazioni, come il language processing, il knowledge management ed il data mining.

Le misure di *relatedness* fatte da Wikipedia Miner vengono generate usato gli hyperlinks tra gli articoli di Wikipedia. Quest'ultimi sono collegati ampiamente tra loro ed a prima vista i collegamenti portano con sé interessanti relazioni semantiche. In generale, però, non sempre i collegamenti tra gli articoli possono essere tradotti in relazioni "forti". Per separare le relazioni utili da quelle irrilevanti, WM attua una metrica di valutazione ben definita. I dettagli di questa procedura sono presenti in [Milne-Witten08a].

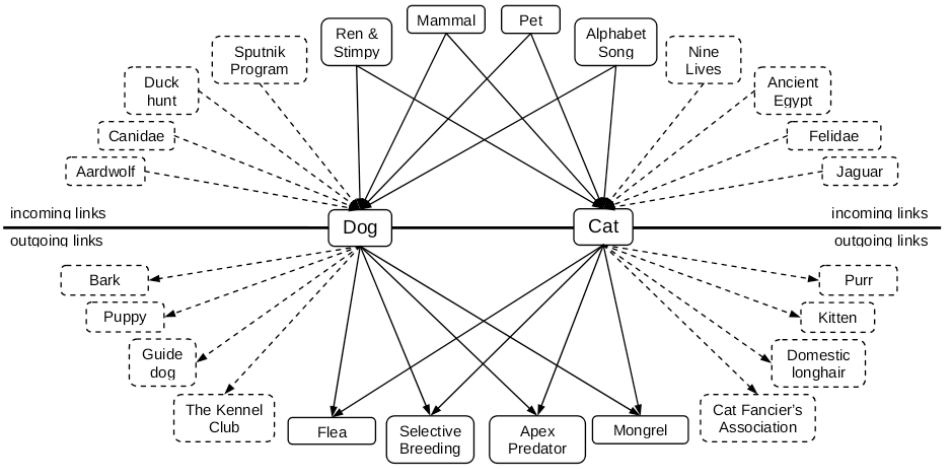

Figura 3.9: La misura di relatedness semantica tra *Dog* e *Cat* tramite i collegamenti di Wikipedia.

La Figura 3.9 fornisce una vista approssimativa di come il toolkit lavora nel caso dei due termini di esempio. Se si considerano i link creati da questi articoli, mostrati nella metà inferiore della figura, si notano alcune sovrapposizioni che ci fanno capire come i due concetti siano tra loro correlati. Nella figura si notano anche collegamenti "unici" verso alcune entità (*puppy*, *bark*, *kitten*, *purr*,...), ad indicare che esse non sono correlate. Lo stesso procedimento può essere applicato ai collegamenti che esistono verso ognuno dei due articoli (metà superiore di Figura 3.9). Dopo un'operazione di normalizzazione, la proporzione di sovrapposizione e non-sovrapposizione dei link fornisce una misura di *relatedness* pari al 77%.

La misura di parentela semantica descritta finora riguarda i concetti: è possibile comparare quindi solo un articolo con un altro articolo. In genere, può essere utile mettere in relazione termini o frasi generiche; WM supporta questa funzionalità ed utilizza a tal proposito gli *anchors*, descritti in precedenza. Per fare questa operazione WM disambigua gli anchors scegliendo un particolare

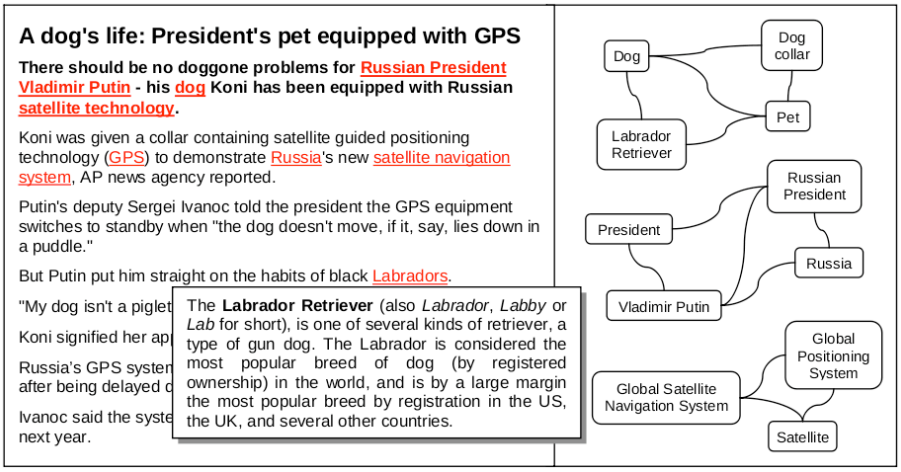

Figura 3.10: Un articolo di *news* esteso con argomenti (topics) Wikipedia.

articolo (o senso) per rappresentare ogni anchor. I dettagli di questa procedura sono presenti in [Milne-Witten08a].

3.4.4 Apprendimento ed estensione dei documenti

Per ogni documento, è probabile che Wikipedia abbia delle conoscenze circa gli argomenti discussi al suo interno e sia in grado di aggiungervi informazione. La Figura 3.10 mostra un breve articolo di news nel quale possono essere identificati alcuni topics Wikipedia. Gli argomenti rilevati possono migliorare la modellazione del documento e magari essere appresi da sistemi automatici. Quest'ultimi, tipicamente, rappresentano i documenti come *bag of words*; consultando Wikipedia si possono rintracciare quindi i concetti che queste parole rappresentano. Questo passo è indicato nella parte destra della Figura 3.10, che mostra alcuni dei concetti rilevati e le loro relazioni.

La sfida di questa parte è rilevare questi topics e creare i links appropriati accuaratamente ed efficientemente. Il toolkit fornisce algoritmi comprovati per fare questo. Questo processo coinvolge tre passi: *candidate selection* (raccolta

dei termini e delle frasi per il linking), *disambiguation* (decidere la destinazione di un link) e *detection* (decidere se un link deve essere stabilito con tutti). Ulteriori dettagli di questa fase, che non rientrano negli obiettivi della trattazione, possono essere trovati in [Milne-Witten08b].

3.5 Le API di Flickr

L'ultima parte di questo lavoro di tesi consiste nell'estrazione di foto dal sito Flickr[21], utili al lavoro svolto da [Smeraldo10] per quanto concerne la localizzazione dei tag negli shot di un video. In questa sezione vogliamo dare una breve descrizione di **flickrj**[22], le API Java attraverso le quali è possibile interagire con Flickr all'interno di un'applicazione client.

Le Flickr API consistono di un insieme di metodi e di alcuni endpoint. Per effettuare un'operazione con Flickr API, è necessario selezionare una convenzione di chiamata ed inviare una richiesta al suo endpoint specificando un metodo ed alcuni argomenti. Il risultato consiste in una risposta formattata. Per poter utilizzare le funzionalità delle API di Flickr, è necessario ottenere una chiave di accesso (*apiKey*), contattando direttamente gli sviluppatori.

Vediamo in che modo è possibile salvare su disco immagini prese da questa sorgente di foto, tramite istruzioni Java all'interno di una comune applicazione client. Per prima cosa è necessario istanziare un oggetto della classe Flickr:

```
Flickr flickr = new Flickr(apiKey, secret, rest);
```

dove `apiKey` indica la chiave menzionata in precedenza, `secret` è una stringa alfanumerica e `rest` un oggetto di tipo `REST` gestito nel modo seguente:

```
REST rest = new REST();
rest.setHost(svr);
```

[21] http://www.flickr.com/
[22] http://flickrj.sourceforge.net/

A questo punto devono essere settati i parametri di ricerca per definire il criterio di estrazione delle foto. Le righe di codice che adempiono a questa funzione sono le seguenti:

```
SearchParameters searchParams = new SearchParameters();
searchParams.setSafeSearch(Flickr.SAFETYLEVEL_SAFE);
searchParams.setSort(SearchParameters.RELEVANCE);
```

Per stabilire in che modo vengono scaricate le immagini, si utilizza l'ultima delle tre righe sovrastanti. Essa stabilisce che l'ordinamento delle foto deve essere fatto sulla base della RELEVANCE, ovvero sulla "rilevanza" dell'immagine. Altri criteri di ordinamento includono la data in cui la foto è stata inserita in Flickr nonché la sua "interessantezza" (interestingness).

La ricerca delle immagini viene effettuata sulla base dei "termini di ricerca", creati per lo più tramite la composizione dei tag iniziali e grazie all'espansione realizzata con Wikipedia Miner. L'istruzione che consente di settare questi termini all'interno dei parametri di ricerca è la seguente:

```
searchParams.setText("termine_di_ricerca");
```

Per estrarre realmente le foto, si utilizza un oggetto della classe `photosInterface`:

```
photoList = photosInterface.search(searchParams, maxFoto,
    1);
```

La gestione dell'immagine con *flickrj* necessita di utilizzare un oggetto della classe `Photo`, attraverso il quale è possibile, come vedremo, il salvataggio di un'immagine all'interno di un file (ad esempio *.jpg*). Il codice Java a tal proposito è il seguente:

```
Photo photo = (Photo) photoList.get(z);
photo = photosInterface.getPhoto(photo.getId());
```

Come anticipato, è il momento di stabilire in che modo è possibile creare un file con la foto appena estratta da Flickr:

```
BufferedImage im = photosInterface.getImage(photo, Size.
    SMALL);
File outputFile = new File("./"+dir+"/image0"+contaFoto
    +".jpg");
ImageIO.write(im, "jpg", outputFile);
```

La foto scaricata viene salvata in un oggetto di tipo `BufferedImage`; per creare fisicamente il file con l'immagine viene quindi utilizzato il metodo `write` della classe `ImageIO`. Possiamo concludere qui l'introduzione alle API di Flickr, visto che abbiamo già trattato gli aspetti legiat a questo lavoro di tesi. Ulteriori approfondimenti in tal senso sono possibili consultando la pagina web di *flickrj*.

Capitolo 4

Il metodo implementato

In questo capitolo descriviamo nel dettaglio la parte "operativa" del presente lavoro di tesi, rispetto agli obiettivi presentati nel Capitolo Introduzione. Verrà data notevole importanza ad ogni aspetto del software realizzato, per fare una descrizione puntuale delle idee che di volta in volta sono state implementate. Come già anticipato nel Capitolo Introduzione, l'ambiente di sviluppo scelto consiste in *Eclipse*, all'interno del Sistema Operativo *Ubuntu 10.04 LTS*[1]. Il linguaggio di programmazione utilizzato è Java; nel modulo di interrogazione al framework DBpedia è stato inoltre utilizzato il linguaggio SPARQL[2].

L'esposizione è organizzata in sotto-sezioni, al fine di analizzare modulo per modulo l'approccio creato. All'interno di questo capitolo non verrà fatto riferimento alle basi teoriche degli strumenti utilizzati, in quanto per tale funzione è presente il Capitolo 3. Ci concentreremo sulla programmazione vera e propria, spiegando le idee che motivano ogni parte dell'algoritmo. Per una maggiore chiarezza, inoltre, in alcuni casi saranno inserite delle porzioni di codice sorgente Java.

[1] http://www.ubuntu.com/
[2] http://www.w3.org/TR/rdf-sparql-query/

Per quanto concerne i risultati sperimentali, si è deciso di scrivere un capitolo a parte. Questo ci consente di rendere l'esposizione più strutturata ed al tempo stesso analizzare meglio le prestazioni rispetto ai due obiettivi del lavoro: il *filtraggio* (con relativo ranking) dei tag e la loro *espansione* (integrazione col Semantic Web). Questi due task sono il nucleo di fondo dei prossimi paragrafi, e verranno perciò trattati nei minimi dettagli.

Prima di entrare nel dettaglio delle singole sezioni, riassumiamo le linee guida dell'intero progetto. L'idea di base del metodo consiste nella progettazione di una procedura di classificazione dei tag associati ad un video di YouTube. Ad ogni etichetta sarà associato un punteggio, al fine di esprimerne la "rilevanza". I tag ritenuti validi saranno poi utilizzati per la fase di espansione, dove sarà raggiunto un duplice obiettivo:

- l'estrazione di contenuto semantico da affiancare alla descrizione dei filmati, già presente tra i metadati di ogni video residente in YouTube;
- l'incremento del numero di tag associati ad ogni filmato.

4.1 Le strutture dati

Per iniziare la descrizione della parte operativa, è utile focalizzarsi in primo luogo sulle strutture dati che sono state create. Come già menzionato nel Capitolo 3, il software sviluppato consiste in una *Java Client Application*. Sono state perciò definite delle classi che permettono di adempiere le funzioni volute. La modellazione del codice sorgente è stata fatta definendo tre classi: *Video.java*, *queryVideo.java*, *Categoria.java*. Di seguito vengono elencate le principali caratteristiche di ognuna di esse.

Video.java è la classe che ci permette di gestire le informazioni associate ad ogni video di YouTube che viene analizzato. Questa classe è utilizzata per la

memorizzazione dei metadati (*titolo*, *autore*, *ID, descrizione*, *tag*,...) associati ad ogni video e delle altre informazioni di interesse.

Video.java memorizza il set iniziale dei tag, i loro punteggi (il grado di "bontà" dell'etichetta), il set finale dei tag stessi (ottenuto al termine del processo di espansione) e molte altre variabili di interesse che saranno meglio descritte nelle prossime sezioni. Possiamo affermare che questa classe è stata creata per manipolare ogni singola informazione di un video analizzato, e per calcolare di volta in volta i parametri che lo caratterizzano.

queryVideo.java è la classe che contiene il metodo principale *"main"*. Utilizza le funzionalità messe a disposizione dalla classe descritta in precedenza e permette all'utente di avviare l'analisi di un video di interesse. Con questa struttura dati viene effettuata la connessione a YouTube per mezzo delle relative API (descritte nel Capitolo 3) e vengono effettuate le interrogazioni on-line al repository DBpedia (anch'esso descritto nel Capitolo 3). Rappresenta il cuore dell'intero software in quanto controlla sia la parte relativa alla validazione dei tag sia quella inerente l'integrazione con il Semantic Web.

Fanno parte della classe *queryVideo.java* alcune funzioni membro che permettono l'archiviazione su disco dei metadati di ogni video nonché la loro manipolazione (e quindi l'utilizzo ai fini della classificazione e dell'espansione dei tag). Nelle prossime sezioni, ogni funzione membro sarà descritta nel dettaglio.

Categoria.java è la classe che permette la mappatura delle ontologie di DBpedia all'interno delle *categorie* di YouTube. Questo procedimento sarà descritto nel dettaglio nel seguito di questo capitolo; per il momento inquadriamo brevemente le idee che vi stanno alla base. Come è noto, ogni video di YouTube appartiene ad una categoria (non è possibile che un filmato sia legato a più di una categoria) tra tutte quelle disponibili.

Le categorie di YouTube sono in totale 15: *Autos & Vehicles*, *Comedy, Edu-*

cation, Entertainment, Film & Animation, Gaming, Howto & Style, Music, News & Politics, *Nonprofits & Activism*, *People & Blogs*, *Pets & Animals*, *Science & Technology*, *Travel & Events*, *Sports*. L'idea di fondo è quella di stabilire, per ognuna di esse, quali sono le classi ontologiche di interesse. Questo concetto verrà approfondito naturalmente nelle prossime sezioni, dove risulterà più chiaro.

Con questa prima sezione è stata fatta una panoramica iniziale circa l'organizzazione del codice sorgente. A questo punto siamo pronti per affrontare la descrizione sequenziale delle operazioni svolte nel metodo implementato.

Il resto del capitolo è organizzato in blocchi, che rispecchiano la sequenza ordinata delle operazioni svolte dal software sviluppato nel corso del lavoro di tesi.

4.2 L'algoritmo di filtraggio ed espansione dei tag

In questa sezione descriveremo nel dettaglio l'algoritmo sviluppato in questo lavoro di tesi. Come già anticipato nel capitolo precedente, si ha la necessità di interagire in modalità *on line* con i sito web YouTube[3]. Per poter fare questo è necessario utilizzare le funzionalità messe a disposizione dalle *Google Data API* di YouTube ed in particolar modo dalla *Java Client Library*.

La prima operazione svolta dal software consiste nell'autenticazione verso il servizio messo a disposizione dalle API di YouTube. Per fare ciò è sufficiente utilizzare i metodi della classe `YouTubeQuery`, come anticipato nel paragrafo 3.1.1. Nel caso in cui la fase di *login* vada a buon fine, viene inviata verso YouTube la query che consente di estrarre una lista di video (`videoFeed`) sulla base dei parametri di ricerca impostati. In realtà, infatti, l'autenticazione non è proprio la prima operazione svolta dal programma. Prima di effettuare il login vengono

[3] http://www.youtube.com/

richieste tre informazioni: la categoria che dovrebbe essere associata ai video cercati (vedremo in seguito come questa potrà essere modificata dall'algoritmo) e due *parametri* di ricerca. Il primo consiste nella stringa di ricerca (i termini chiave attraverso i quali ricercare il filmato) mentre il secondo rappresenta semplicemente il numero massimo di video da ricercare.

Nel momento in cui la lista di video è disponibile, vengono memorizzati i suoi **metadati** in un file testo residente su disco. Le informazioni che vengono estratte sono facilmente visibili nella parte bassa della Figura 4.1. La parte che più ci interessa riguarda i tag del video; essi sono le etichette testuali evidenziate con un'ellisse. Dall'alto verso il basso della figura, inoltre, possiamo riconoscere le altre informazioni del video: il titolo, l'autore, la descrizione e la categoria di YouTube.

Nel file di testo contenente i metadati, la scrittura delle informazioni segue una formattazione ben precisa. In primo luogo, tra tutte le informazioni restituite dalle API di YouTube, si è scelto di registrare nel file soltanto sei campi: *titolo*, *autore*, *videoID*, *descrizione*, *Web URL*, *tag*. Ognuna delle informazioni prededenti è registrata in una riga del file; tra un campo e l'altro è inserito un carriage-return. In secondo luogo, tra i metadati di un video e quelli di un altro è presente una linea bianca, stante ad indicare il termine delle informazioni utili. Per meglio chiarire la formattazione del file, si consideri la Figura 4.2.

Dopo la registrazione dei metadati nel file di testo, è possibile andare a riempire le variabili membro della classe `Video` con le stringhe ivi memorizzate. La scelta di archiviare in un file *.txt* le informazioni testuali deriva da pure ragioni di comodità: dopo che è stata specificata la stringa di ricerca per la query, è possibile controllare in tempo reale se i video estratti corrispondono a quelli cercati (nel caso in cui venga fatta una ricerca "mirata"). I sei campi registrati nel file testo vengono trasferiti quindi all'interno di altrettante variabili

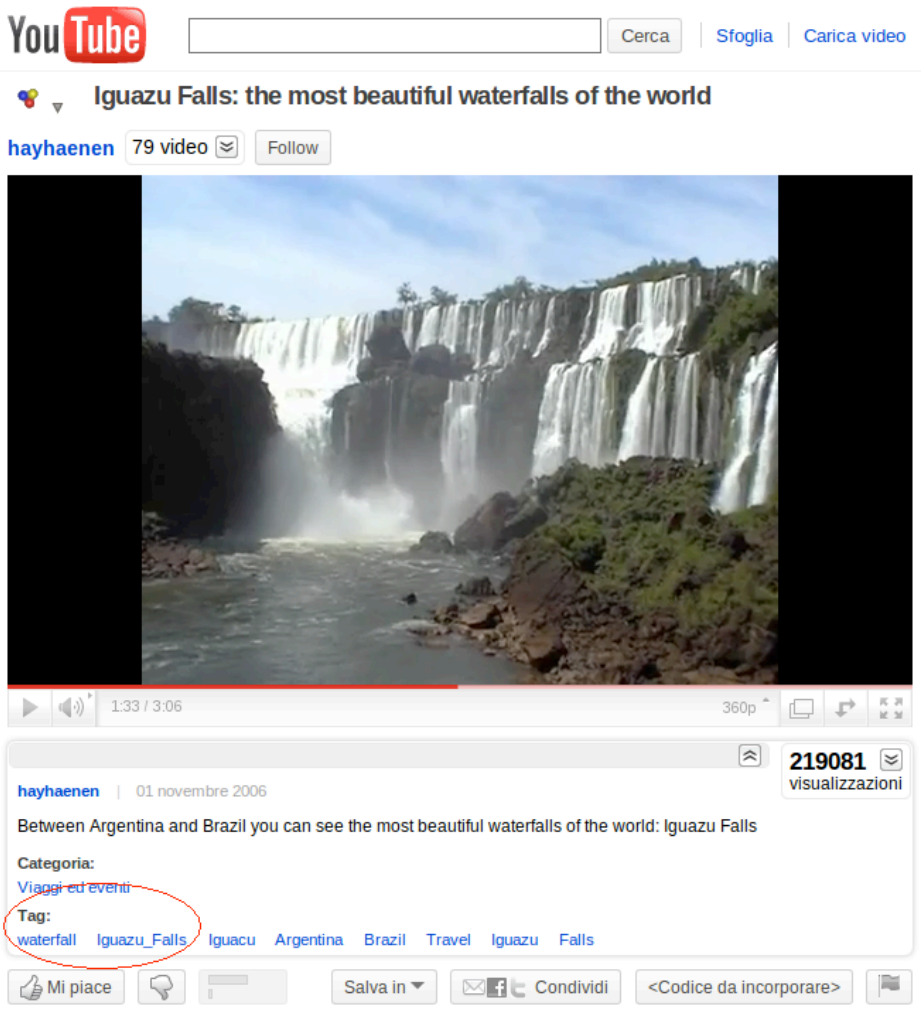

Figura 4.1: I *metadati* associati ad un video YouTube di esempio.

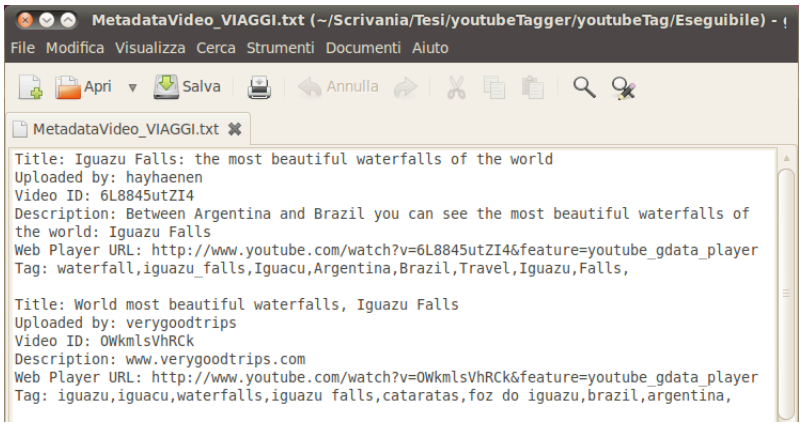

Figura 4.2: Screenshot del file testo con i metadati di due video di esempio.

membro, per essere disponibili nel seguito dell'elaborazione. Durante il *parsing* del file, inoltre, viene preparata la stringa con il nome del file testo all'interno del quale sarà inserita l'espansione dei tag tramite DBpedia. Questa procedura, naturalmente, sarà descritta nel dettaglio nelle prossime sezioni.

A questo punto possiamo passare alla prossima operazione svolta dall'algoritmo. Si tratta dell'estrazione dei *Related Videos* (video correlati) di tutti i filmati carpiti tramite la query menzionata prima, nonchè della gestione della loro *YouTube Category* (la categoria YouTube del video inserita dall'autore). Nel prossimo paragrafo daremo i dettagli rispetto al concetto dei "video correlati" e tratteremo le problematiche associate alla "categoria".

4.2.1 L'estrazione dei Related Video e la gestione della YouTube Category

Nel Capitolo Introduzione, abbiamo chiarito come il primo obiettivo di questo lavoro di tesi coincida con il *filtraggio* ed il *ranking* dei tag associati ad un video di YouTube. Come mostreremo meglio nel seguito, la validazione dei tag avviene facendo riferimento ai *Related Videos*, ovvero ai video correlati a quello

che stiamo correntemente analizzando. Il concetto di video correlato necessita di alcune puntualizzazioni. Innanzitutto, partiamo col dire che i *Related Videos* possono essere estratti attraverso una funzionalità delle API di YouTube.

L'algoritmo con il quale YouTube trova i video correlati è una vera e propria "scatola nera"; non esiste documentazione ufficiale che spieghi in che modo viene effettuata la ricerca. Facendo alcuni esempi on line, comunque, appare evidente come l'algoritmo di ricerca si basi sulle informazioni testuali del video corrente ed in particolar modo sui suoi tag. Un altro aspetto che appare evidente è il seguente: per ogni filmato di partenza, le API di YouTube restituiscono sempre un numero di correlati compreso tra 20 e 25. L'importanza dei Related Videos apparirà più chiara nelle prossime sezioni, quando entreremo nel dettaglio della procedura di filtraggio e ranking delle etichette.

Dopo ever fatto questa breve digressione, torniamo alle operazioni svolte dal software sviluppato in questo lavoro di tesi. Per estrarre i video correlati, si crea un URL di struttura fissa, al quale viene aggiunto l'identificativo unico (ID) del video stesso; le righe di codice corrispondenti sono le seguenti:

```
String videoEntryUrl = "http://gdata.youtube.com/feeds/
    api/videos/";
videoEntryUrl = videoEntryUrl.concat(this.videoID);
```

La funzione che permette di trovare i correlati a partire da un URL è la `getRelatedVideoLinks` della classe `VideoEntry`. Con le prossime tre istruzioni:

```
VideoEntry videoEntry = service.getEntry(new URL(
    videoEntryUrl), VideoEntry.class);
String feedUrl = videoEntry.getRelatedVideosLink().
    getHref();
VideoFeed videoFeed = service.getFeed(new URL(feedUrl),
    VideoFeed.class);
```

si arriva ad ottenere la lista di video (`videoFeed`) correlati con il video di partenza. L'archiviazione dei metadati dei *Related Videos* avviene in modo analogo a quanto fatto nella fase di ricerca dei video di partenza. Viene infatti creato un file testo all'interno del quale sono memorizzate le sei informazioni di interesse: *titolo*, *autore*, *ID*, *descrizione*, *Web URL*, *tag*. La formattazione del file con i correlati segue le stesse regole descritte in precedenza ed evidenziate nella Figura 4.2.

Passiamo ora alla gestione della *YouTube Category* alla quale appartiene il filmato correntemente analizzato. A questo proposito è utile fare subito un'importante precisazione. Come già anticipato, un video residente in YouTube è legato ad una ed una sola categoria, scelta dall'autore al momento della pubblicazione. Accade spesse volte che la categoria al quale è associato il video non sia proprio quella corretta. Volendo fare un esempio, è frequente il caso in cui filmati raffiguranti aerei che atterrano siano catalogati come "Travel & Events" piuttosto che come "Auto & Vehicles". Questa incongruenza si traduce in una problematica da gestire accuratamente, per quelle che sono le operazioni svolte dal nostro software.

L'aspetto appena menzionato verrà chiarito nel dettaglio quando tratteremo la parte relativa all'espansione semantica dei tag; per il momento pensiamo solamente che una categoria YouTube errata comporta uno scarso risultato nella fase di integrazione dei tag con il Semantic Web.

In base a quanto affermato, è perciò necessario definire una procedura accurata di gestione della *YouTube Category*. Inizialmente, la categoria del video corrente coincide con quella stabilita dall'autore del filmato che stiamo analizzando. Il passo successivo consiste in una procedura di validazione o cambiamento della categoria stessa; vediamo nel dettaglio come viene effettuato tale controllo. Come primo passo, si estrae la YouTube Category di ognuno dei Re-

lated Videos e la si memorizza all'interno di un vettore di stringhe. Il passo successivo consiste nella creazione di un'istogramma delle categorie. In pratica, si associa ad ognuna delle 15 categorie di YouTube il relativo numero di occorrenze all'interno dell'insieme dei video correlati.

La prossima operazione consiste nella determinazione del bin più elevato dell'istogramma, ovvero della categoria che presenta la frequenza più alta nei Related Videos. Assieme alla categoria si estrae anche l'altezza del bin (numero di occorrenze) e si calcola il rapporto:

$$RapportoDiOccorrenza = maxAltezza/\#RelatedVideos \tag{4.1}$$

dove con # intendiamo il numero di filmati correlati. Se il rapporto in (4.1) è maggiore di una *threshold* (tipicamente 0.5), la categoria del video corrente viene modificata nel caso in cui non coincida con quella collegata al bin più elevato dell'istogramma. Nel caso in cui, invece, il rapporto in (4.1) sia minore della soglia, si mantiene la YouTube Category impostata dall'autore del filmato. A questo punto, possiamo passare alla descrizione dello step successivo svolto dal software, che consiste nella strategia di filtraggio e pesatura dei tag.

4.2.2 La procedura di filtraggio e ranking dei tag

In questa sezione descriveremo nel dettaglio la strategia attuata per la validazione e la pesatura degli *user-defined tag*. Al termine di questo procedimento, l'algoritmo sviluppato in questo lavoro di tesi attribuisce un punteggio ad ogni etichetta inserita dall'autore del filmato. Come conseguenza, quindi, soltanto un sottoinsieme delle etichette di partenza sarà mantenuto per la fase successiva di espansione semantica.

L'algoritmo di filtraggio inizia con l'eliminazione delle cosiddette stop-words, ovvero di quei tag che devono essere esclusi subito dal set e che non saranno

certamente mantenuti. Nel nostro software, si è deciso di eliminare le parole di lunghezza inferiore a tre caratteri, i tag numerici e le date. Entriamo ora nel dettaglio di come è stata progettata la strategia di filtraggio delle etichette. Come vedremo, buona parte di questa fase si basa sull'utilizzo dei video correlati al filmato di partenza.

Il primo passo consiste nella creazione di un'istogramma di frequenza dei tag validi (si escludono soltanto le stop-words) all'interno dei Related Videos. Per ogni etichetta definita dall'autore, si contano le sue occorrenze nel set dei video correlati. Questo calcolo è importante in quanto per stabilire il ranking dei tag ci si basa in maniera piuttosto decisa sulle annotazioni fatte dagli autori dei video correlati. A questo punto, possiamo descrivere con precisione l'algoritmo di attribuzione dei punteggi agli *user-defined tag*.

In primo luogo, si prende in considerazione l'istogramma di frequenza dei tag di cui abbiamo parlato in precedenza. Si cerca al suo interno qual è il termine che mostra il più alto numero di occorrenze nell'insieme dei Related Videos. Vediamo ora i vari casi che si traducono nei diversi modi di attribuzione del punteggio. Nel caso in cui il tag sia "non valido" (si tratta di una stop-word), il punteggio coincide con -1.0. Il caso opposto si ha quando il tag corrente è contenuto all'interno del titolo del filmato analizzato; da varie prove sperimentali si è notato come la promozione delle etichette che fanno parte del titolo porti ad un miglioramento apprezzabile delle prestazioni dell'intero sistema. Quando siamo in presenza della suddetta condizione, il punteggio del tag coincide con 1.0.

Nel caso in cui il tag non sia contenuto nel titolo, vengono effettuati calcoli più approfonditi per arrivare alla sua classificazione. Si considerano due *threshold* per il filtraggio, che possiamo pensare come i due estremi di un intervallo di "mantenimento" dell'etichetta. Si calcola il rapporto tra il numero di occor-

renze del tag corrente nei *Related Videos* ed il numero di quest'ultimi, per avere una misura di quanto gli autori dei filmati correlati hanno utilizzato il tag in questione nelle loro annotazioni. Si noti come questa idea ricalchi abbastanza fedelmente quanto evidenziato nel paragrafo 2.1. Quando il rapporto appena calcolato è inferiore alla prima soglia (poco selettiva, valore tipico 0.15) il tag corrente assume un punteggio pari a 0.0, mentre se il rapporto è maggiore o uguale alla seconda soglia (molto selettiva, valore tipico 0.85) il punteggio è pari a 1.0. Per comodità rispetto a quelli che saranno i prossimi casi, rinominiamo il rapporto appena descritto con $termine_1$.

Nel caso in cui non sia verificato alcuno dei casi descritti finora, si apre la strada al terzo scenario di calcolo del punteggio. Quando il tag corrente è quello che presenta il bin più alto nell'istogramma delle frequenze, si calcola una quantità $termine_2$ come segue:

$$termine_2 = termine_1 + (1 - thresh_2) \tag{4.2}$$

dove con $thresh_2$ ci si riferisce alla seconda soglia di cui abbiamo parlato sopra. Nel caso in cui, invece, il tag corrente non coincida con il bin più alto dell'istogramma, si è scelto di includere nel punteggio una parte basata sulla *co-occorrenza*. Questa idea riprende in un certo qual modo i principi descritti nel paragrafo 2.2. La co-occorrenza viene misurata tra il tag corrente e quello che ha il maggior numero di occorrenze all'interno del set dei Related Videos. In altre parole:

$$coOccorrenza\,(tag_{corr}, tag_{rif}) = \#presenze_{simultanee}/\#presenze_{totali} \tag{4.3}$$

dove $\#presenze_{totali}$ indica il numero di volte che il tag corrente tag_{corr} è presente nelle annotazioni dei video correlati. In presenza di questo scenario, in

definitiva, la quantità $termine_2$ viene calcolata come segue:

$$termine_2 = coOccorrenza \cdot (termine_1 + (1 - thresh_2)) \tag{4.4}$$

La quantità in (4.4) viene utilizzata perciò nel calcolo del punteggio da attribuire al tag corrente. La formula a tal proposito è la seguente:

$$punteggioTag_{corr} = weight_1 \cdot termine_1 + weight_2 \cdot termine_2 \tag{4.5}$$

dove $weight_1$ (valore tipico 0.25) è il peso attribuito nel punteggio alla sola frequenza di comparsa del tag corrente nei Related Videos mentre $weight_2$ (valore tipico 0.75) è il peso associato alla sua co-occorrenza con il tag "forte" (quello che ha la frequenza più elevata).

Fino a questo momento, il punteggio di ogni *user-defined tag* dipende esclusivamente dall'utilizzo dei video correlati. Durante lo svolgimento del presente lavoro di tesi, è emerso come la strategia descritta finora fosse essenzialmente troppo debole. In altre parole, il solo utilizzo dei *Related Videos* porta all'eliminazione di alcuni tag che in realtà sono buoni. Per cercare di superare questo inconveniente, si è deciso di utilizzare la funzionalità di *comparazione tra termini* proposta dal toolkit Wikipedia Miner, descritto nel paragrafo 3.4. Vediamo nel dettaglio come si opera.

A partire dal set iniziale dei tag (inseriti dall'autore del video YouTube che stiamo analizzando) si estraggono, se esistono, quelli che presentano un punteggio pari a 1.0 e che quindi possono essere considerati tag "forti". Nel caso in cui non siano presenti tag forti nel set, i punteggi calcolati con la strategia dei *Related Videos* non vengono modificati. Questa scelta potrebbe sembrare troppo restrittiva, in quando l'utilizzo del toolkit Wikipedia Miner rimane vincolato alla presenza dei tag forti. In realtà, il metodo di calcolo del punteggio produce quasi

sempre almeno un'etichetta con punteggio 1.0 e quindi nella quasi totalità dei casi si riesce a sfruttare le potenzialità del suddetto strumento.

Nel caso in cui siano presenti dei tag forti, inizia la procedura di possibile modifica del punteggio. Si preparano due array contenenti rispettivamente l'insieme delle etichette con punteggio uguale a 1.0 e l'insieme delle etichette "deboli" (punteggio diverso da 1.0). A questo punto, si effettua un'iterazione lungo tutti i termini deboli che stanno nel rispettivo set: per ognuno di essi si calcola il valore di *relatedness* (parentela semantica) con i tag forti che abbiamo. Quando otteniamo un valore di parentela semantica che soddisfa la condizione:

$$relatedness/thresh_{tag} \geqslant thresh_1 \tag{4.6}$$

vuol dire che il tag debole in questione è fortemente legato dal punto di vista semantico al tag forte con il quale è stato messo in corrispondenza. Nell'espressione (4.6), $thresh_{tag}$ indica la soglia di filtraggio dei tag. E'un valore molto importante in quanto indica il limite al di sotto del quale il tag non viene mantenuto. In altre parole, se al termine del processo di ranking un tag assume un punteggio inferiore a $thresh_{tag}$, esso viene filtrato e non sarà considerato per la successiva fase di espansione semantica. Il valore $thresh_1$ indica invece quanto deve essere "forte" la similarità semantica tra le due etichette; come si può facilmente notare, $thresh_1$ è controllata da $thresh_{tag}$ e quindi più selettivo è il filtraggio che stiamo attuando ($thresh_{tag}$ elevata) maggiore dovrà essere *relatedness* per verificare ugualmente la condizione.

Quando si verifica la condizione (4.6), il punteggio di un tag debole viene modificato in accordo alla seguente espressione:

$$punteggio_{tagDebole} = thresh_{tag} + relatedness \tag{4.7}$$

Come si può notare, al tag debole viene incrementato il punteggio di una quantità pari al valore di parentela semantica con il corrispettivo tag forte. Nel caso in cui il punteggio modificato superi il valore massimo (1.0), esso viene tagliato a 1.0. Siamo arrivati a questo punto ad aver descritto per l'intero l'algoritmo di filtraggio e ranking dei tag. Ricordiamo che l'obiettivo di tutto questo processo è l'eliminazione di quelle etichette che presentano un punteggio inferiore a $thresh_{tag}$. E' possibile descrivere ora il prossimo step dell'algoritmo, ovvero la definizione di alcune strategie per l'espansione dello spazio iniziale dei tag.

4.2.3 L'espansione dei tag tramite l'utilizzo dei Related Videos

In questa sezione descriveremo la prima strategia attuata durante la tesi per quanto concerne l'incremento del numero iniziale di tag associati ad un video di YouTube. L'idea di base è quella di sfruttare i tag dei video correlati; fino ad ora essi hanno avuto funzione di filtraggio e ranking, mentre in questo frangente vogliamo sfruttare tali etichette per suggerire nuove annotazioni al filmato analizzato. Come vedremo in seguito, il metodo di espansione descritto in questa sezione non garantisce un grande livello di espansione. Tramite i *Related Videos*, infatti, non si riesce ad incrementare di molto il numero di tag e saranno necessarie ulteriori strategie per raggiungere tale obiettivo. Descriviamo in ogni caso il metodo di utilizzo dei Related Videos ai fini dell'espansione.

In primo luogo viene creato un array contenente l'insieme dei tag distinti presenti all'interno del set dei Related Videos. Questa operazione è necessaria in quanto nei video correlati avremo numerose ripetizioni delle etichette. Una volta che abbiamo a disposizione l'insieme dei tag unici, si costruisce un istogramma di frequenza di ognuno di essi nel set dei video correlati. Ogni bin dell'istogramma,

naturalmente, avrà un'altezza pari al numero di occorrenze del tag in questione all'interno del suddetto set. La procedura di suggerimento prevede quindi di promuovere quelle etichette che verificano:

$$altezza_{bin}/\#RelatedVideos \geqslant thresh \tag{4.8}$$

dove *thresh* è un valore di soglia impostato in base a quanto si vuole che sia restrittivo il suggerimento (valore tipico compreso in [0.6;0.8]). Come abbiamo già anticipato, la strategia appena descritta non garantisce buoni risultati in termini di espansione. Il motivo di tale risultanza deriva dal fatto che i tag molto frequenti nei Related Videos sono già presenti tra quelli definiti dall'autore del filmato; in altre parole vengono suggeriti dei doppioni. Alla base di quanto detto sta anche il fatto che l'algoritmo YouTube di ricerca dei correlati si basa tra l'altro sui tag del video di partenza.

A questo punto, è possibile passare alla descrizione dell'integrazione dei tag con il Semantic Web, al termine della quale noteremo come le etichette verranno arricchite sia in termini di numero sia dal punto di vista delle informazioni semantiche.

4.2.4 L'integrazione dei tag con il Semantic Web

In questa sezione affronteremo la parte del lavoro relativa al collegamento tra gli *user-defined tag* ed i due strumenti utilizzati in questo elaborato per quanto concerne la loro espansione semantica: **DBpedia** e **Wikipedia Miner**. Lo scopo di questa fase, come già annunciato, è quello di associare ad ogni filmato una serie di informazioni aggiuntive, che i soli tag non possono garantire. In pratica, ad ogni filmato vengono associati due file di testo all'interno dei quali sono inserite le note aggiuntive. Come vedremo nel seguito di questa sezio-

ne, l'algoritmo sviluppato effettua delle interrogazioni verso DBpedia al fine di ottenere un matching tra i tag e le risorse ivi presenti.

Nella seconda parte delle sezione presenteremo invece l'espansione basata su Wikipedia Miner; in quel caso il software non lavora in modalità *on line* bensì effettua delle interrogazioni verso un database locale. Tutti questi aspetti saranno trattati nel dettaglio all'interno del paragrafo dedicato a Wikipedia Miner. Partiamo adesso con la descrizione dettagliata delle strategie di integrazione col Semantic Web, facendo particolare attenzione al concetto di matching tra i tag di un video e le risorse legate ad ontologie.

4.2.4.1 L'espansione degli user-defined tag basata su DBpedia

Vediamo nel dettaglio come avviene l'espansione degli user-defined tag tramite il framework DBpedia, descritto nel paragrafo 3.2. Il primo obiettivo in questo contesto è il reperimento delle informazioni collegate ai tag filtrati del filmato, per mezzo delle risorse contenute proprio in DBpedia. In questo lavoro di tesi, la ricerca delle risorse all'interno del framework si basa sui *termini di ricerca*, creati a partire dai tag validati e classificati (come descritto in 4.2.2). Un termine di ricerca può essere composto da un singolo tag oppure tramite la combinazione di più etichette. Durante lo svolgimento del lavoro, infatti, è emerso come l'utilizzo dei tag singoli ai fini di query non sia sufficiente per ottenere un numero soddisfacente di match. Vediamo nel dettaglio i singoli passi di cui si compone la procedura di creazione dei termini di ricerca.

A partire dal set degli user-defined tag validati e classificati, si indagano le etichette dei *Related Videos* per controllare se esiste la possibilità di accorpare più tag insieme in modo da creare un termine di ricerca complesso. La procedura a questo proposito è molto ben definita. Si pensi di avere a disposizione l'array con il set di etichette appena menzionato. A partire dal primo tag, si controllano i tag dei video correlati al fine di verificare il numero di occorrenze simulanee

tra quest'ultimo ed il successivo all'interno dell'array. In base a quanto detto, il tag successivo viene concatenato al corrente se è verificata la condizione:

$$presenzeSimultane\,(tag_i, tag_{i+1})\,/presenze\,(tag_i) \geqslant thresh$$

dove *thresh* è un valore di soglia molto selettivo (valore tipico 0.9). Nel momento in cui un tag viene concatenato ad un altro, esso non può essere più preso in considerazione per i successivi passi di accorpamento. In altre parole, se otteniamo ad esempio un termine di ricerca costruito con la concatenazione di tag_1 e tag_2, non potrà mai accadere che un altro termine di ricerca comprenda tag_2 come prima parola. Un punto cruciale di questa fase consiste nella scelta del numero massimo di tag di cui può essere composto un termine di ricerca. Le prove sperimentali svolte in questo lavoro di tesi hanno mostrato che è inutile (ed in molti casi deleterio) effettuare interrogazioni a DBpedia con frasi composte da più di quattro termini. Più avanti, in questa sezione, mostreremo nel dettaglio come sono state fatte le query SPARQL verso DBpedia.

Dal punto di vista dell'accorpamento dei tag, si è scelto di non basarsi esclusivamente sui *Related Videos*. Effettuando una serie di prove, è emerso che anche il titolo del filmato può essere utilizzato per la creazione dei termini di ricerca DBpedia. L'idea di base è quella di accorpare insieme più tag che compaiono nel titolo del video analizzato. A partire dal set degli *user-defined tag* validati e classificati, si estraggono quelle etichette che hanno un punteggio pari a 1.0 (i tag cosiddetti forti). In buona sostanza, si legano tra loro questi tag, con il vincolo già utilizzato prima sul massimo numero di aggiunte (termini formati da non più di quattro etichette). L'unione dei tag che compaiono nel titolo non è comunque esente da un controllo ulteriore.

Vediamo di chiarire tale aspetto. Unire insieme dei tag solo perchè sono presenti nel titolo non è proprio la strategia migliore, in quanto spesso si rischia

di unire parole che in realtà sono piuttosto inefficaci all'atto della ricerca in DBpedia. Dopo aver riflettuto su varie possibilità, si è deciso di unire i tag che fanno parte del titolo basandosi sulla loro *co-occorrenza* all'interno del set dei *Related Videos*. Quando due etichette compaiono troppe poche volte insieme nelle annotazioni dei correlati, vuol dire che esse non sono poi così strettamente legate da giustificare una loro concatenazione. Come nei casi precedenti, ci si basa su una *threshold* al di sotto della quale i due tag del titolo non vengono uniti insieme. Può accadere, infine, che il termine di ricerca creato tramite il titolo del filmato sia già presente nel set dei termini di ricerca pre-computati: in questo caso si tratta semplicemente di non inserire un doppione all'interno dell'insieme.

Al termine della procedura descritta finora, vengono create una serie di stringhe per effettuare le query verso DBpedia. Si è scelto di utilizzare quest'ultime anche per la ricerca delle foto in Flickr[4], come abbiamo già annunciato nel paragrafo 3.5. Nelle prossime sezioni tratteremo nel dettaglio anche questa fase del lavoro. Assieme ai termini di ricerca per l'espansione tramite DBpedia, l'algoritmo forma inoltre quelli utilizzati per il toolkit Wikipedia Miner, rispetto al task di incrementare lo spazio iniziale dei tag. La creazione delle stringhe di ricerca per WM segue le medesime regole elencate finora per quanto riguarda le ontologie di DBpedia.

Entriamo adesso nel dettaglio del processo di matching tra gli user-defined tag e le risorse DBpedia, condotto tramite la ricerca all'interno delle rispettive classi ontologiche. Come già anticipato nel paragrafo 4.1, in questo lavoro di tesi si è deciso di effettuare una **mappatura** tra le ontologie (o "classi ontologiche") di DBpedia e le categorie dei video residenti in YouTube. In questo frangente, il software si occupa proprio della fase di accoppiamento tra classi ontologiche e categorie. Come abbiamo già descritto in 4.1, per effettuare tale

[4]http://www.flickr.com/

```
CATEGORIA "Autos & Vehicles"
SpeedwayLeague,GrandPrix,NascarDriver,Automobile,AutoRacingLeague,Aircraft,Spacecraft,AutomobileEngine,

CATEGORIA "Comedy"
Musical,Broadcast,ComicsCreator,Artist,Film,AdultActor,ComicsCharacter,Actor,Comedian,TelevisionEpisode

CATEGORIA "Education"
Scientist,School,Judge,Eukaryote,Country,SiteOfSpecialScientificInterest,Politician,Monarch,President,C

CATEGORIA "Entertainment"
Musical,Sport,Stadium,Celebrity,Broadcast,GrandPrix,MixedMartialArtsEvent,PlayboyPlaymate,SportsEvent,R

CATEGORIA "Film & Animation"
Musical,Broadcast,FilmFestival,Artist,Band,Film,AdultActor,Actor,Comedian,MusicFestival,MusicalWork,Eur

CATEGORIA "Gaming"
GaelicGamesPlayer,PokerPlayer,RadioControlledRacingLeague,VideogamesLeague,Game,VideoGame,
```

Figura 4.3: La mappatura delle *classi ontologiche* di DBpedia all'interno delle *categorie* di YouTube.

mappatura è stata sviluppata un'apposita classe Java utile alla memorizzazione delle informazioni all'interno di array. Quest'ultime vengono caricate da un file testo contenente i nomi delle categorie e le classi ad esse associate. Per avere un'idea circa la formattazione di tale file, si consideri la Figura 4.3.

Come si vede dalla Figura 4.3, ad ogni YouTube Category corrisponde una serie di classi ontologiche di DBpedia. L'assegnazione delle classi di interesse alle varie categorie è stata fatta manualmente, controllando una ad una le 257 classi e scegliendo di volta in volta a quali categorie associarle. Naturalmente, l'associazione tra classi e categorie è di tipo molti-a-molti: una certa ontologia può essere legata a diverse YouTube Category (e viceversa). Questa scelta consente di effettuare ricerche più accurate in DBpedia, ma al tempo stesso comporta un decadimento delle prestazioni del sistema in termini di tempo di elaborazione. Come vedremo in seguito, infatti, ogni termine viene ricercato all'interno di tutte le classi ontologiche (la ricerca si interrompe appena viene rilevato un match) quindi le prestazioni dell'algoritmo risentono dei tempi di attesa richiesti dal server *Sparql* di DBpedia.

Entriamo adesso nel dettaglio della procedura di matching tra i termini di ricerca DBpedia e le rispettive risorse. Per adempiere a tale compito, si è pensato di costruire una "struttura di query" SPARQL all'interno della quale

inserire l'ontologia di riferimento ed il *search term* in questione. I concetti teorici relativi al linguaggio SPARQL sono già stati trattati nel paragrafo 3.3.1. In questa sezione ci focalizzeremo sulla trattazione dell'aspetto pratico di SPARQL, ovvero la sua applicazione all'interno del Semantic Web. La query che abbiamo usato per tentare di trovare corrispondenze tra tag e risorse è la seguente:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT DISTINCT ?entita ?nome
WHERE {
    ?entita rdf:type dbo:Classe_Ontologica .
    ?entita rdfs:label ?nome .
    FILTER regex (?nome, "Termine_Di_Ricerca", 'i')
      }
LIMIT 100
```

Come si può vedere, i parametri della query sovrastante coincidono proprio con l'*ontologia* di riferimento e con il *search term* utilizzato. L'algoritmo effettua un'iterazione lungo tutti i search term; per ogni termine inizia la ricerca all'interno di tutte le classi ontologiche fino a quando non viene riscontrata una corretta corrispondenza. A quel punto la ricerca viene interrotta e si passa al search term successivo. Al termine di questa procedura, è possibile avere a disposizione una serie di nomi di risorse DBpedia da mettere in corrispondenza con gli *user-defined tag* dai quali siamo partiti.

A questo punto, dopo aver catturato i nomi delle risorse DBpedia, si possono carpire le informazioni semantiche ad esse collegate. In altre parole, si effettua un'interrogazione verso DBpedia mirata ad ottenere i valori delle *proprietà* associate alle risorse. L'idea, come nel caso precedente, è quella di creare una

struttura di query all'interno della quale inserire il nome della risorsa trovato. Il codice SPARQL di riferimento è il seguente:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT DISTINCT ?entita ?nome
WHERE {
        ?entita rdf:type dbo:Classe_Ontologica_Risorsa .
        ?entita rdfs:label ?nome .
        ?entita ?prop ?valoreProprieta .
        FILTER (?nome = "Nome_risorsa"@en)
          }
```

La stringa `Nome_risorsa` coincide col nome della risorsa DBpedia trovata tramite la query precedente, mentre `Classe_Ontologica_Risorsa` è il nome di quella particolare ontologia che ha garantito la corrispondenza con il termine di ricerca. Come si può vedere dall'interrogazione sovrastante, vengono catturati il soggetto (`?entita`) ed il valore del predicato della risorsa (`?nome`). In altri termini, si effettua un'espansione semantica basata sulle proprietà delle risorse e sui loro valori alfanumerici.

Conviene chiarire il meccanismo di matching tra i tag definiti dagli utenti e le risorse DBpedia tramite un esempio. Supponiamo di analizzare un filmato YouTube raffigurante una gara di vela, la "Volvo Ocean Race", e di cercare il matching con la rispettiva risorsa DBpedia. Il filmato ha a disposizione una serie di tag, tra cui *Volvo Ocean Race*. L'algoritmo di ricerca delle corrispondenze, tramite la prima query SPARQL mostrata sopra, individuerà una serie di risorse candidate a validare la corrispondenza. Tra queste, verrà scelta la risorsa corretta, attraverso la quale possono essere estratte tutte le informazioni

Figura 4.4: Matching tra i tag di YouTube ed i concetti DBpedia.

relative alla "Volvo Ocean Race". Per avere una visione globale di tale processo, si consideri la Figura 4.4.

Fino ad ora, è stata volutamente trascurata la questione relativa a "come" l'algoritmo sceglie la risorsa DBpedia da mettere in corrispondenza con i tag. In altre parole, non abbiamo descritto in che modo il software effettua il matching tra DBpedia e le etichette di partenza dei video. Descriviamo questo aspetto adesso, dopo aver mostrato l'intero procedimento di estrazione delle informazioni semantiche.

A partire dal nome di ogni risorsa candidata e dal set dei termini di ricerca DBpedia, vengono effettuati una serie di controlli. In primo luogo, si controlla se il *nome* della risorsa candidata corrente **coincide** con il termine di ricerca dal quale sono partito. In caso di esito positivo, il matching può essere immediatamente validato. Se il nome della risorsa candidata contiene uno degli altri termini di ricerca creati in precedenza, il matching può essere validato senza ulteriori controlli. In caso contrario, si utilizza il toolkit Wikipedia Miner per controllare il grado di correlazione semantica tra search term e risorsa candidata. Vediamo nel particolare come si utilizza tale framework. In primo luogo,

si estraggono gli "articoli" di riferimento per il termine di ricerca e per il nome della risorsa (si ricordi a tal proposito quanto scritto nel paragrafo 3.4). A partire dagli articoli, si utilizza la funzione `getRelatednessTo(...)` per trovare il valore di correlazione semantica tra le due stringhe. Nel caso in cui tale valore superi una certa *threshold*, la corrispondenza viene validata. In definitiva, quindi, la fase di ricerca delle corrispondenze tra tag e concetti si basa su due fattori: il nome della risorsa candidata e la correlazione semantica tra termini. A questo punto, si può passare alla descrizione della seconda parte di espansione semantica, fatta come già annunciato in precedenza tramite il toolkit Wikipedia Miner.

4.2.4.2 L'espansione degli user-defined tag basata su Wikipedia Miner

La parte operativa inerente questa parte ha coinvolto due aspetti: la creazione di un database nel quale archiviare la struttura di una delle versioni Inglesi di Wikipedia e l'utilizzo delle funzionalità messe a disposizione dal toolkit stesso. Iniziamo perciò con la descrizione di quanto fatto rispetto alla prima parte, analizzando le varie operazioni svolte per quanto riguarda la gestione del database contenente Wikipedia.

Occorre fare innanzitutto chiarezza riguardo ai passi da svolgere legati alla creazione della base di dati appena menzionata. La prima operazione da fare è eseguire il download di una versione di Wikipedia. A questo proposito, sono due le strade che possono essere percorse. Se si vuole archiviare su disco fisso l'intero contenuto dell'enciclopedia on line, è necessario eseguire il *dump* completo di Wikipedia. Per effettuare questa operazione, è sufficiente utilizzare l'indirizzo web[5] attraverso il quale è possibile scaricare le varie versioni di backup. Normalmente, effettuare il dump completo dell'intero contenuto di Wikipedia comporta

[5] http://download.wikimedia.org/backup-index.html

un notevole dispendio di risorse per quanto concerne lo spazio occupato su disco. Attualmente, alcune versioni dell'enciclopedia occupano infatti anche 20 GB di hard disk. Valutando attentamente i pro ed i contro di un simile scenario, si è deciso di operare effettuando il download[6] di una delle versioni *pre-computate* di Wikipedia. Si tratta di archivi già pronti che contengono la sola "struttura" dell'enciclopedia, senza tenere traccia del contenuto dei vari articoli. Per quelli che sono gli scopi di questo lavoro di tesi, si è scelto di operare con una versione pre-computata risalente al 23/04/2009.

Vediamo a questo punto di chiarire con esattezza la struttura del database grazie al quale è possibile utilizzare le funzionalità messe a disposizione dal toolkit Wikipedia Miner. Prima di fare questo, facciamo una rapida puntualizzazione. La base di dati appena menzionata è stata gestita tramite il *Data Base Management System MySQL 5.1*, installato all'interno del Sistema Operativo *Ubuntu 10.04 LTS*. Il database con cui lavoriamo si chiama "wikipedia" ed è composto da 16 tabelle; per avere una prima impressione circa la sua struttura si consideri la Figura 4.5, dove troviamo i nomi delle varie tabelle. Per dare una visione più completa dell'intero database, descriviamo nel dettaglio la struttura ed il contenuto delle singole tabelle:

1. **anchor**. E' la tabella che associa il testo utilizzato nei "collegamenti" alle pagine alle quali essi portano. Contiene tre campi: `an_text` (la stringa presente nei collegamenti), `an_to` (la pagina di destinazione), `an_count` (il numero di volte che quel particolare anchor è utilizzato per collegarsi alla pagina specificata). Vedremo nel seguito come questa tabella sia molto importante per quanto concerne l'incremento del numero di tag di un video. La struttura di *anchor* è visibile in Figura 4.6.

2. **anchor_occurance**. Contiene il numero di articoli al quale un certo

[6] http://sourceforge.net/projects/wikipedia-miner/files/

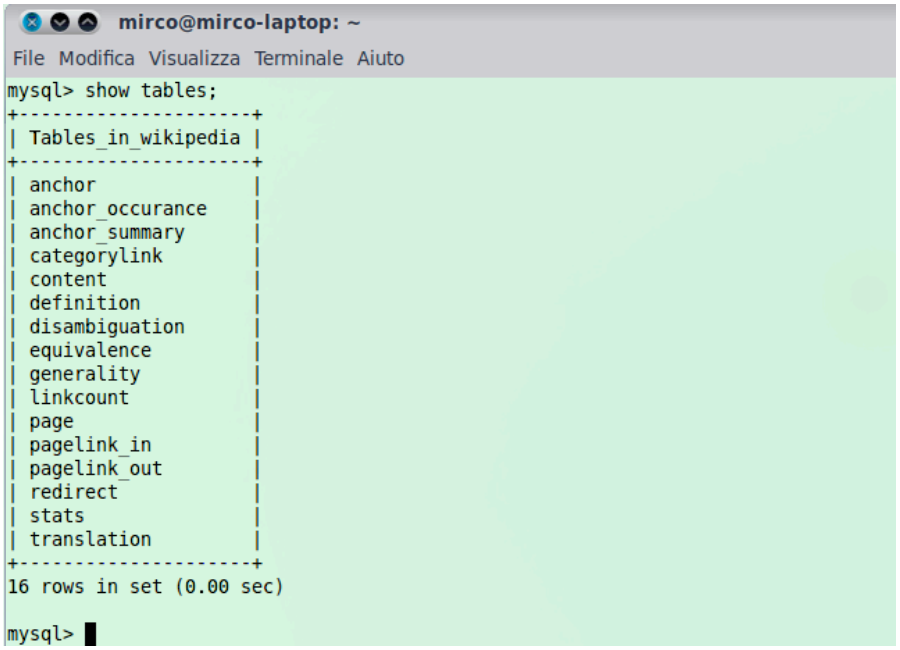

Figura 4.5: Le tabelle del database “wikipedia”.

```
mysql> describe anchor;
+----------+-----------------+------+-----+---------+-------+
| Field    | Type            | Null | Key | Default | Extra |
+----------+-----------------+------+-----+---------+-------+
| an_text  | varchar(10000)  | NO   |     | NULL    |       |
| an_to    | int(8) unsigned | NO   |     | NULL    |       |
| an_count | int(8) unsigned | NO   |     | NULL    |       |
+----------+-----------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql>
```

Figura 4.6: Database “wikipedia”: tabella *anchor*.

```
mysql> describe anchor_occurance;
+--------------+-----------------+------+-----+---------+-------+
| Field        | Type            | Null | Key | Default | Extra |
+--------------+-----------------+------+-----+---------+-------+
| ao_text      | varchar(10000)  | NO   |     | NULL    |       |
| ao_linkCount | int(8) unsigned | NO   |     | NULL    |       |
| ao_occCount  | int(8) unsigned | NO   |     | NULL    |       |
+--------------+-----------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql>
```

Figura 4.7: Database "wikipedia": tabella *anchor_ occurance*.

```
mysql> describe anchor_summary;
+------------+----------------+------+-----+---------+-------+
| Field      | Type           | Null | Key | Default | Extra |
+------------+----------------+------+-----+---------+-------+
| anchorText | varchar(10000) | YES  |     | NULL    |       |
| countComb  | mediumtext     | YES  |     | NULL    |       |
+------------+----------------+------+-----+---------+-------+
2 rows in set (0.00 sec)

mysql> select * from anchor_summary limit 1;
+-----------------------------+------------+
| anchorText                  | countComb  |
+-----------------------------+------------+
| Kenya national football team | 1105751:10 |
+-----------------------------+------------+
1 row in set (0.04 sec)

mysql>
```

Figura 4.8: Database "wikipedia": tabella *anchor_ summary*.

anchor fa riferimento ed il numero di articoli ai quali l'anchor stesso non si riferisce. E'una tabella formata quindi da tre campi, come si può notare in Figura 4.7.

3. **anchor_summary**. Riassume le informazioni relative ad un particolare *anchor*. I campi contenuti in questa tabella sono due: `anchor_text` (la stringa indicante l'*anchor* in questione) e `countComb` (la lista delle pagine di destinazione ed il numero di volte in cui un *anchor* è stato utilizzato). La Figura 4.8 chiarisce la struttura della tabella e mostra un esempio di

```
mysql> describe categorylink;
+-----------+-----------------+------+-----+---------+-------+
| Field     | Type            | Null | Key | Default | Extra |
+-----------+-----------------+------+-----+---------+-------+
| cl_parent | int(8) unsigned | NO   |     | NULL    |       |
| cl_child  | int(8) unsigned | NO   |     | NULL    |       |
+-----------+-----------------+------+-----+---------+-------+
```

Figura 4.9: Database "wikipedia": tabella *categorylink*.

```
mysql> describe content;
+------------+-----------------+------+-----+---------+-------+
| Field      | Type            | Null | Key | Default | Extra |
+------------+-----------------+------+-----+---------+-------+
| co_id      | int(8) unsigned | NO   | PRI | NULL    |       |
| co_content | mediumblob      | NO   |     | NULL    |       |
+------------+-----------------+------+-----+---------+-------+
2 rows in set (0.00 sec)

mysql>
```

Figura 4.10: Database "wikipedia": tabella *content*.

record contenuto al suo interno.

4. **categorylink**. Associa i vari articoli alla categoria di riferimento. Contiene due campi: `parent_id` (intero che rappresenta l'articolo) e `child_id` (valore della categoria). La Figura 4.9 chiarisce quanto detto finora.

5. **content**. E' la tabella che dovrebbe memorizzare il contenuto degli articoli Wikipedia. Attualmente non viene utilizzata in quanto come detto si è scelto di lavorare con una versione pre-computata di Wikipedia che ne contiene solamente la struttura. La Figura 4.10 mostra comunque la struttura di questa tabella.

6. **definition**. Contiene le definizioni associate alle entità presenti in Wikipedia. La tabella è formata da tre campi: `df_id`, `df_firstSentence` (la definizione breve) e `df_firstParagraph` (la definizione più lunga). La Figura 4.11 ne mostra la struttura.

```
mysql> describe definition;
+------------------+-----------------+------+-----+---------+-------+
| Field            | Type            | Null | Key | Default | Extra |
+------------------+-----------------+------+-----+---------+-------+
| df_id            | int(8) unsigned | NO   | PRI | NULL    |       |
| df_firstSentence | mediumtext      | NO   |     | NULL    |       |
| df_firstParagraph | mediumtext     | NO   |     | NULL    |       |
+------------------+-----------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql>
```

Figura 4.11: Database "wikipedia:" tabella *definition*.

```
mysql> describe disambiguation;
+----------+-----------------+------+-----+---------+-------+
| Field    | Type            | Null | Key | Default | Extra |
+----------+-----------------+------+-----+---------+-------+
| da_from  | int(8) unsigned | NO   |     | NULL    |       |
| da_to    | int(8) unsigned | NO   |     | NULL    |       |
| da_index | int(3) unsigned | NO   |     | NULL    |       |
| da_scope | mediumblob      | NO   |     | NULL    |       |
+----------+-----------------+------+-----+---------+-------+
4 rows in set (0.00 sec)

mysql>
```

Figura 4.12: Database "wikipedia": tabella *disambiguation*.

7. **disambiguation**. Associa le pagine di disambiguamento con i vari sensi ad esse associati. Contiene quattro campi: `from_id` (*id* della pagina di disambiguamento), `to_id` (*id* della pagina con il senso), `index` (la posizione assunta dal "senso" in questione), `scope_note` (una stringa che spiega in che modo quel particolare "senso" è legato al termine ambiguo). La Figura 4.12 mostra la struttura della tabella.

8. **equivalence**. Associa le categorie e gli articoli che corrispondono allo stesso concetto. E' formata da due campi che indicano l'identificativo della categoria e dell'articolo. La Figura 4.13 ne mostra la struttura.

9. **generality**. Contiene la distanza minima (profondità) tra un articolo o una categoria e la radice della struttura di categorie Wikipedia. Valori

```
mysql> describe equivalence;
+--------+-----------------+------+-----+---------+-------+
| Field  | Type            | Null | Key | Default | Extra |
+--------+-----------------+------+-----+---------+-------+
| eq_cat | int(8) unsigned | NO   | PRI | NULL    |       |
| eq_art | int(8) unsigned | NO   |     | NULL    |       |
+--------+-----------------+------+-----+---------+-------+
2 rows in set (0.00 sec)

mysql>
```

Figura 4.13: Database "wikipedia": tabella *equivalence*.

```
mysql> describe generality;
+----------+-----------------+------+-----+---------+-------+
| Field    | Type            | Null | Key | Default | Extra |
+----------+-----------------+------+-----+---------+-------+
| gn_id    | int(8) unsigned | NO   | PRI | NULL    |       |
| gn_depth | int(2) unsigned | NO   |     | NULL    |       |
+----------+-----------------+------+-----+---------+-------+
2 rows in set (0.00 sec)

mysql>
```

Figura 4.14: Database "wikipedia": tabella *generality*.

bassi di profondità stanno ad indicare argomenti generali mentre valori alti rappresentano concetti molto particolari. La Figura 4.14 mostra la struttura di tale tabella.

10. **linkcount**. Tiene traccia del numero di collegamenti in ingresso ed in uscita ad un cero articolo. E' formata da tre campi: `lc_id` (identificativo dell'articolo), `lc_in` (numero di entrate), `lc_out` (numero di uscite). La Figura 4.15 mostra la struttura della tabella.

11. **page**. E' la tabella che archivia i titoli ed i tipi delle pagine di Wikipedia. Contiene tre campi: `page_id`, `page_title`, `page_type` (1 = ARTICLE, 2 = CATEGORY, 3 = REDIRECT, 4 = DISAMBIGUATION). La Figura 4.16 mostra la struttura di tale tabella.

12. **pagelink_in**. E' la tabella che tiene traccia dei collegamenti in entrata

```
mysql> describe linkcount;
+--------+-----------------+------+-----+---------+-------+
| Field  | Type            | Null | Key | Default | Extra |
+--------+-----------------+------+-----+---------+-------+
| lc_id  | int(8) unsigned | NO   | PRI | NULL    |       |
| lc_in  | int(8) unsigned | NO   |     | NULL    |       |
| lc_out | int(8) unsigned | NO   |     | NULL    |       |
+--------+-----------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Figura 4.15: Database "wikipedia": tabella *linkcount*.

```
mysql> describe page;
+------------+-----------------+------+-----+---------+-------+
| Field      | Type            | Null | Key | Default | Extra |
+------------+-----------------+------+-----+---------+-------+
| page_id    | int(8) unsigned | NO   | PRI | NULL    |       |
| page_title | varchar(255)    | NO   |     | NULL    |       |
| page_type  | int(2)          | NO   |     | 0       |       |
+------------+-----------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql>
```

Figura 4.16: Database "wikipedia": tabella *page*.

alle varie pagine. Nella Figura 4.17 abbiamo mostrato la struttura di questa tabella ed anche un'istantanea dei primi cinque records in essa contenuti.

13. **pagelink_out**. Ha lo stesso significato della tabella precedente, solo che tiene traccia del collegamenti in uscita anzichè in entrata. Anche i tipi dei campi sono i medesimi del caso precedente.

14. **redirect**. E' la tabella che permette di ri-direzionare le pagine. Contiene due campi: `rd_from` (un intero che indica la pagina di partenza), `rd_to` (la pagina di destinazione).

15. **stats**. Fornisce alcune statistiche relative al dump di Wikipedia con il quale si lavora. La Figura 4.18 mostra il contenuto (un solo record) di questa tabella.

```
mysql> describe pagelink_in;
+---------+-----------------+------+-----+---------+-------+
| Field   | Type            | Null | Key | Default | Extra |
+---------+-----------------+------+-----+---------+-------+
| li_id   | int(8) unsigned | NO   | PRI | NULL    |       |
| li_data | mediumblob      | NO   |     | NULL    |       |
+---------+-----------------+------+-----+---------+-------+
2 rows in set (0.00 sec)

mysql> select * from pagelink_in limit 5;
+----------+-------------------------------------------+
| li_id    | li_data                                   |
+----------+-------------------------------------------+
| 17509100 | 100735:7272957                            |
| 11643172 | 2504450:3013547                           |
|  8391912 |                                           |
|  9028018 | 407665:1808559:2109455:11485468:11486064  |
|  8264616 | 514622:1202879:15125787:15736610:17220571 |
+----------+-------------------------------------------+
```

Figura 4.17: Database “wikipedia”: tabella *pagelink_ in*.

```
mysql> select * from stats;
+-------------+---------------+--------------+--------------+
| st_articles | st_categories | st_redirects | st_disambigs |
+-------------+---------------+--------------+--------------+
|     2388612 |        389980 |      2964924 |       107560 |
+-------------+---------------+--------------+--------------+
1 row in set (0.01 sec)

mysql>
```

Figura 4.18: Database “wikipedia”: il contenuto della tabella *stats*.

16. **translation**. Contiene le traduzioni in varie lingue dei titoli degli articoli. La tabella presenta tre campi: `tl_id` (articolo), `tl_lang` (iniziali della lingua in cui viene fatta la traduzione (es. *en*, *fr*, *jp*, etc...), `tl_text` (stringa che rappresenta la traduzione).

Dopo aver descritto nel dettaglio il database attraverso il quale viene realizzata la seconda fase di espansione semantica, possiamo passare alla spiegazione vera e propria dell'algoritmo implementato. Prima di addentrarci nella descrizione della procedura, facciamo una piccola digressione. Occorre ricordare, in primo luogo, che essa viene effettuata a partire dai tag validati grazie alla strategia descritta nel paragrafo 4.2.2. Partendo dai tag, poi, vengono creati i **termini di ricerca** che rappresentano la base per il processo di espansione, come abbiamo già descritto (per DBpedia) nel paragrafo 4.2.4.1. Nel caso corrente di Wikipedia Miner, i *search term* vengono creati in modo analogo al caso precedente, con una sola differenza legata al numero massimo di tag che possono comporre il termine di ricerca.

Se nel caso di DBpedia esiste la possibilità di avere termini di ricerca composti da quattro tag, per l'espansione con Wikipedia Miner si è scelto di aggregare al massimo due tag. Il motivo di questa scelta è legato all'algoritmo di matching tra il termine e l'articolo di Wikipedia. Sperimentando varie soluzioni, infatti, risulta molto più probabile avere una corrispondenza corretta nel caso di stringhe di ricerca "corte". Da qui la scelta di non legare più di due tag assieme. Nel seguito della trattazione, quando entreremo nel merito della procedura di matching, queste osservazioni risulteranno più chiare.

Il primo passo svolto dall'algoritmo consiste nella "formattazione" dei termini di ricerca associati a Wikipedia Miner. All'interno del database descritto in precedenza, i titoli delle pagine contenuti nella tabella *page* presentano una struttura ben precisa. In pratica, nel titolo, ogni parola inizia con una lettera

maiuscola. E' quindi preferibile, per realizzare correttamente la ricerca, adeguare i *search terms* a tale sintassi. Per ogni termine di ricerca, si costruisce un array di stringhe contenente i termini di "contesto", ovvero gli altri *search terms* che fanno parte del set finale. La ricerca dell'articolo di Wikipedia collegato al termine corrente avviene tramite il metodo `getWeightedArticles` della classe Wikipedia. La riga di codice che inizia la ricerca è la seguente:

```
SortedVector<Article> artTag = w.getWeightedArticles(
    searchFormatted[i], null, contextSearch);
```

dove gli argomenti utilizzati consistono nel search term corrente (formattato) e nel vettore dei termini di contesto. La chiamata a questo metodo produce una lista ordinata di articoli Wikipedia, dalla quale dovremo estrarre il match corretto.

A questo punto si tratta di estrarre l'articolo corretto tra quelli presenti in `artTag`. Descriviamo quindi nel dettaglio l'algoritmo di scelta. La prima condizione per la rilevazione di una corretta corrispondenza si basa sui termini di contesto. In particolare, si scorre la lista degli articoli esaminando il loro titolo. Se il titolo di un articolo contiene uno dei termini di contesto, il match viene subito verificato. Si è scelto di privilegiare questa condizione rispetto a tutte le altre perchè la presenza di uno dei termini di contesto all'interno del titolo è garanzia di fortissima correlazione tra l'articolo stesso ed il termine di ricerca dal quale si parte.

La seconda condizione che porta ad un matching utilizza l'insieme dei tag filtrati e pesati tramite la procedura descritta nel paragrafo 4.2.2. L'idea è la medesima del caso precedente: se il titolo di un articolo contiene uno dei tag di partenza allora la corrispondenza è verificata. Il terzo criterio si basa invece sul *peso* che il metodo `getWeightedArticles` attribuisce ad ogni articolo del set. Nel caso in cui esso superi una *threshold* pre-impostata, l'articolo collegato

è senza dubbio meritevole di entrare in relazione con il termine di ricerca dal quale siamo partiti. Nel caso in cui nessuna delle condizioni precedenti venga verificata, si sceglie l'articolo che presenta il peso maggiore tra tutti quelli presenti nella lista ordinata.

Dopo aver chiarito in che modo avviene la procedura di scelta dell'articolo corretto, descriviamo le prossime fasi del processo di espansione semantica. A partire dall'articolo scelto, vengono estratti i suoi **sinonimi** tramite il metodo `getAnchorText` della classe `Article`. Conviene evidenziare questa parte tramite il codice Java di riferimento:

```
for (Article.AnchorText at:realArt.getAnchorTexts()) {
    vettSinonimi[numSinonimi] = at.getText();
    countSinonimi[numSinonimi] = at.getCount();
    numSinonimi++;
}
```

Come si può vedere, vengono costruiti due array, contenenti rispettivamente i sinonimi veri e propri e la variabile indicante il numero di volte in cui il sinonimo corrente è utilizzato come riferimento alla pagina Wikipedia in questione (questo valore coincide col campo *an_count* della tabella *anchor*). Normalmente, l'iterazione mostrata nel codice sovrastante produce una moltitudine di sinonimi ed è necessario perciò eseguire su di loro un filtraggio. Un'espansione semantica basata su tutti i sinonimi disponibili, infatti, causerebbe un incremento spropositato dello spazio iniziale dei tag ma al tempo stesso introdurrebbe una notevole quantità di rumore. Si è scelto quindi di attuare un'algoritmo di decisione sul set totale dei sinonimi: alcuni vengono mantenuti mentre altri (la maggior parte) vengono esclusi. Vediamo di spiegare nel dettaglio l'idea che sta alla base di questa procedura.

Dopo aver valutato con alcuni esempi il database, è emerso che i sinonimi

"migliori" sono quelli il cui valore di *an_ count* è più alto. In base a tale rilevanza, si è scelto di attuare una procedura volta a promuovere questo tipo di anchors. Dal punto di vista operativo, si calcola innanzitutto il valore medio di *an_ count* per l'intero set di sinonimi associati ad un articolo Wikipedia. Ricordando che non sono ammessi sinonimi coincidenti con il termine di ricerca iniziale, la formula di promozione degli anchors è la seguente:

$$anCount_i / mean\,(anCount) \geqslant thresh \tag{4.9}$$

dove $anCount_i$ è il valore di *an_ count* del sinonimo i-esimo, $mean\,(anCount)$ è il valore medio appena menzionato mentre $thresh$ è un valore di soglia preimpostato (valore tipico in [0.2;0.4]). Dopo che il set di sinonimi buoni è stato estratto, si crea un file di testo all'interno del quale essi vengono memorizzati, assieme alle altre informazioni che compongono l'espansione semantica basata su Wikipedia Miner. Oltre agli anchors, infatti, in tale file viene inserita anche la definizione relativa all'articolo di Wikipedia che è stato selezionato. Quest'ultima consiste nel campo *df_ firstParagraph* della tabella *definition*.

In definitiva, quindi, l'espansione semantica consiste nei sinonimi estratti (che vanno ad incrementare il set finale dei tag) e nelle informazioni di contorno date dalla definizione relativa all'articolo. Ogni **anchor** registrato su file, come detto, sarà un **tag** in più nel set finale delle etichette. Per avere una visione più precisa della struttura di questo file, si consideri la Figura 4.19 , dove è visualizzata una parte dell'espansione semantica relativa al video YouTube rintracciabile all'indirizzo web `http://www.youtube.com/watch?v=6L8845utZI4`.

Come si può vedere, nel file di espansione è presente anche una sezione identificata con "FIRST PARAGRAPH"; questa parte coincide con la definizione dell'articolo che è stato selezionato. In definitiva, quindi, col processo di espansione semantica di Wikipedia Miner si raggiunge un duplice obiettivo: l'incre-

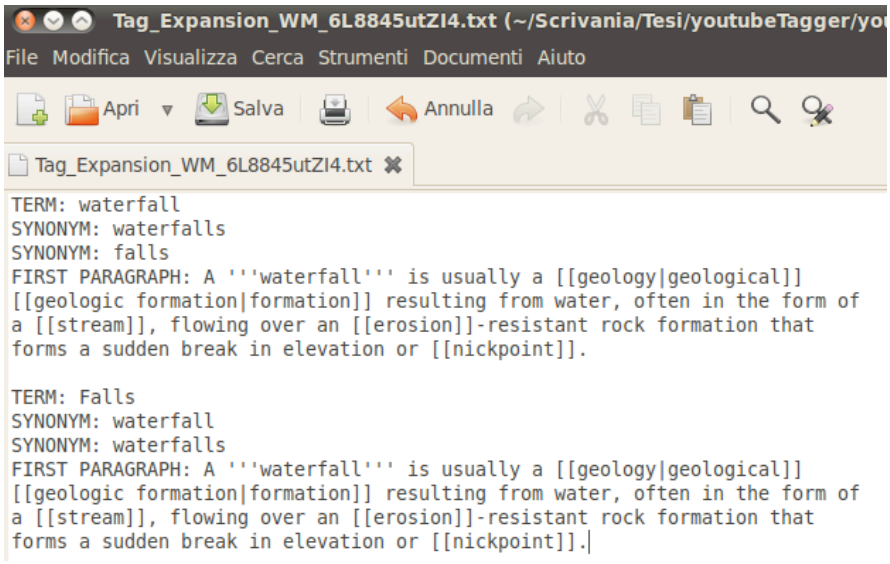

Figura 4.19: Esempio di espansione semantica tramite Wikipedia Miner.

mento del numero di tag associati al video e l'aggiunta di contenuto informativo descrittivo al video stesso. Benchè quest'ultimo aspetto non si presti facilmente alla valutazione quantitativa del sistema, rappresenta comunque un primo passo verso possibili sviluppi futuri del presente lavoro di tesi. Tratteremo i possibili scenari futuri nell'ultimo capitolo di questo elaborato, per il momento limitiamoci a pensare che l'espansione semantica potrebbe essere il punto di partenza per l'interconnessione di contenuto multimediale all'interno del Web of Data. Possiamo passare adesso all'ultimo step dell'algoritmo proposto: il download delle immagini dal sito web Flickr.

4.2.5 Il download delle immagini dal sito Flickr

In questa sezione descriviamo la fase di download delle immagini dal sito web Flickr[7]. Le foto vengono ricercate all'interno di questa sorgente sulla base dei termini di ricerca creati appositamente per Flickr. Per dare maggiore chiarezza a

[7] http://www.flickr.com/

questo aspetto, descriviamo nel dettaglio come avviene la creazione dei termini di ricerca. Nei *seach terms* destinati a Flickr, vengono inseriti innanzitutto quelli creati in fase precedente per DBpedia. Una parte dei termini di ricerca di Flickr coincide perciò con quelli utilizzati nella ricerca in DBpedia.

In secondo luogo, i search terms di Flickr sono costituiti dagli *anchors* estratti durante l'espansione semantica fatta con Wikipedia Miner. A questo proposito è utile fare un ulteriore chiarimento. Durante l'estrazione dei sinonimi con Wikipedia Miner, è frequente il caso in cui vengono mantenuti dei termini (che poi diventano anche nuovi tag del video analizzato) "vicini" tra loro, rispetto al numero di lettere uguali che li compongono. Per non appesantire troppo il download delle immagini da Flickr, si è scelto di creare dei termini di ricerca piuttosto diversi l'uno dall'altro. I motivi che hanno portato a questa scelta risiedono essenzialmente nel tempo di elaborazione richiesto dalla fase di download. Come già anticipato nel Capitolo Introduzione, il download delle immagini da Flickr serve per l'integrazione di questa tesi con il lavoro svolto da [Smeraldo10]. In quest'ultimo, il sistema per poter funzionare necessità di svariate immagini collegate ai concetti di partenza; in questa tesi si è perciò scelto di limitare il numero dei termini di ricerca in modo da poter scaricare un buon numero di foto per ogni termine.

Dopo questa puntualizzazione, torniamo alla descrizione della fase di download. L'intero processo di scaricamento avviene all'interno di un'iterazione sul numero di termini di ricerca Flickr. Per ogni termine, si impostano i parametri di ricerca, imponendo che le immagini carpite dalla sorgente web siano ordinate per "rilevanza" (`SearchParameters.RELEVANCE`, vedi paragrafo 3.5). A seconda che il termine di ricerca sia composto da più parole o che sia singolo, si utilizzano due metodi diversi della classe `SearchParameters` per finire di impostare i parametri di ricerca. Nel caso di search term composto, si utilizza il metodo

`setText`, altrimenti il metodo `setTags`.

Per ogni termine di ricerca, il download delle immagini avviene sulla base di un numero massimo di foto (500) che ogni volta vengono inserite in una lista di immagini candidate. Si parla di lista di foto candidate in quanto si è scelto di impostare il vincolo di **unique-user** sull'insieme delle immagini. Nella cartella che le contiene, quindi, è presente al massimo una foto per ogni utente registrato in Flickr. Questa scelta deriva da una motivazione ben precisa. Svolgendo delle prove sperimentali, è emerso che prendere più immagini dello stesso utente comporta la presenza di foto molto simili all'interno del set finale. Questo fatto è facilmente spiegabile considerando che un utente ogni volta pubblicherà molte foto (simili tra loro) relative ad un certo contenuto. Per le necessità mostrate dal lavoro di tesi di Salvatore Smeraldo ([Smeraldo10]), svolto parallelamente al mio elaborato, si è dunque scelto di imporre il vincolo dell'utente unico sul set finale di immagini.

Nel caso di termine di ricerca composto il numero di foto scaricate è 25, mentre con termine singolo (una sola parola) vengono estratte 20 foto. Le immagini prese da Flickr vengono salvate in formato *jpg* all'interno di una cartella residente nella stessa directory del file eseguibile. All'interno di questa cartella, il software provvede a creare un file di testo con le informazioni di interesse relative alle immagini estratte: il path di ogni foto, il suo termine di ricerca, il numero di tag e la relativa lista inserita dall'autore dell'immagine. Per avere maggiore chiarezza circa la struttura di questo file, si consideri la Figura 4.20. Nella figura si può notare che i termini di ricerca ed i tag delle foto sono stati inseriti senza gli spazi tra le parole che li compongono: questa scelta deriva da una richiesta di Smeraldo ed è legata ad una più semplice integrazione con il lavoro da egli svolto. Esiste la possibilità che il numero massimo di foto estratte per ogni termine (500) non sia sufficiente a produrre le 20 o le 25 immagini

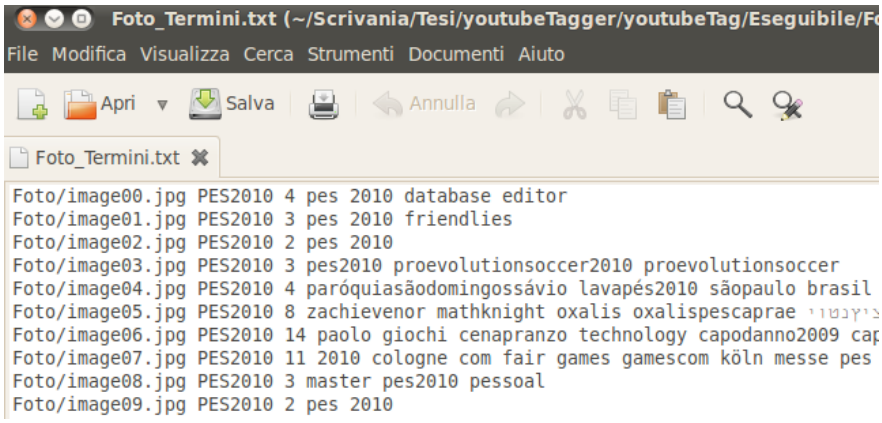

Figura 4.20: File "Foto_Termini.txt": informazioni di interesse legate alle foto estratte.

con rispetto del vincolo di unique-user: in questo caso, semplicemente, ci si accontenta di avere a disposizione un minor numero di immagini. Lo scenario appena descritto, comunque, non accade quasi mai nella pratica. Con quest'ultima precisazione, è esaurita pure la descrizione dell'ultima operazione svolta dal software realizzato nel presente lavoro di tesi. Passiamo adesso ad un breve resoconto finale circa gli aspetti più rilevanti affrontati durante la realizzazione del sistema.

4.3 Considerazioni finali sul software sviluppato

In questo capitolo abbiamo affrontato la parte implementativa che ha caratterizzato il presente lavoro di tesi. L'obiettivo iniziale del lavoro consisteva nella definizione di una procedura per il filtraggio e l'espansione dei tag di un video di YouTube. Abbiamo visto come entrambi gli aspetti siano stati affrontati in questa tesi, anche grazie all'ausilio di strumenti quali DBpedia e Wikipedia Miner. Ai fini del filtraggio, ha una grande importanza l'utilizzo dei *Related Videos* assieme ad alcuni concetti (tra cui ad esempio la co-occorrenza) trattati

nel Capitolo Stato dell'Arte.

Dal punto di vista dell'espansione, invece, è stato necessario ricorrere ad altre sorgenti di dati, ed in particolare effettuare l'integrazione con il Semantic Web. Da qui è nata l'idea di lavorare con le informazioni strutturate di Wikipedia (DBpedia) ed utilizzate toolkit di ausilio grazie ai quali si riesce a capire il grado di correlazione semantica tra entità diverse. Nel corso della trattazione, è emerso come DBpedia non sia uno strumento molto utile per quanto concerne l'incremento dello spazio iniziale dei tag. Lo stesso ragionamento non può essere invece fatto per Wikipedia Miner, che costituisce la pietra miliare dell'intero sistema di accrescimento delle etichette.

Capitolo 5

Risultati sperimentali

In questa sezione presenteremo le prove sperimentali fatte al fine di valutare quantitativamente le prestazioni offerte dal software descritto al capitolo precedente. I risultati saranno divisi a seconda della tipologia di esperimento; in particolare andremo ad analizzare la bontà dell'algoritmo rispetto alle seguenti parti:

- **filtraggio** degli *user-defined* tag nei video presenti in YouTube;
- **espansione** dei tag validati nella fase di filtraggio;
- procedura di matching ***Tag to Concept*** tra i termini di ricerca (creati a partire dai tag validati) e le risorse DBpedia;
- espansione dei concetti di LSCOM rispetto al dataset **TRECVID 2005**.

Tutte le prove sperimentali sono state svolte su un notebook dotato di processore Intel Core i7-720QM con clock a 1.60 GHz e memoria RAM con capacità pari a 8 GB.

5.1 Filtraggio ed Espansione degli user-defined tag nei video di YouTube

In questa sezione si analizzano le prestazioni del metodo creato rispetto alla prima fase: la procedura di valutazione della "bontà" dei tag inseriti dagli autori dei video residenti in YouTube[1]. Grazie a questa procedura, viene mantenuto solo un sottoinsieme dei tag originali, come descritto ampiamente nel paragrafo 4.2.2.

Per svolgere le prove sperimentali è stato necessario creare un dataset di riferimento, contenente video residenti in YouTube. L'idea di base è stata quella di analizzare le prestazioni dell'algoritmo abbracciando tutte le 15 categorie di YouTube: *Auto & Vehicles*, *Comedy*, *Education*, *Entertainment*, *Film & Animation*, *Gaming*, *Howto & Style*, *Music*, *News & Politics*, *Nonprofits & Activism*, *Pets & Animals*, *Science & Technology*, *Sports*, *Travel & Events*. Per ogni categoria sono stati scelti quattro video, valutando "manualmente" se i tag ivi presenti si prestassero bene alla procedura di filtraggio e relativa espansione. In altre parole, sono stati scelti dei video dotati di tag buoni e tag palesemente da eliminare. Questa scelta è dettata da una motivazione ben precisa. Se vengono elaborati video nei quali i tag non sono affatto validi (tag tutti errati, tag in lingue non europee, etc...), la successiva fase di espansione non potrà garantire risultati soddisfacenti in quanto è improbabile poter espandere annotazioni già palesemente errate. I video scelti, quindi, hanno mediamente almeno 10 tag al seguito, dei quali una parte è sicuramente da mantenere.

Le prestazioni del metodo creato vengono evidenziate rispetto ad ogni categoria di YouTube dopo di che faremo una valutazione complessiva dell'intero dataset (60 video). Prima di entrare nel dettaglio di ogni categoria, facciamo un esempio di prova di elaborazione su uno dei sessanta video presi in conside-

[1] http://www.youtube.com/

razione. Supponiamo di analizzare un filmato della categoria *Pets & Animals*, mostrato in Figura 5.1. Come si può vedere, il filmato contiene otto tag: *horse*, *ride*, *derby*, *race*, *ranch*, *wild*, *west*, *corral*. Di questi, solo i tag *horse*, *ranch*, *wild*, *corral* sono da considerarsi validi.

L'algoritmo opera valutando l'affidabilità delle singole annotazioni; per avere un'istantanea di come il software lavora, si consideri la Figura 5.2. Come si può vedere, il software attribuisce un punteggio (compreso in [0;1]) ad ogni annotazione, filtrando tutti i tag ad eccezione dei tre che vengono mantenuti: *horse*, *ranch*, *wild*. In questo caso, quindi, l'accuratezza raggiunta è pari a 0.875 (sette operazioni corrette su otto). L'algoritmo implementato effettua anche una procedura di espansione dei tag validati nella fase di filtraggio.

A partire dai tre tag mantenuti (*horse*, *ranch*, *wild*), si cercano nuovi termini che vanno a comporre il set finale delle annotazioni. Per rendere più chiare le modalità operative del software, si consideri la Figura 5.3 . Come si può vedere, vengono aggiunti ai tag originali altri otto termini, dei quali cinque possono essere considerati buoni (sufficientemente validi): *horses*, *equine*, *racehorse*, *rancher*, *cattle raising*.

A questo punto, dopo aver dato un'idea circa le operazioni svolte dal software e le modalità di valutazione, possiamo specificare nel dettaglio i risultati sperimentali sul dataset da 60 video, analizzando per ora le varie categorie una ad una. Prima di presentare le tabelle ed i grafici di riferimento, chiariamo la metrica di valutazione delle prestazioni dell'algoritmo. Sia il filtraggio dei tag che la relativa espansione, vengono valutati rispetti ad una misura di *Accuracy*. In particolare, essa è calcolata come il rapporto tra il numero di operazioni eseguite correttamente su ogni tag ed il totale dei tag stessi.

Nell'esempio precedente, abbiamo un Accuracy pari a 0.875 in quanto vengono effettuate complessivamente sette operazioni corrette su otto. La fase di

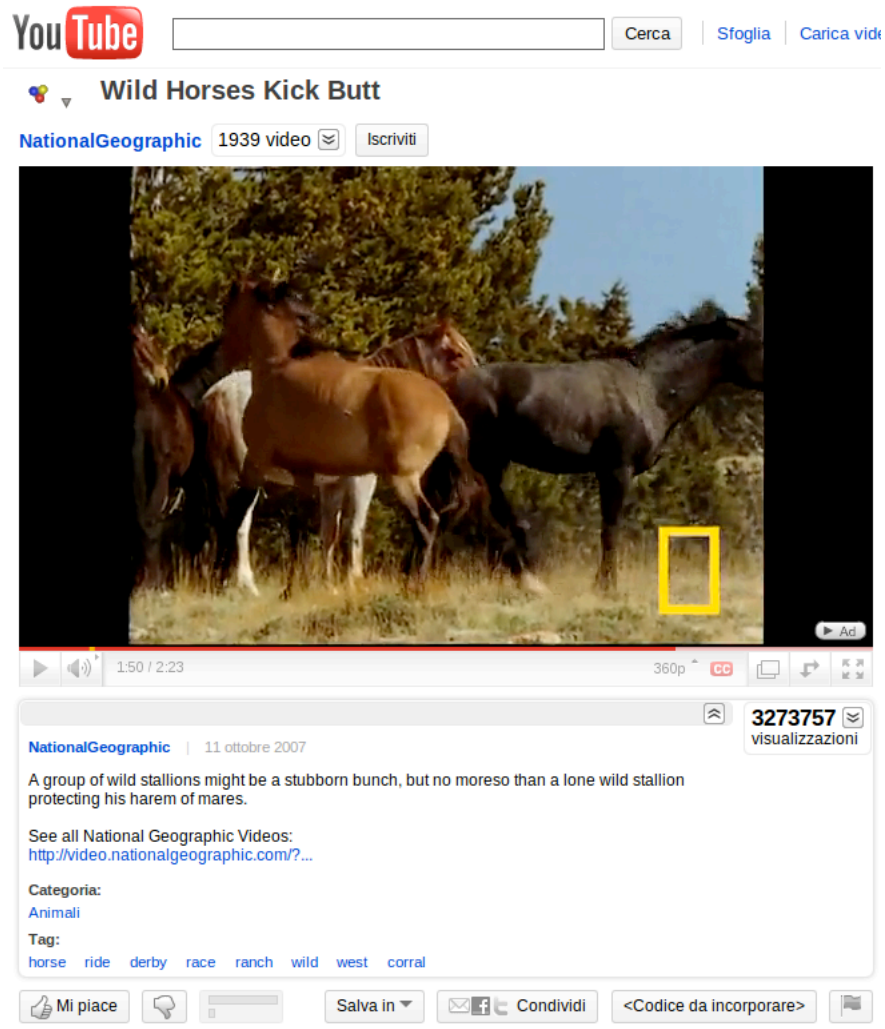

Figura 5.1: Filmato di esempio "Horses": categoria *Pets & Animals*.

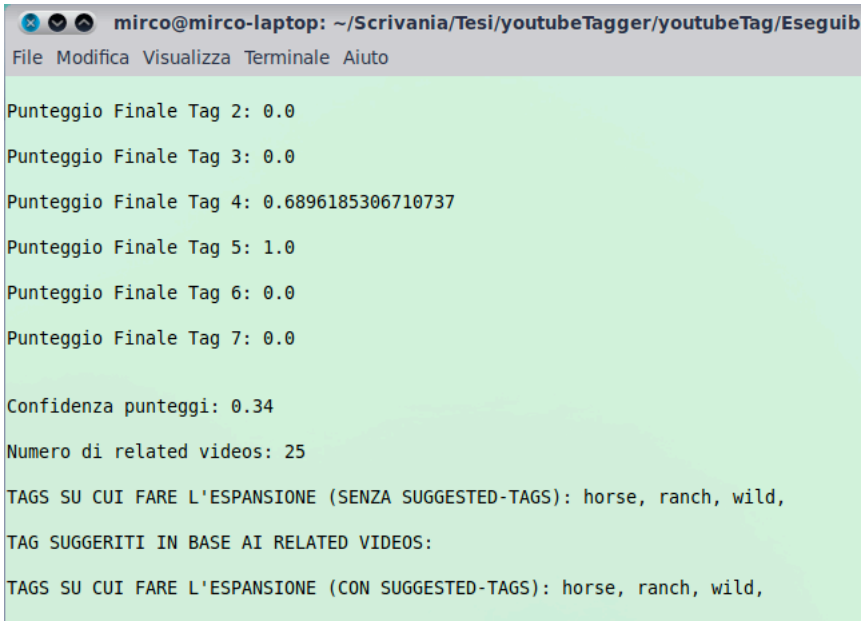

Figura 5.2: Filmato di esempio "Horses" - Fase di filtraggio degli user-defined tag: screenshot del software.

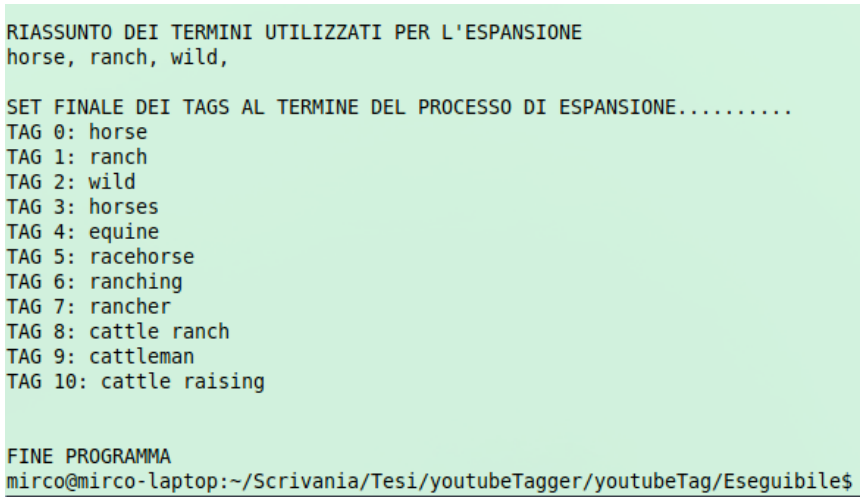

Figura 5.3: Screenshot video di esempio "Horses": espansione degli user-defined tag (visualizzazione del set finale delle annotazioni).

filtraggio e di espansione dei tag è stata svolta calcolando l'Accuracy in funzione di sette diversi valori di *Threshold*. Quest'ultimo è la soglia al di sotto della quale il tag viene filtrato e corrisponde al termine $thresh_{tag}$ nell'espressione (4.6) del paragrafo 4.2.2.

Gli esperimenti svolti su diversi valori di threshold hanno permesso di individuare la *soglia ottimale*, da utilizzare in generale su tutti i video residenti in YouTube. Siamo pronti quindi per analizzare nel dettaglio le prestazioni del sistema. Per avere un'idea circa il contenuto dei filmati analizzati, fare riferimento alla didascalia di tutte le prossime tabelle, dove viene fornito un titolo simbolico di ogni video. La valutazione circa i risultati raggiunti dal metodo sarà effettuata dopo la presentazione dei grafici relativi ad ogni categoria di YouTube e dei due grafici finali che ne sintetizzano le prestazioni complessive sull'intero dataset.

Partiamo con l'analisi della categoria "Travel & Events" (si tenga conto che l'ordine in cui sono inseriti i risultati non ricalca l'ordine alfabetico delle categorie). La Tabella 5.1 contiene i valori di Accuracy per la fase di **filtraggio** e di **espansione** in funzione dei sette valori di soglia (da 0.1 a 0.7). Il grafico corrispondente è mostrato invece in Figura 5.4. Si noti che l'accuratezza relativa ad ogni *Threshold* consiste nella media dei quattro filmati presi in considerazione. Per la fase di espansione, inoltre, la misura di *Accuracy* consiste naturalmente nel rapporto tra il numero di tag corretti suggeriti ed il numero totale di tag suggeriti. La Tabella 5.2 analizza il filtraggio e l'espanzione della categoria "Auto & Vehicles". Il relativo grafico è mostrato in Figura 5.5. Per la categoria "Science & Technology", si consideri invece la Tabella 5.3. Il grafico corrispondente è mostrato in Figura 5.6. Le prestazioni relative alla categoria "Comedy" sono in Tabella 5.4. Il grafico relativo a questo esperimento è mostrato invece in Figura 5.7. I risultati della categoria "Education" sono visibili in

	Video1	Video2	Video3	Video4	Media
Acc. Filtraggio 0.1	0.875	0.5	0.846	0.909	0.783
Acc. Espansione 0.1	0.875	1.0	0.4	0.9	0.794
Acc. Filtraggio 0.2	0.75	0.5	0.769	0.909	0.732
Acc. Espansione 0.2	0.833	1.0	0.45	0.875	0.79
Acc. Filtraggio 0.3	0.75	0.5	0.769	0.818	0.709
Acc. Espansione 0.3	0.833	1.0	0.45	0.875	0.79
Acc. Filtraggio 0.4	0.625	0.5	0.846	0.727	0.675
Acc. Espansione 0.4	1.0	1.0	0.5	0.875	0.844
Acc. Filtraggio 0.5	0.5	0.5	0.654	0.636	0.573
Acc. Espansione 0.5	1.0	1.0	0.6	0.714	0.829
Acc. Filtraggio 0.6	0.5	0.5	0.615	0.546	0.54
Acc. Espansione 0.6	1.0	1.0	0.667	0.714	0.845
Acc. Filtraggio 0.7	0.5	0.25	0.615	0.546	0.478
Acc. Espansione 0.7	1.0	1.0	0.667	0.714	0.845

Tabella 5.1: Accuracy Filtraggio ed Espansione Tag categoria *Travel & Events*. Video1: "Cascate" - Video2: "Madrid" - Video3: "Marsa Alam" - Video4: "Colosseo".

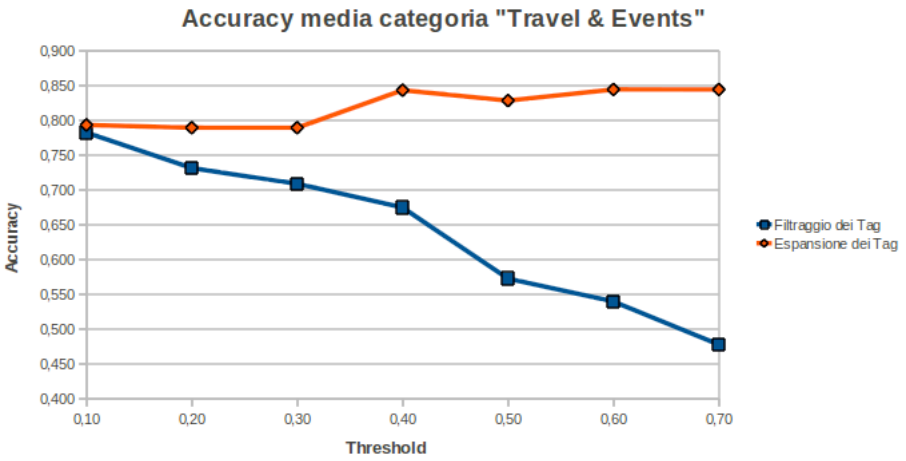

Figura 5.4: Accuracy media categoria *Travel & Events*.

	Video1	Video2	Video3	Video4	Media
Acc. Filtraggio 0.1	0.885	0.947	0.615	0.8	0.812
Acc. Espansione 0.1	0.581	0.643	0.5	1.0	0.681
Acc. Filtraggio 0.2	0.808	0.947	0.923	0.8	0.867
Acc. Espansione 0.2	0.581	0.714	1.0	1.0	0.824
Acc. Filtraggio 0.3	0.615	0.947	0.923	0.8	0.821
Acc. Espansione 0.3	0.79	0.714	1.0	1.0	0.876
Acc. Filtraggio 0.4	0.462	0.947	0.923	0.8	0.783
Acc. Espansione 0.4	0.818	0.714	1.0	1.0	0.883
Acc. Filtraggio 0.5	0.462	0.947	0.923	0.8	0.783
Acc. Espansione 0.5	0.778	0.714	1.0	1.0	0.783
Acc. Filtraggio 0.6	0.462	0.947	0.923	0.8	0.783
Acc. Espansione 0.6	0.778	0.714	1.0	1.0	0.783
Acc. Filtraggio 0.7	0.385	0.947	0.923	0.8	0.764
Acc. Espansione 0.7	0.571	0.714	1.0	1.0	0.821

Tabella 5.2: Accuracy Filtraggio ed Espansione Tag categoria *Auto & Vehicles*. Video1: "PlaneMaarten" - Video2: "FerrariCrash" - Video3: "Boeing" - Video4: "Tractor".

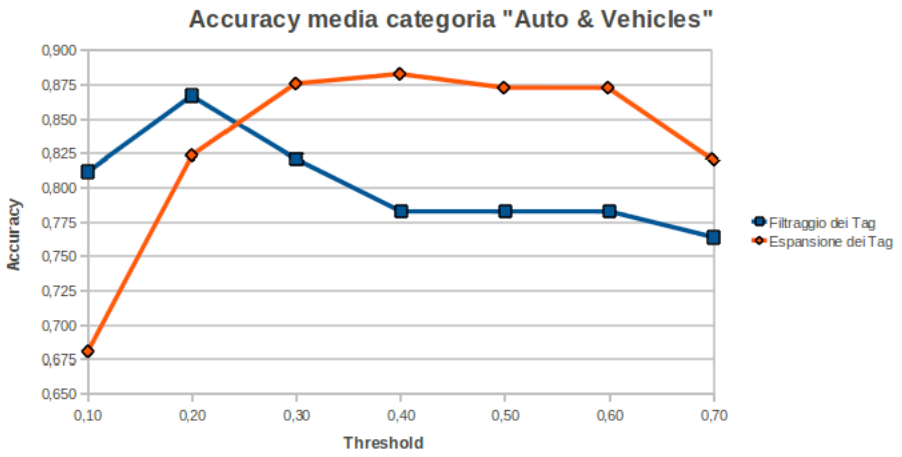

Figura 5.5: Accuracy media categoria *Auto & Vehicles*.

	Video1	Video2	Video3	Video4	Media
Acc. Filtraggio 0.1	0.846	0.667	1.0	0.737	0.813
Acc. Espansione 0.1	0.688	0.444	1.0	0.222	0.589
Acc. Filtraggio 0.2	0.846	0.667	1.0	0.79	0.826
Acc. Espansione 0.2	0.667	0.571	1.0	0.625	0.716
Acc. Filtraggio 0.3	0.692	0.5	1.0	0.895	0.772
Acc. Espansione 0.3	0.546	0.6	1.0	0.867	0.753
Acc. Filtraggio 0.4	0.615	0.417	1.0	0.842	0.719
Acc. Espansione 0.4	0.625	0.5	1.0	1.0	0.781
Acc. Filtraggio 0.5	0.615	0.417	1.0	0.684	0.679
Acc. Espansione 0.5	0.625	0.5	1.0	1.0	0.781
Acc. Filtraggio 0.6	0.539	0.417	1.0	0.684	0.66
Acc. Espansione 0.6	0.333	0.5	1.0	1.0	0.708
Acc. Filtraggio 0.7	0.539	0.417	1.0	0.684	0.66
Acc. Espansione 0.7	0.333	0.6	1.0	1.0	0.733

Tabella 5.3: Accuracy Filtraggio ed Espansione Tag categoria *Science & Technology*. Video1: "Helicopter" - Video2: "Airbus" - Video3: "Nokia" - Video4: "Iphone4G".

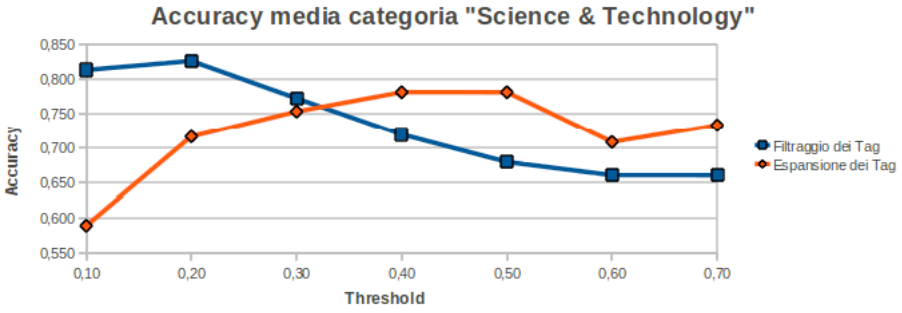

Figura 5.6: Accuracy media categoria *Science & Technology*.

	Video1	Video2	Video3	Video4	Media
Acc. Filtraggio 0.1	1.0	0.667	0.875	0.833	0.844
Acc. Espansione 0.1	0.556	0.4	0.235	1.0	0.548
Acc. Filtraggio 0.2	1.0	0.667	0.875	0.833	0.844
Acc. Espansione 0.2	0.556	0.375	0.289	1.0	0.555
Acc. Filtraggio 0.3	1.0	0.667	0.688	0.833	0.797
Acc. Espansione 0.3	0.556	0.375	0.289	1.0	0.555
Acc. Filtraggio 0.4	1.0	0.778	0.563	0.667	0.794
Acc. Espansione 0.4	0.556	0.6	0.5	1.0	0.664
Acc. Filtraggio 0.5	1.0	0.667	0.563	0.667	0.724
Acc. Espansione 0.5	0.556	0.6	0.5	1.0	0.664
Acc. Filtraggio 0.6	0.833	0.778	0.563	0.667	0.71
Acc. Espansione 0.6	0.714	0.6	0.5	1.0	0.704
Acc. Filtraggio 0.7	0.833	0.778	0.563	0.333	0.627
Acc. Espansione 0.7	0.714	0.6	0.5	1.0	0.704

Tabella 5.4: Accuracy Filtraggio ed Espansione Tag categoria *Comedy*. Video1: "GoalCelebration" - Video2: "FunnyFootball" - Video3: "StatueLiberty" - Video4: "TalkingCat".

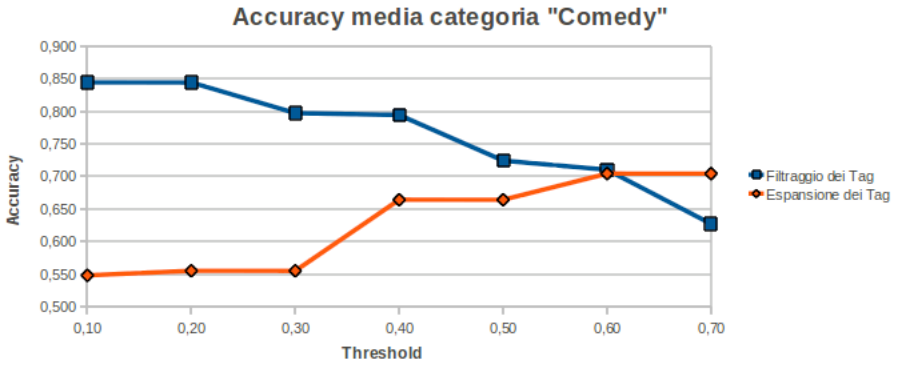

Figura 5.7: Accuracy media categoria *Comedy*.

	Video1	Video2	Video3	Video4	Media
Acc. Filtraggio 0.1	0.846	0.556	0.875	0.905	0.796
Acc. Espansione 0.1	0.292	0.455	0.6	0.476	0.456
Acc. Filtraggio 0.2	0.846	0.444	0.875	0.905	0.768
Acc. Espansione 0.2	0.5	0.5	0.667	0.808	0.619
Acc. Filtraggio 0.3	0.846	0.556	0.75	0.762	0.729
Acc. Espansione 0.3	0.5	0.5	1.0	0.824	0.706
Acc. Filtraggio 0.4	0.462	0.667	0.75	0.619	0.625
Acc. Espansione 0.4	0.286	0.5	1.0	0.8	0.647
Acc. Filtraggio 0.5	0.462	1.0	0.75	0.476	0.672
Acc. Espansione 0.5	0.286	0.7	1.0	0.889	0.719
Acc. Filtraggio 0.6	0.462	0.778	0.75	0.476	0.617
Acc. Espansione 0.6	0.286	0.667	1.0	0.889	0.711
Acc. Filtraggio 0.7	0.385	0.778	0.75	0.429	0.586
Acc. Espansione 0.7	0.222	0.667	1.0	0.765	0.664

Tabella 5.5: Accuracy Filtraggio ed Estensione Tag categoria *Education*. Video1: "GraduationDaily" - Video2: "Oxford" - Video3: "Princeton" - Video4: "ColosseumRome".

Tabella 5.5. La Figura 5.8 mostra l'andamento del valore di Accuracy rispetto alle diverse Threshold. Le prestazioni relative alla categoria "Pets & Animals" sono visualizzate in Tabella 5.6 ed in Figura 5.9. La categoria "People & Blogs" è evidenziata in Tabella 5.7 ed in Figura 5.10. Per quanto concerne la categoria "Howto & Style" si consideri la Tabella 5.8 e la Figura 5.11. I risultati sperimentali ottenuti per la categoria "Sports" sono mostrati in Tabella 5.9 ; con il grafico di Figura 5.12 è possibile invece prendere nota dei valori di Accuracy in funzione

delle possibile soglie sul punteggio. La categoria "Nonprofits & Activism" è analizzata in Tabella 5.10

ed in Figura 5.13. La categoria successiva da prendere in considerazione è "News & Politics", le cui rilevanze sperimentali sono mostrate in Tabella 5.11 ed

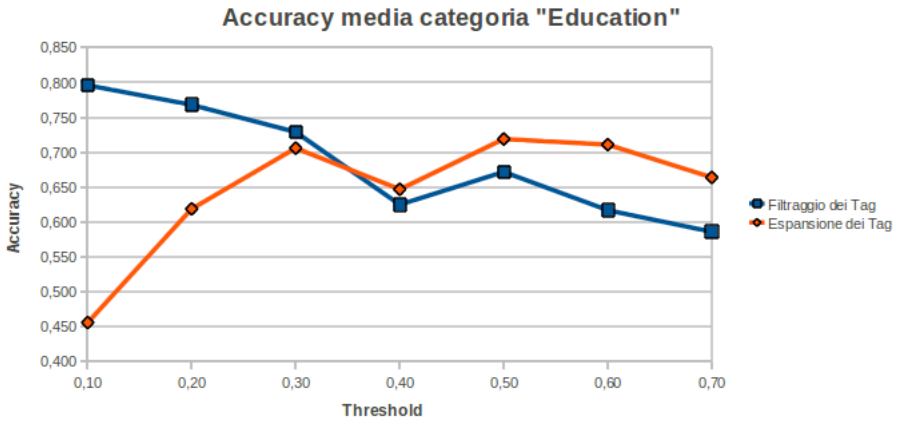

Figura 5.8: Accuracy media categoria *Education*.

	Video1	**Video2**	**Video3**	**Video4**	**Media**
Acc. Filtraggio 0.1	1.0	0.625	0.778	0.546	0.737
Acc. Espansione 0.1	0.2	0.385	0.429	0.2	0.304
Acc. Filtraggio 0.2	1.0	0.875	0.667	0.546	0.772
Acc. Espansione 0.2	0.2	0.75	0.375	0.2	0.381
Acc. Filtraggio 0.3	1.0	0.875	0.778	0.636	0.822
Acc. Espansione 0.3	0.2	0.625	0.429	0.333	0.397
Acc. Filtraggio 0.4	1.0	0.875	0.778	1.0	0.913
Acc. Espansione 0.4	0.2	0.625	0.429	0.333	0.397
Acc. Filtraggio 0.5	1.0	0.875	0.667	1.0	0.886
Acc. Espansione 0.5	0.2	0.625	0.6	0.333	0.44
Acc. Filtraggio 0.6	1.0	0.75	0.667	1.0	0.854
Acc. Espansione 0.6	0.2	0.667	0.6	0.333	0.45
Acc. Filtraggio 0.7	1.0	0.75	0.667	1.0	0.854
Acc. Espansione 0.7	0.2	0.667	0.6	0.333	0.45

Tabella 5.6: Accuracy Filtraggio ed Espansione Tag categoria *Pets & Animals*. Video1: “JackRussell” - Video2: “Horses” - Video3: “Pinscher” - Video4: “Cat”.

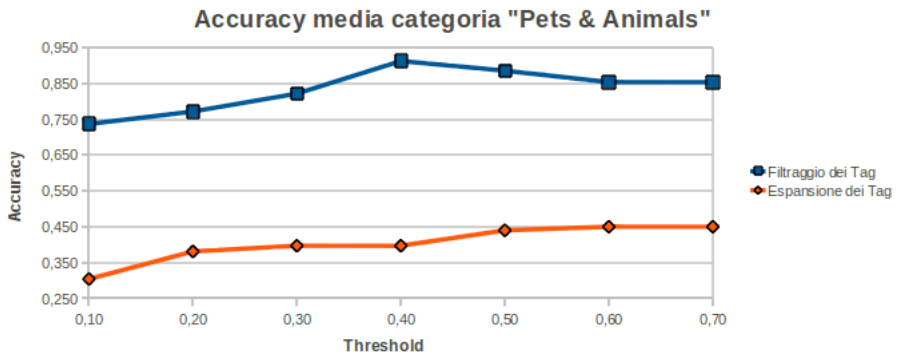

Figura 5.9: Accuracy media categoria *Pets & Animals*.

	Video1	**Video2**	**Video3**	**Video4**	**Media**
Acc. Filtraggio 0.1	0.875	1.0	0.75	1.0	0.906
Acc. Espansione 0.1	0.667	0.857	0.667	0.25	0.61
Acc. Filtraggio 0.2	0.875	1.0	0.875	1.0	0.938
Acc. Espansione 0.2	0.571	0.857	0.667	0.25	0.586
Acc. Filtraggio 0.3	1.0	1.0	0.875	1.0	0.969
Acc. Espansione 0.3	0.571	0.857	0.667	0.25	0.586
Acc. Filtraggio 0.4	1.0	0.875	0.875	1.0	0.938
Acc. Espansione 0.4	0.714	0.857	0.667	0.25	0.622
Acc. Filtraggio 0.5	1.0	0.875	0.75	1.0	0.906
Acc. Espansione 0.5	0.714	0.857	0.5	0.25	0.58
Acc. Filtraggio 0.6	1.0	0.875	0.875	1.0	0.938
Acc. Espansione 0.6	0.714	0.857	1.0	0.25	0.705
Acc. Filtraggio 0.7	1.0	0.875	0.875	1.0	0.938
Acc. Espansione 0.7	0.714	0.857	1.0	0.25	0.705

Tabella 5.7: Accuracy Filtraggio ed Espansione Tag categoria *People & Blogs*. Video1: “OctopusPaul” - Video2: “Dalai Lama” - Video3: “Wedding” - Video4: “Communion”.

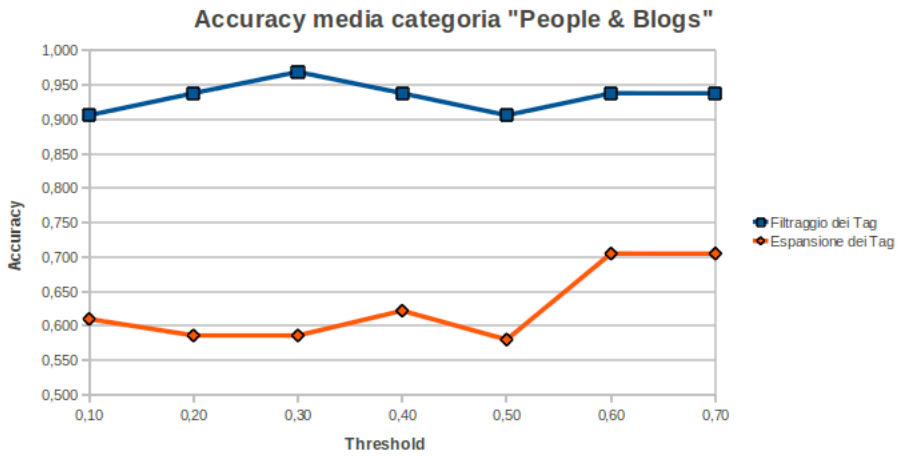

Figura 5.10: Accuracy media categoria *People & Blogs*.

	Video1	**Video2**	**Video3**	**Video4**	**Media**
Acc. Filtraggio 0.1	0.818	1.0	0.667	0.857	0.836
Acc. Espansione 0.1	0.5	0.667	0.5	0.364	0.508
Acc. Filtraggio 0.2	0.818	1.0	0.667	0.857	0.836
Acc. Espansione 0.2	0.571	0.667	0.5	0.417	0.539
Acc. Filtraggio 0.3	0.818	1.0	0.667	0.786	0.818
Acc. Espansione 0.3	0.571	0.667	0.5	0.375	0.528
Acc. Filtraggio 0.4	0.818	1.0	0.667	0.857	0.836
Acc. Espansione 0.4	0.571	0.667	0.5	0.6	0.585
Acc. Filtraggio 0.5	0.818	1.0	0.667	0.857	0.836
Acc. Espansione 0.5	0.571	0.667	0.5	0.6	0.585
Acc. Filtraggio 0.6	0.909	1.0	0.667	0.857	0.858
Acc. Espansione 0.6	0.667	0.667	0.5	0.6	0.609
Acc. Filtraggio 0.7	0.727	1.0	0.667	0.857	0.813
Acc. Espansione 0.7	1.0	0.667	0.5	0.6	0.692

Tabella 5.8: Accuracy Filtraggio ed Espansione Tag categoria *Howto & Style*. Video1: "ItalianSoup" - Video2: "IpodNano" - Video3: "Rossetto" - Video4: "Steak".

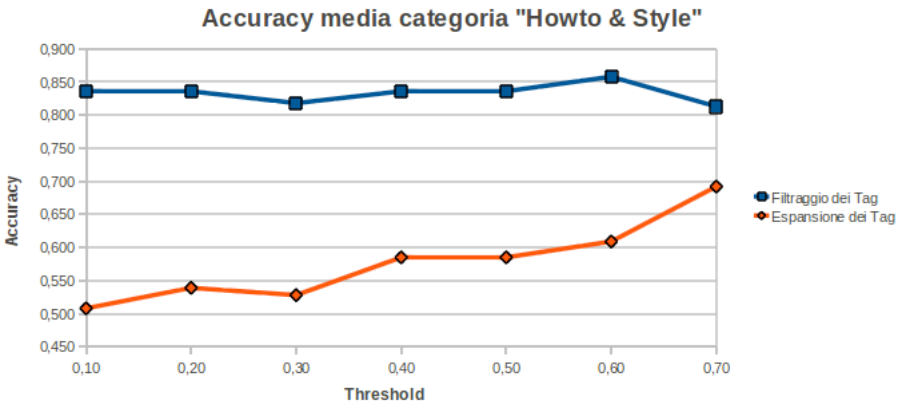

Figura 5.11: Accuracy media categoria *Howto & Style*.

	Video1	**Video2**	**Video3**	**Video4**	**Media**
Acc. Filtraggio 0.1	0.867	1.0	0.75	0.667	0.821
Acc. Espansione 0.1	0.625	1.0	0.5	1.0	0.781
Acc. Filtraggio 0.2	0.933	1.0	0.75	0.667	0.838
Acc. Espansione 0.2	0.533	1.0	0.5	1.0	0.758
Acc. Filtraggio 0.3	0.867	1.0	0.75	0.667	0.821
Acc. Espansione 0.3	0.571	1.0	0.5	1.0	0.768
Acc. Filtraggio 0.4	0.667	0.8	0.75	0.333	0.638
Acc. Espansione 0.4	0.667	1.0	0.5	1.0	0.792
Acc. Filtraggio 0.5	0.533	0.8	0.75	0.333	0.604
Acc. Espansione 0.5	0.6	1.0	0.5	1.0	0.775
Acc. Filtraggio 0.6	0.533	0.8	0.75	0.0	0.521
Acc. Espansione 0.6	0.6	1.0	0.5	—	0.7
Acc. Filtraggio 0.7	0.467	0.8	0.75	0.0	0.504
Acc. Espansione 0.7	0.667	1.0	0.5	—	0.722

Tabella 5.9: Accuracy Filtraggio ed Espansione Tag categoria *Sports*. Video1: “Formula1” - Video2: “Tyson” - Video3: “Wimbledon” - Video4: “Surf”.

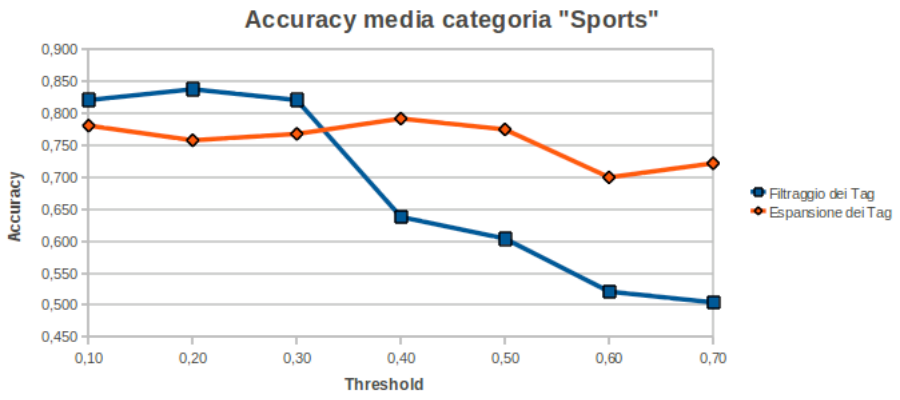

Figura 5.12: Accuracy media categoria *Sports*.

	Video1	**Video2**	**Video3**	**Video4**	**Media**
Acc. Filtraggio 0.1	0.769	0.9	0.556	0.667	0.723
Acc. Espansione 0.1	0.636	0.857	0.5	—	0.664
Acc. Filtraggio 0.2	0.923	0.9	0.667	0.667	0.789
Acc. Espansione 0.2	0.75	0.455	0.6	—	0.602
Acc. Filtraggio 0.3	0.923	0.9	0.667	0.667	0.789
Acc. Espansione 0.3	0.75	0.455	0.6	—	0.602
Acc. Filtraggio 0.4	0.846	0.8	0.667	0.667	0.745
Acc. Espansione 0.4	0.818	0.455	0.5	—	0.591
Acc. Filtraggio 0.5	0.692	0.6	0.778	0.667	0.684
Acc. Espansione 0.5	0.714	0.455	0.0	—	0.39
Acc. Filtraggio 0.6	0.692	0.4	0.778	0.667	0.634
Acc. Espansione 0.6	0.714	0.333	0.0	—	0.349
Acc. Filtraggio 0.7	0.769	0.4	0.778	0.667	0.654
Acc. Espansione 0.7	0.714	0.333	0.0	—	0.349

Tabella 5.10: Accuracy Filtraggio ed Espansione Tag categoria *Nonprofits & Activism*. Video1: "GazaProtestBoston" - Video2: "UnicefChild" - Video3: "UnicefAfrica" - Video4: "PopoloViola".

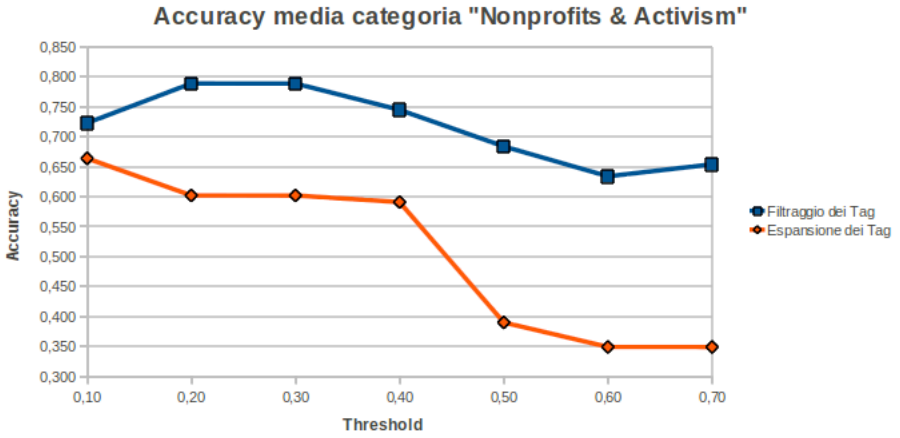

Figura 5.13: Accuracy media categoria *Nonprofits & Activism*.

	Video1	**Video2**	**Video3**	**Video4**	**Media**
Acc. Filtraggio 0.1	1.0	1.0	1.0	0.667	0.917
Acc. Espansione 0.1	0.714	0.438	0.75	0.267	0.542
Acc. Filtraggio 0.2	1.0	1.0	1.0	0.762	0.941
Acc. Espansione 0.2	0.714	0.438	0.75	0.231	0.533
Acc. Filtraggio 0.3	1.0	1.0	1.0	0.81	0.953
Acc. Espansione 0.3	0.714	0.438	0.75	1.0	0.726
Acc. Filtraggio 0.4	1.0	1.0	1.0	0.81	0.953
Acc. Espansione 0.4	0.714	0.438	0.75	1.0	0.726
Acc. Filtraggio 0.5	0.8	0.833	1.0	0.81	0.861
Acc. Espansione 0.5	0.714	0.875	0.75	1.0	0.835
Acc. Filtraggio 0.6	0.4	0.667	0.8	0.762	0.657
Acc. Espansione 0.6	1.0	0.833	0.714	1.0	0.887
Acc. Filtraggio 0.7	0.4	0.5	0.8	0.762	0.616
Acc. Espansione 0.7	1.0	0.667	0.714	1.0	0.845

Tabella 5.11: Accuracy Filtraggio ed Espansione Tag categoria *News & Politics*. Video1: "TsunamiPhuket" - Video2: "FireDisaster" - Video3: "G20London" - Video4: "NeveMilano".

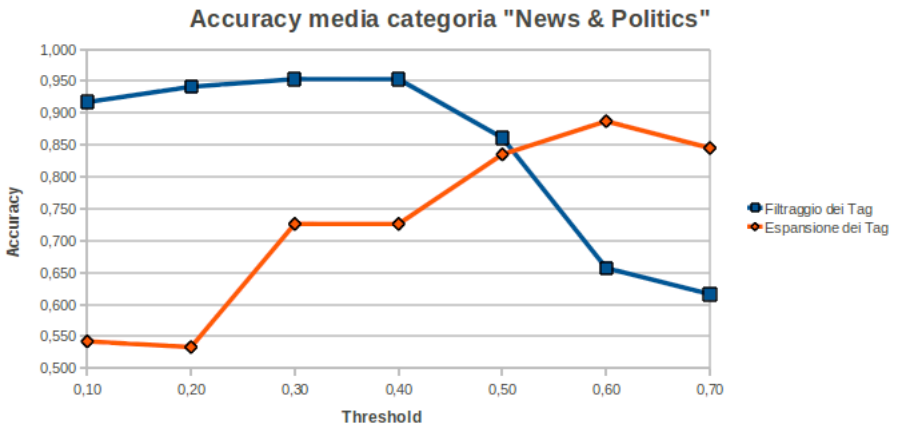

Figura 5.14: Accuracy media categoria *News & Politics*.

in Figura 5.14. Per quanto concerne invece la categoria "Film & Animation", si consideri la Tabella 5.12

e la Figura 5.15. Continuando questa serie di tabelle e grafici, si può passare alla categoria "Gaming", evidenziata in Tabella 5.13 e tramite la Figura 5.16. Per completare le 15 categorie alle quali appartengono i filmati di YouTube, mancano le categorie "Music" e "Entertainment". Le valutazioni sperimentali di queste due categorie sono rintracciabili rispettivamente nella Tabella 5.14 , nella Figura 5.17 , nella Tabella 5.15 e nella Figura 5.18.

Facciamo a questo punto un breve resoconto dei risultati ottenuti. La prima nota da fare è relativa all'andamento generale della curva di filtraggio. Come si può vedere dai grafici, all'aumentare della *Threshold*, l'accuratezza di filtraggio tende a diminuire. Il ragionamento opposto può invece essere fatto nel caso della curva di espansione. Esistono comunque alcuni casi particolari. Per la curva di filtraggio, si noti la presenza di un'anomalia nelle categorie *People & Blogs*, *Pets & Animals*, *Howto & Style* e *Music*.

Relativamente alla categoria Pets & Animals, l'andamento anomalo della curva è dovuto alla presenza di tag poco significativi che però vengono classificati

	Video1	Video2	Video3	Video4	Media
Acc. Filtraggio 0.1	0.824	0.7	0.813	0.941	0.82
Acc. Espansione 0.1	0.412	0.75	0.75	0.412	0.581
Acc. Filtraggio 0.2	0.941	0.7	0.813	0.941	0.849
Acc. Espansione 0.2	0.727	0.5	0.765	0.412	0.601
Acc. Filtraggio 0.3	0.882	0.7	0.75	0.941	0.818
Acc. Espansione 0.3	0.5	0.5	0.765	0.412	0.544
Acc. Filtraggio 0.4	0.765	0.8	0.563	0.941	0.767
Acc. Espansione 0.4	0.7	1.0	0.733	0.412	0.711
Acc. Filtraggio 0.5	0.647	0.6	0.375	0.882	0.626
Acc. Espansione 0.5	0.625	1.0	0.714	0.467	0.702
Acc. Filtraggio 0.6	0.588	0.6	0.25	0.412	0.463
Acc. Espansione 0.6	0.833	1.0	0.692	0.333	0.715
Acc. Filtraggio 0.7	0.529	0.6	0.313	0.412	0.464
Acc. Espansione 0.7	1.0	1.0	0.9	0.333	0.808

Tabella 5.12: Accuracy Filtraggio ed Estensione Tag categoria *Film & Animation*. Video1: "NewMoon" - Video2: "Avatar" - Video3: "MrFox" - Video4:"Twilight".

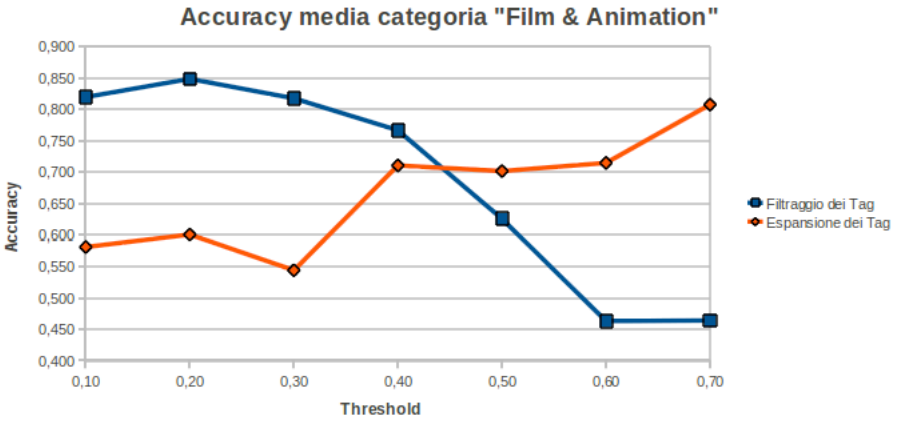

Figura 5.15: Accuracy media categoria *Film & Animation*.

	Video1	Video2	Video3	Video4	Media
Acc. Filtraggio 0.1	0.778	0.75	0.75	0.875	0.788
Acc. Espansione 0.1	0.4	0.0	0.267	0.833	0.375
Acc. Filtraggio 0.2	0.778	0.917	0.667	1.0	0.841
Acc. Espansione 0.2	0.3	0.143	0.667	0.833	0.486
Acc. Filtraggio 0.3	0.778	0.917	0.917	0.875	0.872
Acc. Espansione 0.3	0.3	0.143	0.6	1.0	0.511
Acc. Filtraggio 0.4	0.667	0.833	0.867	0.625	0.748
Acc. Espansione 0.4	0.125	0.778	0.6	0.8	0.576
Acc. Filtraggio 0.5	0.611	0.583	0.867	0.5	0.64
Acc. Espansione 0.5	0.143	0.0	0.6	0.667	0.353
Acc. Filtraggio 0.6	0.667	0.583	0.867	0.5	0.654
Acc. Espansione 0.6	0.143	0.0	0.571	0.667	0.345
Acc. Filtraggio 0.7	0.722	0.667	0.867	0.5	0.689
Acc. Espansione 0.7	0.2	—	0.571	0.667	0.479

Tabella 5.13: Accuracy Filtraggio ed Espansione Tag categoria *Gaming*. Video1: "NeedForSpeed" - Video2: "Warcraft" - Video3: "SimCity" - Video4: "Assassin".

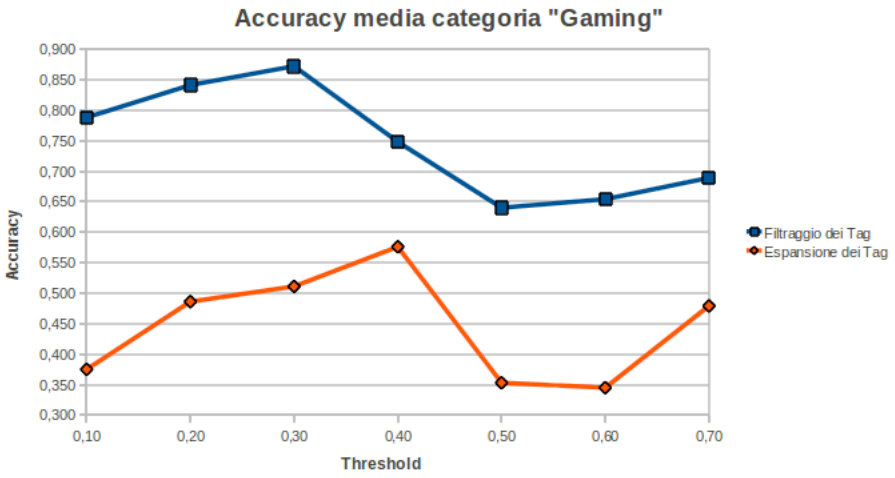

Figura 5.16: Accuracy media categoria *Gaming*.

	Video1	Video2	Video3	Video4	Media
Acc. Filtraggio 0.1	0.813	0.7	0.441	0.75	0.676
Acc. Espansione 0.1	0.556	0.375	0.098	0.4	0.357
Acc. Filtraggio 0.2	0.875	0.55	0.382	0.875	0.671
Acc. Espansione 0.2	0.2	0.5	0.195	0.6	0.374
Acc. Filtraggio 0.3	0.875	0.7	0.382	0.875	0.708
Acc. Espansione 0.3	0.308	0.5	0.195	0.6	0.401
Acc. Filtraggio 0.4	0.875	0.9	0.618	0.875	0.817
Acc. Espansione 0.4	0.417	0.727	0.211	0.6	0.489
Acc. Filtraggio 0.5	0.875	0.75	0.765	0.938	0.832
Acc. Espansione 0.5	0.417	0.364	0.2	0.571	0.388
Acc. Filtraggio 0.6	0.875	0.75	0.853	0.938	0.854
Acc. Espansione 0.6	0.417	0.364	0.133	0.571	0.371
Acc. Filtraggio 0.7	0.875	0.75	0.941	1.0	0.892
Acc. Espansione 0.7	0.417	0.364	0.182	0.714	0.419

Tabella 5.14: Accuracy Filtraggio ed Estensione Tag categoria *Music*. Video1: "Madonna" - Video2: "LadyGaga" - Video3: "LadyGaga2" - Video4: "Madonna2".

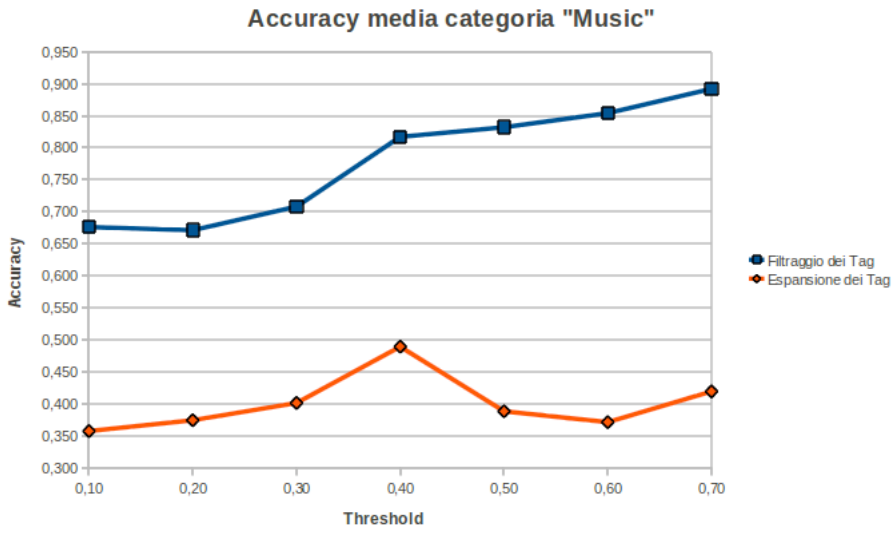

Figura 5.17: Accuracy media categoria *Music*.

	Video1	Video2	Video3	Video4	Media
Acc. Filtraggio 0.1	1.0	0.8	0.625	1.0	0.856
Acc. Espansione 0.1	0.5	0.5	1.0	0.8	0.7
Acc. Filtraggio 0.2	1.0	0.8	0.625	1.0	0.856
Acc. Espansione 0.2	0.5	0.5	1.0	0.7	0.675
Acc. Filtraggio 0.3	1.0	0.8	0.625	1.0	0.856
Acc. Espansione 0.3	0.5	0.5	1.0	0.7	0.675
Acc. Filtraggio 0.4	1.0	0.6	0.625	1.0	0.806
Acc. Espansione 0.4	0.5	0.667	1.0	0.7	0.717
Acc. Filtraggio 0.5	1.0	0.4	0.625	0.8	0.706
Acc. Espansione 0.5	0.5	0.333	1.0	0.571	0.601
Acc. Filtraggio 0.6	1.0	0.4	0.625	0.7	0.681
Acc. Espansione 0.6	0.5	0.333	1.0	0.571	0.601
Acc. Filtraggio 0.7	1.0	0.6	0.625	0.5	0.681
Acc. Espansione 0.7	0.5	1.0	1.0	0.667	0.792

Tabella 5.15: Accuracy Filtraggio ed Espansione Tag categoria *Entertainment*. Video1: "WWE" - Video2: "HelicopterCrash" - Video3: "MissUniverso" - Video4: "Saturday NightLive".

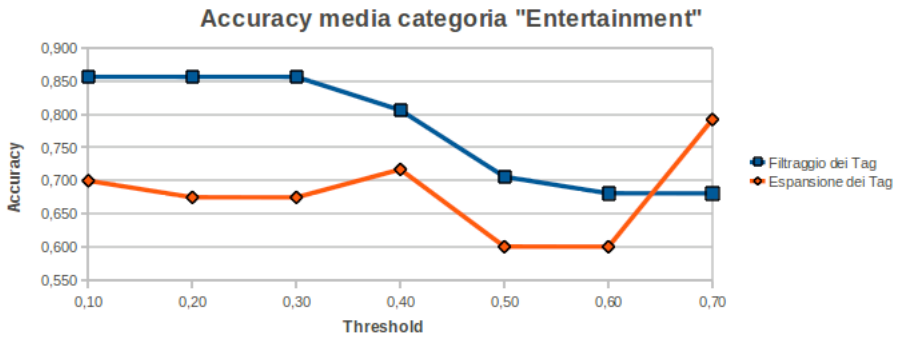

Figura 5.18: Accuracy media categoria *Entertainment*.

dall'algoritmo come importanti. Possiamo affermare senza ombra di dubbio che si tratta di un caso nel quale il metodo sviluppato non garantisce i risultati sperati, andando così a commettere degli errori nell'attribuzione del punteggio ai tag. Un ragionamento simile può essere fatto anche per la categoria *Music*.

Relativamente alla curva di espansione, invece, si può notare come l'accuratezza (nel caso generale) cresca per valori alti di *Threshold*. Tale risultanza ha una motivazione ben definita. Con valori di soglia elevati, vengono mantenuti in generale meno tag (che però sono assai "rilevanti") e questo consente di effettuare l'espansione su termini più significativi. Il risultato è che vengono aggiunti pochi tag, ma dotati di una rilevanza molto alta.

Una citazione deve essere fatta per la categoria *Nonprofits & Activism*. L'andamento anomalo della curva di espansione è dovuto alla presenza di due video "problematici" all'interno del dataset. In particolare, quest'ultimi sono dotati di pochi tag e per di più difficilmente "espandibili", in quanto sono termini non rintracciabili all'interno del database gestito con Wikipedia Miner (paragrafo 4.2.4.2). In questo caso, quindi, possiamo affermare che l'andamento casuale della curva non è attribuibile ad una scarsa efficienza dell'algoritmo sviluppato bensì ad una scelta "sfortunata" dei due video incriminati inseriti nel dataset.

Analizziamo quindi i risultati numetici ottenuti, per la fase di filtraggio e per quella di espansione. Nel primo caso, ci sono alcune categorie (es. *Auto & Vehicles*, *Comedy*, *People & Blogs*, *News & Politics*) nelle quali si riescono ad ottenere valori di Accuracy superiori a 80% (il massimo si ha per *People & Blogs*, dove si arriva a 96.9%), mentre in altre i risultati sono assai inferiori (il minimo si ha in *Film & Animation*, con Accuracy = 46.3%). Per quanto concerne l'espansione, invece, i picchi di accuratezza sono inferiori a quelli della fase di filtraggio. La motivazione risiede nella maggiore criticità di questa fase rispetto al caso del filtraggio; aggiungere delle annotazioni rilevanti, in linea livello

	0.10	**0.20**	**0.30**	**0.40**	**0.50**	**0.60**	**0.70**
Filtraggio Tag	0.808	0.825	0.817	0.784	0.734	0.695	0.681
Espansione Tag	0.564	0.603	0.628	0.67	0.638	0.642	0.677

Tabella 5.16: Valori medi di Accuracy ottenuti sull'intero dataset YouTube da 60 video rispetto ai possibili valori di *Threshold*.

generale, è senza dubbio più complicato che capire la bontà dei tag originali.

La categoria che mediamente ha il miglior andamento è *Travel & Events*, mentre la peggiore è *Music*. Per la fase di espansione, il valore di accuratezza massimo ottenuto è pari a 88.3% (in *Auto & Vehicles*), mentre il minimo si ha nella categoria *Pets & Animals* ed è pari a 30.4%.

Dopo aver evidenziato nel dettaglio i valori di accuratezza delle singole categorie, può essere utile fare una valutazione collettiva circa le prestazioni ottenute sull'intero dataset dei 60 video presi da YouTube. In altre parole, ci interessa effettuare un calcolo della media delle Accuracy rispetto ai sette possibili valori di *Threshold*. In questo modo si può evidenziare la *Threshold* ottimale, da utilizzare nei futuri esperimenti.

Quello che ci interessa è scegliere il valore di $thresh_{tag}$ (espressione (4.6) del paragrafo 4.2.2) che garantisce il massimo dell'accuratezza, in termini di filtraggio e di espansione dei tag. Come vedremo, gli andamenti dell'accuratezza media rispetto a filtraggio ed espansione sono in contro-tendenza; si tratta quindi di scegliere un valore che rappresenti un buon compromesso tra le due esigenze. La Tabella 5.16 riassume i valori medi di Accuracy ottenuti sull'intero dataset da 60 video in funzione dei sette valori di $thresh_{tag}$.

L'andamento delle curve relative al filtraggio e all'espansione dei tag sono visibili in Figura 5.19, dove si può notare come l'andamento della curva di filtraggio, rispetto all'aumentare della *Threshold*, sia in contrapposizione a quello della curva di espansione. Per arrivare a stabilire il valore ottimale di *Threshold*,

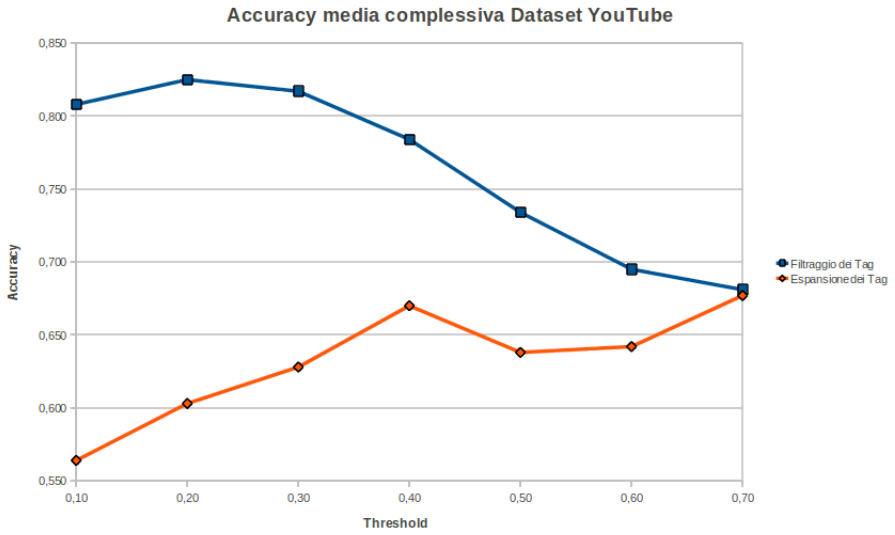

Figura 5.19: Accuracy media complessiva dataset YouTube.

Threshold	0.10	0.20	0.30	0.40	0.50	0.60	0.70
#Tag corretti	5.1	5.383	5.0	4.667	3.733	3.233	3.05

Tabella 5.17: Numero medio di tag corretti suggeriti dalla fase di espansione.

è necessario inserire un'ulteriore informazione. Ci interessa infatti conoscere il numero medio di tag corretti che vengono suggeriti dalla fase di espansione, in funzione delle sette soglie a cui facciamo riferimento. Quest'ultima informazione ci permette di capire fino a che valore di *Threshold* è possibile spingersi senza decrementare troppo il numero (assoluto) di termini aggiunti al set originale. In altre parole, vogliamo scegliere una soglia che garantisca in media di aggiungere un buon numero di annotazioni all'insieme di partenza delle stesse. La Tabella 5.17 e la Figura 5.20 mostrano nel dettaglio quanto detto finora. In base ai due grafici inseriti, possiamo concludere che il valore ottimale di *Threshold* è 0.40, grazie al quale si riesce ad ottenere un buon compromesso tra le due accuratezze (filtraggio ed espansione) ed il numero medio di tag corretti suggeriti. Con tale valore, infatti, la misura di *Accuracy* media garantita è del 78.4% (filtraggio) e

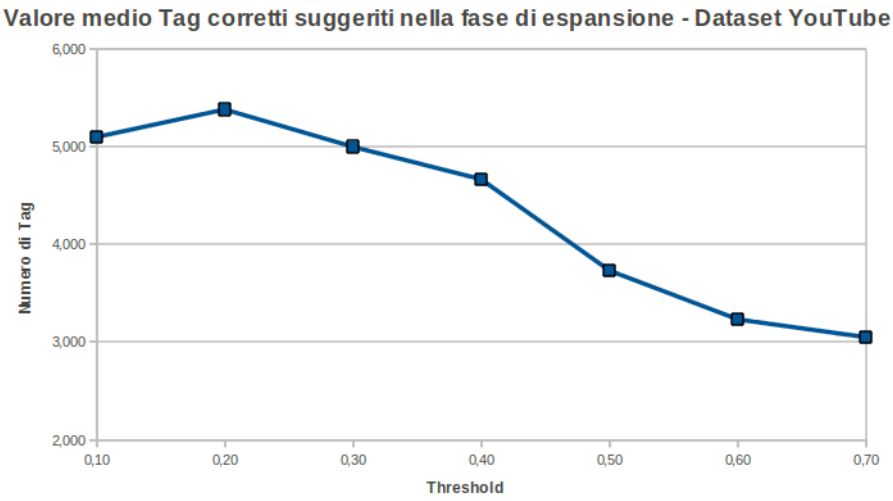

Figura 5.20: Numero medio di tag suggeriti in funzione della *Threshold*.

del 67% (espansione); mediamente inoltre si riescono ad aggiungere 4.667 tag al set originale delle annotazioni.

Dopo tutte queste considerazioni, possiamo concludere che il metodo implementato garantisce buoni risultati in termini di filtraggio ed espansione delle annotazioni. Il livello medio di accuratezza raggiunto è da ritenersi piuttosto soddisfacente, considerando anche che l'obiettivo è quello di creare un metodo capace di classificare le annotazioni di YouTube indipendentemente dal tipo di video, dal suo contenuto visuale o da qualsiasi altra forma di particolarità. Naturalmente, per ottenere delle buone prestazioni in termini di espansione, è necessario partire da un set orginale di tag sufficientemente valido.

Un'ultima considetazione deve essere fatta circa i tempi di esecuzione del software. Essi dipendono dal numero di tag che devono essere classificati ed espansi, ma considerando che un video ha al massimo (in generale) 25-30 tag, il tempo impiegato è compreso tra 60 e 120 secondi.

Possiamo passare adesso alla descrizione del prossimo esperimento, nel quale si misura la capacità di espansione del contenuto semantico del video. I dettagli

di tale prova sono specificati nel prossimo paragrafo.

5.2 Espansione semantica dei video tramite matching Tag to Concept (DBpedia)

In questa sezione descriveremo le prestazioni del sistema rispetto al task dell'espansione semantica dei video analizzati. Senza ritornare nel merito delle motivazioni che hanno spinto a portare avanti questo tipo di lavoro (l'arricchimento del contenuto descrittivo di un video), focalizziamoci adesso sulle prestazioni ottenute dal sistema. Lo scopo è valutare quanto il metodo implementato è capace di ritrovare le informazioni legate alle risorse DBpedia, a partire dal set degli *user-defined* tag validati nella fase di filtraggio. In questo tipo di esperimento si tiene conto anche della capacità dell'algoritmo di costruire i *termini di ricerca* (si faccia riferimento al paragrafo 4.2.4.1), dal momento che essi rappresentano la base di partenza per il matching *Tag to Concept*.

Prima di specificare i valori medi di accuratezza raggiunti rispetto al dataset descritto al paragrafo precedente, facciamo un esempio per chiarire meglio come opera nel concreto il software sviluppato in questo lavoro di tesi. Prendiamo come caso di studio il filmato "Video2" inserito in Tabella 5.2. La Figura 5.21 mostra i metadati associati a questo filmato: in particolare si può notare come i tag del video siano 19, alcuni dei quali altamente significativi. Il risultato della procedura di filtraggio e della creazione dei termini di ricerca è visibile in Figura 5.22. I tag mantenuti sono dieci: *Ferrari*, *Crash*, *enzo*, *f40*, *f430*, *testarossa*, *f50*, *maranello*, *scaglietti*, *modena*. A partire da queste etichette, vengono creati i termini di ricerca attraverso i quali è attuata la strategia di matching con le risorse DBpedia, secondo i dettami del paragrafo 4.2.4.1. Nella figura appena menzionata essi sono specificati di seguito al set di tag. A questo punto inizia

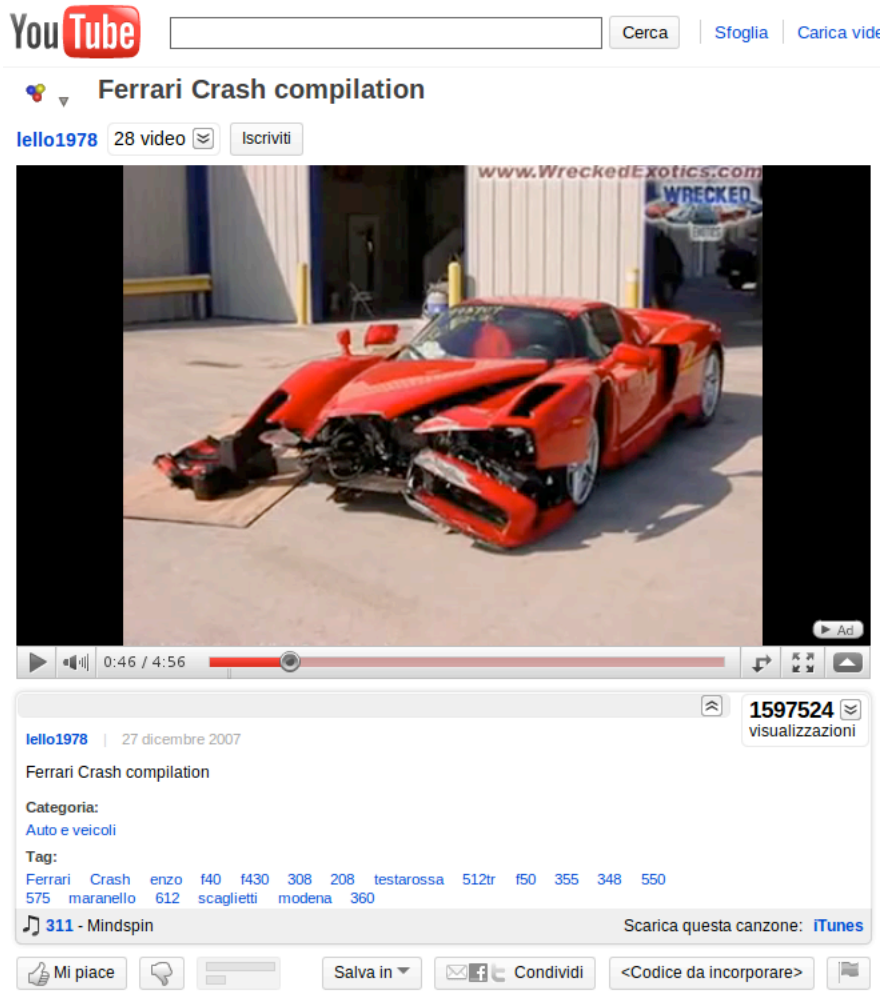

Figura 5.21: Filmato di esempio "FerrariCrash": categoria *Auto & Vehicles*.

```
TAGS SU CUI FARE L'ESPANSIONE (SENZA SUGGESTED-TAGS): Ferrari, Crash, enzo, f40,
 f430, testarossa, f50, maranello, scaglietti, modena,

TAG SUGGERITI IN BASE AI RELATED VIDEOS:
Tag: Ferrari  Frequenza: 0.72

TAGS SU CUI FARE L'ESPANSIONE (CON SUGGESTED-TAGS): Ferrari, Crash, enzo, f40, f
430, testarossa, f50, maranello, scaglietti, modena,

TERMINI SU CUI FARE L'ESPANSIONE (CON ACCORPAMENTO TAGS):
Search Term DBpedia 0: enzo
Search Term DBpedia 1: f40
Search Term DBpedia 2: f430
Search Term DBpedia 3: testarossa f50
Search Term DBpedia 4: maranello
Search Term DBpedia 5: scaglietti
Search Term DBpedia 6: modena
Search Term DBpedia 7: Ferrari Crash
```

Figura 5.22: Screenshot filmato "FerrariCrash": creazione dei *termini di ricerca*.

```
VISUALIZZAZIONE DEI MATCH TRA I TAG VALIDATI E LE RISORSE DBPEDIA
Indice Search:0  Search Term: enzo  Indice Ontologia:3  Ontologia: Automobile  N
ome risorsa: Ferrari Enzo Ferrari
Indice Search:1  Search Term: f40  Indice Ontologia:3  Ontologia: Automobile  No
me risorsa: Ferrari F40
Indice Search:2  Search Term: f430  Indice Ontologia:3  Ontologia: Automobile  N
ome risorsa: Ferrari F430
Indice Search:4  Search Term: maranello  Indice Ontologia:3  Ontologia: Automobi
le  Nome risorsa: Ferrari 575M Maranello
Indice Search:5  Search Term: scaglietti  Indice Ontologia:3  Ontologia: Automob
ile  Nome risorsa: Ferrari 612 Scaglietti
```

Figura 5.23: Screenshot filmato "FerrariCrash": matching *Tag to Concept*.

la fase di matching tra i search term e le risorse DBpedia; il risultato per questo filmato di esempio è mostrato in Figura 5.23.

Il risultato del video di esempio appena considerato è stato ottenuto con Threshold = 0.40 (valore ottimale stabilito al paragrafo precedente). Siamo pronti quindi per entrare nel dettaglio delle rilevanze sperimentali delle varie categorie di YouTube. I test sono stati svolti sui vari filmati con *Threshold* fissa impostata al valore ottimale (0.40). Per ogni filmato è stata calcolata l'*accuratezza di matching*. Questa misura consiste nel rapporto tra il numero di operazioni corrette svolte sui termini di ricerca ed il numero totale di quest'ultimi.

Vediamo di chiarire meglio quanto appena detto. Se un termine di ricerca è

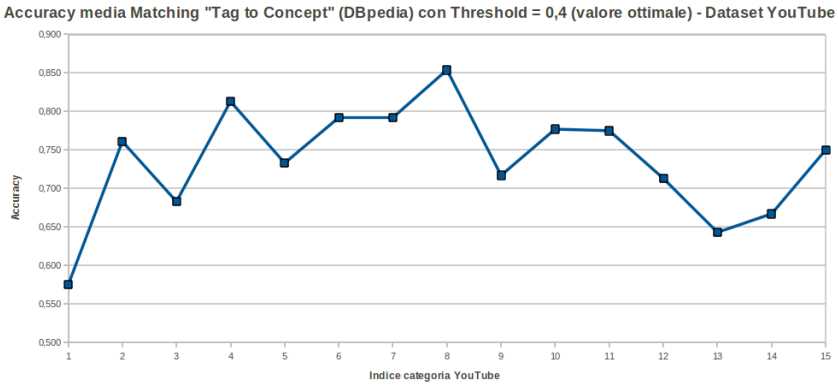

Figura 5.24: Accuracy media *Tag to Concept* rispetto alle 15 categorie del dataset YouTube (60 filmati).

stato creato male (termine errato, termine privo di significato, termine composto da troppi tag, ...), l'operazione ad esso annessa è sicuramente scorretta; un termine di ricerca valido che provoca un match con la risorsa DBpedia corretta è associato ad un'operazione corretta. Un termine di ricerca valido al quale **non è associabile** una risorsa DBpedia e che non provoca un match consiste in un'operazione corretta; in caso contrario (si verifica un match) l'operazione è da considerarsi errata. In definitiva, la misura di Accuracy appena definita tiene conto di due aspetti:

- la capacità dell'algoritmo di creare termini di ricerca adeguati;
- la capacità dell'algoritmo di stabilire delle corrispondenze corrette *Tag to Concept*.

Dopo aver calcolato l'accuratezza per ogni filmato, se ne calcola il valore medio sui quattro filmati di ogni categoria (come negli esperimenti descritti al paragrafo precedente). La Tabella 5.18 riassume i valori numerici di *Accuracy* ottenuti in media su ogni YouTube category. L'andamento della curva dell'accuratezza è mostrato in Figura 5.24.

Categoria YouTube	**Accuracy Tag to Concept**
Travel & Events	0.575
Auto & Vehicles	0.761
Science & Technology	0.683
Comedy	0.813
Education	0.733
Pets & Animals	0.792
People & Blogs	0.792
Howto & Style	0.854
Sports	0.717
Nonprofits & Activism	0.777
News & Politics	0.775
Film & Animation	0.713
Gaming	0.643
Music	0.667
Entertainment	0.75

Tabella 5.18: Accuracy media *Tag to Concept* (DBpedia) rispetto alle 15 categorie analizzate (4 filmati per categoria).

In definitiva, le prestazioni che si riescono ad ottenere per questo esperimento sono piuttosto soddisfacenti (abbiamo un'accuratezza minima di 57.5% ed un'accuratezza massima pari a 85.4%). Sebbene con il matching *Tag to Concept* non si riesca ad incrementare il numero delle annotazioni per un video (i termini che si potrebbero estrarre sarebbero troppo generali e forse poco utili rispetto agli obiettivi di questo lavoro), questa parte risulta utile per quanto concerne l'arricchimento del contenuto descrittico associato ad un video. In pratica, il *Tag to Concept* può essere assimilato ad una sorta di "incremento" della quantità dei metadati associati ad un filmato (col risultato di avere più informazioni semantiche ad esso legate).

In ultima analisi, specifichiamo le prestazioni di questa fase in termini di tempo di elaborazione. La ricerca delle risorse in DBpedia risulta molto onerosa in tal senso; i tempi di esecuzione dipendono fortemente dal numero dei termini di ricerca. Volendo fare una stima del caso peggiore riscontrato nei test, possono occorrere anche 20 minuti per completare la ricerca delle risorse. Nel caso migliore, invece, il tempo richiesto è di circa 4-5 minuti.

5.3 Il test sul dataset TRECVID 2005

In questa sezione descriviamo la terza ed ultima tipologia di prove sperimentali fatte sul software sviluppato in questo lavoro di tesi. In questo caso non vengono eseguiti dei test sui video residenti in YouTube, bensì si valutano le performance su un dataset alternativo (**TRECVID 2005**). Quest'ultimo contiene una grande mole di annotazioni relative a svariati *shot* estratti da filmati che sostanzialmente consistono in notiziari statunitensi. Per ogni shot, è annotata la presenza/non presenza di ognuno dei 433 concetti di LSCOM, a cui il dataset TRECVID fa riferimento. *LSCOM (Large Scale Concept Ontology for Multimedia)* consiste in una serie di progetti portati avanti da Aprile 2004 a Settembre

2006 allo scopo di definire un vocabolario formale standard per l'annotazione ed il recupero dei video.

In questo terzo tipo di test, lo scopo è quello di valutare secondo una certa logica la bontà della fase di espansione dei tag. Chiariamo meglio questo concetto. In pratica, la modalità di svolgimento del test è la seguente: si seleziona, per ogni shot, un sottoinsieme dei concetti LSCOM ivi presenti e si valuta se la procedura di espansione dei tag riesce a "coprire" i restanti concetti (quelli che non sono stati selezionati).

Le prestazioni del sistema sono valutate sempre con una sorta di *Accuracy*: si effettua il rapporto tra i concetti che vengono coperti dalla procedura di espansione ed il loro numero totale. Dopo aver presentato i dati, analizzeremo le prestazioni ottenute in questo test. Possiamo comunque già affermare che esse saranno di gran lunga inferiori a quelle ottenute sul dataset YouTube, per una serie di motivazioni che chiariremo nel seguito.

In definitva, quindi, con questo esperimento non si effettua alcuna fase di filtraggio delle annotazioni ma si valuta soltanto la parte relativa all'espansione. Date le scarse prestazioni che si riescono ad ottenere, in questo elaborato verrà documentato un caso di test svolto con pochi esempi (è emerso che un numero più alto di prove non cambia di molto la statistica sulla valutazione dei risultati).

Descriviamo comunque nel dettaglio le singole operazioni svolte in questo tipo di prova. In primo luogo sono stati selezionati in modo casuale 15 filmati tra i 138 totali del dataset TRECVID 2005; per ogni video si analizza uno dei suoi shot annotando i concetti che in esso sono presenti. Per avere una visione più precisa di tale operazione, si consideri la Figura 5.25, dove è mostrato il file di testo con le annotazioni. Preso uno ad uno ogni shot, si crea un ulteriore file di testo dove su ogni riga viene inserito il sottoinsieme dei concetti selezionati per quel caso di test (tipicamente si seleziona un numero di annotazioni pari al

Figura 5.25: File "TRECVID05.txt": concetti LSCOM presenti negli shot selezionati casualmente.

30% del totale). Questa fase è ben rappresentata dalla Figura 5.26.

Vediamo quindi i singoli valori di accuratezza che si ottengono sui 15 casi di test totali. La Tabella 5.19 contiene a questo proposito tutte le informazioni di interesse. L'andamento dell'accuratezza di copertura è evidenziato in maniera ancora più dettagliata tramite la Figura 5.27.

Come è possibile vedere facilmente, questo terzo tipo di test non ha prodotto risultati soddisfacenti. Le motivazioni di queste basse prestazioni sono legate essenzialmente ad un aspetto fondamentale: il sistema pensato nel presente lavoro di tesi è stato concepito per lavorare con gli user-defined tag associati ai video di YouTube. L'ontologia LSCOM, quindi, si rivela assai distante dalle linee guida del metodo descritto in questo elaborato. Per ottenere buoni risultati su TRECVID 2005 sarebbe necessaria una rivalutazione profonda dell'intero algoritmo, che però non rientra negli obiettivi di questo lavoro.

Come nei due paragrafi precedenti, facciamo un'ultima considerazione rispetto al tempo di elaborazione richiesto per questo tipo di test. Non dovendo impiegare risorse per la fase di filtraggio, i casi di test su TRECVID 2005 vengono completati rapidamente. Tipicamente, per fare un test su uno shot occorrono circa 20 secondi (piccole variazioni si hanno in base alla dimensione del sottoinsieme dei concetti preso in considerazione). Rispetto al caso in esame in questo

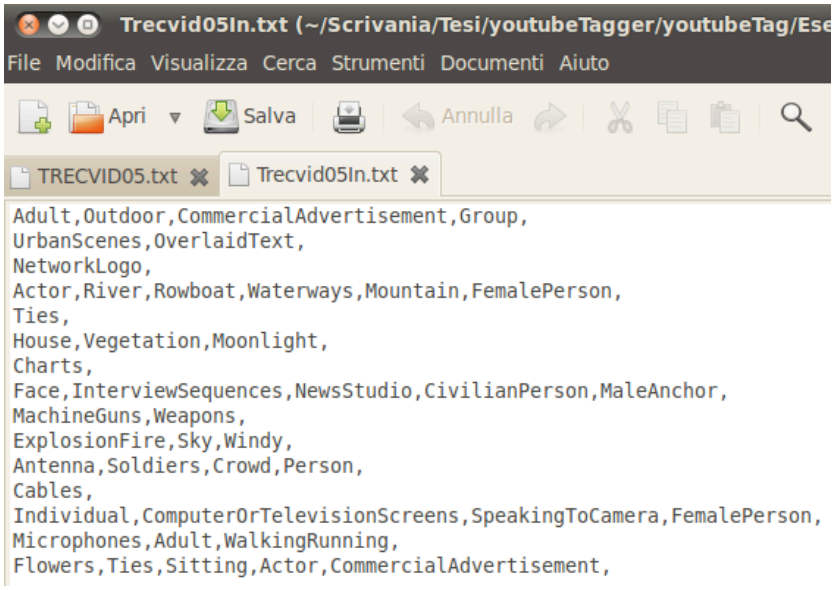

Figura 5.26: File "Trecvid05In.txt": sottoinsieme dei concetti LSCOM di ogni shot.

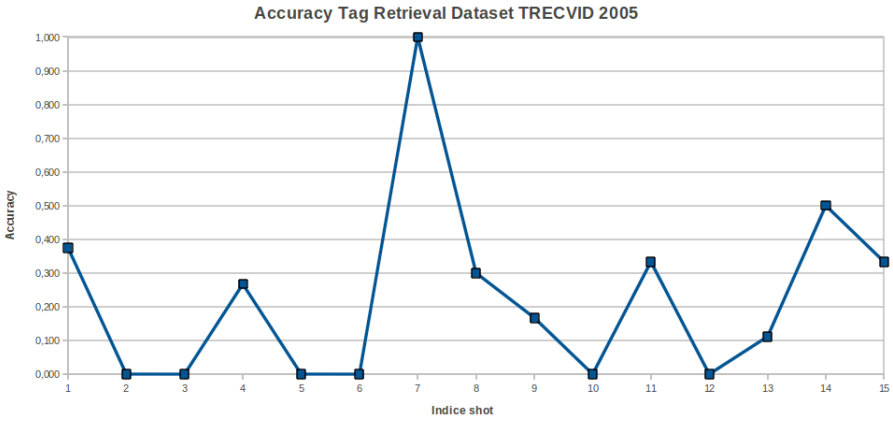

Figura 5.27: Accuracy dataset TRECVID 2005.

Shot	**#Concetti Sel.**	**#Concetti Tot.**	**#Concetti Cop.**	**Acc.**
1	4	12	3	0.375
2	2	6	0	0.0
3	1	3	0	0.0
4	6	21	4	0.267
5	1	2	0	0.0
6	3	11	0	0.0
7	1	2	1	1.0
8	5	15	3	0.3
9	2	8	1	0.167
10	3	9	0	0.0
11	4	10	2	0.333
12	1	4	0	0.0
13	4	13	1	0.111
14	3	9	3	0.5
15	5	14	3	0.333

Tabella 5.19: Accuracy test TRECVID 2005. Shot: indice dello shot analizzato - #Concetti Sel.: concetti selezionati - #ConcettiTot.: concetti totali - #ConcettiCop.: concetti coperti - Acc.: Accuracy

paragrafo (15 casi di test), il tempo di elaborazione richiesto è risultato pari a 300 secondi.

Capitolo 6

Conclusioni e sviluppi futuri

Nel presente lavoro di tesi è stata affrontata la problematica relativa all'affidabilita delle annotazioni associate ai video residenti in YouTube. Nel corso della trattazione ha assunto una notevole importanza la strategia messa in atto per la valutazione dell'affidabilità dei tag, oltre alle metodiche utilizzate per la loro espansione. Il sistema, nel suo complesso, mira a migliorare la qualità delle informazioni testuali associate ad un filmato. Per raggiungere tale obiettivo, siamo partiti dalle annotazioni già inserite dagli autori dei video ed abbiamo definito una procedura di espansione, ristretta ai tag ritenuti validi.

Nella maggior parte dei casi, i soli tag non sono capaci di coprire l'intero contenuto semantico di un filmato. Una parte importante di questo progetto è stata perciò dedicata alla creazione di un metodo che favorisse l'integrazione degli *user-defined tag* con il *Semantic Web*. Da tale filone è nata l'idea di integrare il sistema all'interno di ontologie predefinite, ed in particolare mettere in atto un approccio capace di legare le annotazioni alle entità gestite dal progetto DBpedia.

I risultati sperimentali dimostrano che è necessaria una valutazione differita

dei due tronconi nei quali si divide il lavoro. La fase di filtraggio e quella di espansione garantiscono buoni risultati, sia in termini di accuratezza sia rispetto ai tempi di elaborazione. Il software realizzato riesce a classificare ed aumentare il numero di tag in un tempo compreso tra 60 e 120 secondi, a seconda del filmato. Tali tempistiche si riferiscono a test effettuati tramite un elaboratore portatile dotato di CPU con clock a 1,60 GHz e memoria RAM con capacità pari a 8 GB.

I tempi di elaborazione richiesti dal software per l'integrazione dei tag con il Semantic Web, invece, incidono in misura notevole sulla valutazione complessiva del sistema. L'estrazione del contenuto semantico richiede infatti, per ogni video, un tempo compreso tra 4 e 20 minuti, a seconda della categoria YouTube alla quale esso appartiene. Tale risultato, perciò, ci porta ad affermare che il sistema avrebbe bisogno, attualmente, di una nuova fase di studio volta a migliorare il suddetto aspetto.

In definitiva, il software presentato in questa tesi risulta affidabile per quanto concerne la classificazione e l'espansione degli *user-defined tag*. Il sistema è stato progettato per operare con qualsiasi filmato di YouTube; i risultati ottenuti dimostrano la bontà del metodo anche in relazione al fatto che permette di elaborare video scelti in modo generico senza particolari limitazioni.

Terminiamo quest'ultimo capitolo facendo un breve accenno a quelli che sono i possibili sviluppi futuri legati al presente lavoro. In primo luogo, il sistema potrebbe essere espanso per migliorare l'integrazione con alcune delle ontologie definite nel corso degli ultimi anni. Si pensi ad esempio alla possibilità di integrare questo sistema con l'ontologia LSCOM per sfruttarne i concetti nella fase di annotazione dei filmati. Una valida alternativa è senza dubbio la progettazione di un'espansione dei tag basata su DBpedia. In tal caso, infatti, si avrebbe la possibilità di affrontare aspetti legati ad un interessante tema di

ricerca, tuttora al centro del dibattito nella comunità scientifica. Un ulteriore sviluppo consiste nell'utilizzo delle informazioni visuali associate ad un filmato. In sintesi, potrebbe essere progettata una metodologia capace di effettuare la auto-annotazione di un video indagando la similarità visuale rispetto ad altri filmati, magari residenti in archivi di grande dimensione. Il sistema progettato in questo lavoro di tesi, in definitiva, apre la strada a molteplici scenari di sviluppo.

Bibliografia

[Bizer09] Christian Bizer (1), Jens Lehmann (2), Georgi Kobilarov(1), Soren Auer (2), Christian Becker (1), Richard Cyganiak (3), Sebastian Hellmann (2), "DBpedia - A Crystallization Point for the Web of Data", *(1) Freie Universit¨t Berlin, Web-based Systems Group, Garystr. 21, D-14195 Berlin, Germany. (2) Universit¨t a Leipzig, Department of Computer Science, Johannisgasse 26, D-04103 Leipzig, Germany. (3) Digital Enterprise Research Institute, National University of Ireland, Lower Dangan, Galway, Ireland.*

[Chang05] Hsi-Cheng Chang and Chiun-Chieh Hsu, "Using topic keyword clusters for automatic document clustering", *IEICE TRANSACTIONS on Information and Systems*, vol. E88-D, no. 8, pp. 1852-1860, 2005.

[Choudhury] Smitashree Choudhury (1), John G. Breslin (1)(2), Alexandre Passant (1), "Enrichment and Ranking of the YouTube Tag Space and Integration with the Linked Data Cloud", *(1)DERI, National University of Ireland, Galway, Ireland. (2)School of Engineering and Informatics, National University of Ireland, Galway, Ireland.*

[MICC10] L. Ballan, M. Bertini, A. Del Bimbo, M. Meoni, and G. Serra (Media Integration and Communication Center, Università degli Studi di Firenze). "Tag suggestion and localization in user-generated vi-

deos based on social knowledge". *In WSM'10, October 25-29, 2010, Firenze, Italy.*

[Milne08] David Milne, "An Open-Source Toolkit for Mining Wikipedia", *Department of Computer Science, University of Waikato Private Bag 3105, Hamilton, New Zealand.*

[Milne-Witten08a] David Milne, Ian H. Witten, "An effective, low-cost measure of semantic relatedness obtained from Wikipedia links", *Department of Computer Science, University of Waikato Private Bag 3105, Hamilton, New Zealand.*

[Milne-Witten08b] David Milne, Ian H. Witten, "Learning to Link with Wikipedia", *Department of Computer Science, University of Waikato Private Bag 3105, Hamilton, New Zealand.*

[Smeraldo10] Salvatore Smeraldo, "Analisi della similarità visuale per la ricerca di video annotati in archivi di grande dimensione", *Dipartimento di Sistemi e Informatica, Università degli Studi di Firenze, Facoltà di Ingegneria. Tesi di Laurea Specialistica in Ingegneria Informatica, A.A. 2009/2010.*

[Snoek09] Xirong Li, Cees G. M. Snoek, *Member*, IEEE, Marcel Worring, *Member*, IEEE, "Learning Social Tag Relevance by Neighbor Voting", *IEEE Transaction On Multimedia*, Vol. V, No. N, Month 2009.

[Torralba08] A. Torralba, R. Fergus, and W. T. Freeman, "80 milion tiny images: A large data set for nonparametric object and scene recognition", *IEEE Trans. PAMI*, vol 30, no. 11, pp. 1958-1970, 2008.

[VanZowl08] B.Sigurbjonsson and R. van Zwol, "Flickr tag recommendation based on collective knowledge", in *Proc. WWW*, 2008, pp. 327-336.

[Wang08] C. Wang, F. Jing, L. Zhang, and H.-J. Zhang, "Scalable search-based image annotation", *Multimedia Systems*, vol. 14, no. 4, pp. 205-220, 2008.