UNIVERSITÀ DEGLI STUDI DI MILANO
FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
CORSO DI LAUREA SPECIALISTICA IN INFORMATICA

METODI PER LA PERSONALIZZAZIONE DEL WEB CRAWLING

Relatore: Prof. Sebastiano VIGNA
Correlatore: Prof. Paolo BOLDI
Correlatore: Dott. Massimo SANTINI

Tesi di Laurea di:
Alessio ORLANDI
Matricola 702190

Anno Accademico 2006/07

Ringraziamenti

Questa tesi è frutto di un lungo lavoro che non avrei mai potuto affrontare senza il sostegno di molti.
Il primo ringraziamento va ai Proff. Sebastiano Vigna e Paolo Boldi e al dott. Massimo Santini per il tempo e la pazienza dedicatimi e per i preziosi consigli e l'ispirazione. Un altro è sicuramente alla mia famiglia, il cui sostegno incondizionato non è mai venuto meno durante questi anni.

Voglio inoltre ringraziare, in ordine puramente casuale, Daniele Berto, Sara Cannizzaro, Camillo Gastaldi, Elena Stratta, Francesco Pelaccia, Carlo De Santis, Paola Lorusso e Raffaella Stoppani, per avermi sopportato ed ascoltato anche nei momenti più bui ed avermi aiutato a trovare nuovamente la forza di alzare la testa, oltre che per le splendide serate passate in compagnia.

Una menzione va anche a Enrico Pertoso, Diego Valota, Antonio Zippo, Luca Natali, Lorenzo Valerio e Jacopo Biscella, come a Fabio Pietrosanti, Yvette Agostini, Claudio Agosti e Guido Bolognesi, che sono stati parte integrante, quando non preziosa, della mia vita in questi anni da studente universitario.

A chi non teme le proprie idee

Indice

Capitolo 1

Introduzione

1.1 Prefazione

La crescente mole di informazione sparsa in rete e la conseguente necessità di creare dei punti di accesso centrali alla stessa hanno spinto la comunità scientifica e le aziende ad interessarsi ai motori di ricerca, oggi diventati ormai strumento di uso comune per ogni utente della rete. Immediata conseguenza è stato l'interesse per il problema di come collezionare efficientemente la base di dati (nel nostro caso, le pagine web disponibili su Internet) e come progettare dei software in grado di farlo. Il problema del *crawling* può essere dunque descritto informalmente come il tentativo di utilizzare al meglio le risorse computazionali a disposizione per esplorare Internet a partire da un insieme di indirizzi forniti in partenza. Il primo scenario formatosi è quello di un crawler che archivia incondizionatamente il maggior numero di pagine web su cui si aggancia un motore di ricerca, che organizza i dati raccolti dal crawler proponendo poi all'utente finale un'interfaccia di ricerca generale per recuperare dati sull'intera base.

Successivamente, intorno agli anni 90, alcuni scienziati hanno cominciato ad interessarsi alla personalizzazione della ricerca, ovvero al tentativo di allineare i risultati della ricerca ad un qualche profilo dell'utente. I percorsi disponibili si raggruppano in due grandi filoni: il tentativo di filtrare i risultati della ricerca o alterare l'algoritmo di attribuzione della rilevanza alle pagine web in base al profilo dell'utente oppure applicare delle forme di riduzione e selezione durante la costruzione stessa della base di dati dipendentemente dai profili. Il primo percoros è definito *preferential search* mentre il secondo viene denominato *focused crawling* (o crawling mirato).

Scopo della tesi è quello dipresentare una nuova tecnica basata su metodi di classificazione semi-supervisionata, di cui è presente un'implementazione per il focused crawling ed è organizzata come segue. Il presente capitolo continua con una descrizione più approfondita e formale del problema del focused crawling fornendo una sua inquadratura operativa, dando una panoramica

delle tecniche più significative presenti in letteratura, evidenzionandone i pregi e i difetti salienti. Il Capitolo 2 descrive più in dettaglio due dei modelli presentati in questo capitolo, che saranno poi di interesse ai fini della proposta di un nuovo algoritmo, approfondendo inoltre la teoria degli elementi che li compongono. Il Capitolo 3 introduce un problema laterale che si è dovuto affrontare: la creazione di una base di dati per permettere il confronto e la riproducibilità di esperimenti di focused crawling. Descrive quindi la soluzione adottata e affronta il dilemma teorico di come confrontare l'efficienza di un focused crawler singolarmente in un ambiente sperimentale predeterminato. Il Capitolo 4 affronta la scelta di un classificatore semi-supervisionato per documenti testuali, riportando alcuni esperimenti ulteriori a quelli già proposti negli articoli esistenti, al fine di valutare le performance di alcuni classificatori su dati estremamente realistici quali le pagine web. Nel Capitolo 5 si descrive invece in dettaglio l'idea innovativa del focused crawler, coinvolgendo anche i risultati del Capitolo 4 e dando spazio ad alcuni ragionamenti che riprendono il Capitolo 2. Infine il Capitolo 6 presenta i risultati ottenuti dagli esperimenti condotti, fornendo alcune conclusioni e aprendo quesiti per il futuro.

1.2 Crawling mirato

Lo scopo di un generico crawler, mirato o meno, è quello di eseguire una visita sulla rete a partire da un insieme di pagine fornite esternamente (detto *seed* o seme). Supponiamo per comodità e senza perdere generalità di voler eseguire il crawling di soli documenti ipertestuali (HTML, XML, ...) che quindi contengono riferimenti ad altri documenti ipertestuali. È possibile modellare la rete come un grafo orientato non pesato $G = (V, E)$ in cui i vertici sono le pagine web e gli archi rappresentano i link tra le pagine. Un crawler conosce inzialmente il seed $S \subset V$ ed esegue una visita sul grafo G. La definizione di visita utilizzata in questo contesto è la seguente:

Definizione 1 *Una visita (parziale) su un grafo G a partire dal seme S è un insieme $W = \{w_0, .., w_n\}$ tale che $W \subseteq V$ e $W = \bigcup_{i=0}^{n} W_i$ dove ogni insieme $W_i = \{w_0, .., w_i\}$ è costruito come segue:*

- $W_0 = \{w_0\}$ *e* $w_0 \in S$.
- $W_i = W_{i-1} \cup \{w_i\}$ *dove* $w_i \in S \wedge w_i \notin W_{i-1}$ *oppure* $w_i \notin S \wedge w_i \notin W_{i-1} \wedge \exists x \in W_{i-1}$ *tale che* $x \rightarrow w_i$.

Quando non di interesse o ricavabile dal contesto, ometteremo la dicitura del seme.

Questa definizione descrive immediatamente la dimensione temporale del crawl: se lo stato della visita è fermo al momento t (a cui corrisponde una

sottovisita W_t) la scelta del nodo successivo fa evolvere la visita verso W_{t+1}. In ogni momento quindi il grafo è sempre suddiviso in due insiemi W_t e $V \backslash W_t$ che rappresentano i nodi visitati e non visitati, da cui:

Definizione 2 *Si definisce* **frontiera** *di una visita W su un grafo G a partire dal seme S il sottoinsieme massimale $F \subseteq W$ formato dai nodi per cui esiste almeno un arco $e = (i, j)$ dove $i \in F$ e $j \in V \backslash W$.*

Un crawler normale si limita a definire una visita sul grafo G, mentre per definire un crawler mirato abbiamo bisogno di inglobare il concetto di interesse dell'utente; per mantenere la massima generalità per ora definiremo $R \subseteq V$ come l'insieme delle pagine di interesse dell'utente, che potrà essere uno qualsiasi. Il focused crawler dovrà dunque cercare di effettuare una visita su G per recuperare il più efficientemente e rapidamente possibile gli elementi di R. La scelta dell'effettiva definizione operativa di *efficientemente* e *rapidamente* verrà approfondita nel capitolo dedicato alla collezione per esperimenti. Infatti, come si vedrà nel prossimo sezione, differenti autori hanno utilizzato altrettanto differenti metodologie di misura delle prestazioni nell'affrontare il problema dei focused crawler.

La progettazione di un focused crawler si divide quindi in due problematiche base: incorporare il profilo dell'utente all'interno del crawler e creare una strategia di visita che permetta di far evolvere una visita in modo da mantenere alta la percentuale di pagine web attinenti al profilo dell'utente. Le scelte inerenti queste problematiche sono dettate principalmente dall'utilizzo finale del focused crawler. Pertanto, ci si propone ora di fare una panoramica sulle soluzioni già presenti in letteratura.

1.3 Altri approcci

Il primo rudimentale approccio al focused crawling viene descritto in [12] con un crawler di tipo Best First. Un crawler best-first ordina i nodi della frontiera (e di conseguenza gli archi uscenti dalla stessa) secondo un certo criterio di ordinamento R ed ad ogni passo seleziona il nuovo nodo da visitare prendendo il massimo secondo R tra i nodi presenti in frontiera e seguendo uno dei link uscenti dal nodo selezionato (a caso). L'articolo illustra all'inizio due principali metriche di ordinamento: il conteggio degli inlink conosciuti di una pagina in coda (backlink) oppure il calcolo di PageRank [23] con $\alpha = 0.9$ sulla porzione di grafo conosciuta finora. Una terza metrica, detta di similarità, è d'interesse. Si supponga che il crawler debba essere addestrato per la ricerca di pagine di un singolo argomento, ad esempio "computer", e che esista un classificatore molto semplice basato sul testo delle pagine che descriva se una pagina èpertinente o meno (ad esempio la presenza o meno della parola "computer" nel titolo della pagina). Allora ogni nuova pagina che deve essere inserita nella frontiera viene categorizzata come *hot*

se è on topic oppure è a distanza massima 3 da una pagina on topic. Al momento di scegliere il prossimo nodo da visitare si estrae un link a caso tra le pagine in frontiera *hot* e solo nel caso questo insieme sia vuoto, dai rimanenti. Una naturale evoluzione presentata in seguito dagli stessi autori utilizza una misura di similarità come ad esempio il coseno con soglia per determinare l'appartenenza o meno al topic, scegliendo poi di percorrere i link da quello con misura del coseno più basso tra quelle hot. Il crawler best first è il primo esempio di focused crawler e, paradossalmente, esperimenti svolti in [20] rivelano che il crawler best first ha delle performance migliori di alcuni focused crawler logicamente più complessi.

Un'altra definizione di visita Best-First, che sarà quella usata nel resto della tesi è più semplice e consiste in una coda con priorità che mantiene i nodi in frontiera. Dato un qualsiasi punteggio (ad esempio di regressione di un classificatore) tra 0 e 1 assegnato alle pagine in frontiera man mano che vengono inserite, il crawler sceglie sempre di esplorare i figli della pagina con massima priorità ancora in frontiera.

1.3.1 Primi esperimenti di Chakrabarti

Nel 1999 Chakrabarti ed altri [9] pubblicano il primo vero lavoro su focused crawling, definendo tale oggetto un crawler dotato di due elementi: un *classificatore* e un *distillatore*, il primo dedito a controllare l'attinenza delle pagine recuperate e il secondo a scegliere quali link possa essere utile seguire ai fini di un crawling efficiente. La pubblicazione si basa sulle seguenti due assunzioni: il grado di rilevanza di una pagina è indicativo e di cui, per ognuno dei topic, siano presenti pagine di esempio. Assumendo il modello di generazione dei documenti sia bernoulliano (ovvero ogni termine compaia indipendentemente dagli altri) studiando la probabilità di presenza di ogni termine all'interno degli esempi di un topic è possibile stimare la probabiltà che ogni termine sia presente all'interno del topic. Dato un generico nuovo documento la frequenza del termine all'interno dello stesso fornisce l'informazione necessaria per ricostruire la probabilità che il documento sia all'interno di uno dei vari topic. Scegliendo il topic più è probabile possibile classificare il documento come interessante o meno. In questo modo, scelte le pagine di esempio dei vari topic T si può definire una misura di similarià tra una generica nuova pagina p ed ognuno degli esempi $e(T)$ (ad esempio, il coseno) e utilizzare questa misura come rilevanza. Parallelamente, il focused crawler utilizza una versione modificata dell'algoritmo HITS [17] per definire il grado di autorità delle singole pagine in base a quelle vicine e poter dare una priorità agli elementi nella frontiera.

1.3.2 Reinforcement Learning

Qualche mese dopo, McCallum e Rennie [25] propongono una soluzione basata sul Reinforcement Learning, in quanto il contesto del focused crawling sembra essere molto adatto all'utilizzo di queste tecniche. Si supponga che il crawler sia un oggetto "intelligente", ovvero che abbia un suo stato interno e che abbia, in ogni momento, la possibilità di eseguire alcune azioni che portino ad una ricompensa da parte dell'ambiente e, possibilmente, ad una variazione dello stato interiore. Più precisamente, ad ogni pagina rilevante si associa una ricompensa di valore +1, mentre alle altre, -1. Alla visita costruita dal crawler si può dunque associare la ricompensa a lungo termine

$$R = \sum_{t=1}^{V} \gamma^t r(t)$$

dove $r(t)$ è la ricompensa associata alla pagina recuperata al tempo t e γ è una costante tra 0 e 1. Il problema del focused crawling diventa quindi quello di cercare di apprendere una policy ottima, ovvero una funzione che dato lo stato scelga tra le possibili azioni quella che, in media, produce una ricompensa a lungo termine massima. In questo caso, l'insieme degli stati è l'insieme di pagine rilevante mentre le azioni possibili a partire da ogni stato sono tutti gli archi contenuti dalla frontiera in quel momento. Tutte le azioni che portano a pagine rilevanti generano una ricompensa negativa e lasciano il crawler nello stesso stato in cui si trova. Esistono tecniche standard quali, come proposto nell'articolo, il Q-learning, che tramite tecniche di programmazione dinamica ottimizza la policy usando a supporto una funzione $Q{:}S \times A \rightarrow \mathbb{R}$ che fornisce una stima di R per ogni coppia stato, azione. Tuttavia essa si rivela in pratica non applicabile per la eccessiva grandezza dello spazio degli stati. Gli autori dunque propongono di ignorare lo stato attuale del crawler per ridurre la dimensionalità del problema ed accellerare di conseguenza l'apprendimento, raggruppando inoltre differenti azioni sostituendo ad ogni link un insieme di parole che si trovano nelle vicinanze all'interno del testo della pagina di partenza del link. In questo modo Q diventa una funzione da un insieme di parole ai reali che, una volta discretizzata in pochi gruppi di valori, può essere appresa e riutilizzata con dei classificatori bayesiani naive (si veda a questo proposito il successivo Capitolo). L'articolo propone alcuni risultati su un crawl di qualche migliaio di pagine, sperimentando con varie tecniche di discretizzazione dei valori della *Q-function*.

1.3.3 Grafi contestuali

L'approccio dei grafi contestuali [14] presentato nel 2000 da Diligenti ed altri propone una tecnica semplice ma sicuramente efficace, ponendosi la seguente domanda (chiaramente ispirata anche dagli esperimenti del reinforcement

learning): ogni pagina ha associata, in realtà, una distanza dalla più vicina pagina rilevante, pertanto, esiste un collegamento tra il contenuto della pagina e la distanza? Supponendo risposta affermativa, gli autori introducono il concetto di grafi contestuali, che si basa sulla capcità di avere accesso, ad esempio tramite le limitate capacità di generazione degli elenchi delle pagine "riferenti a" dei motori di ricerca, ad alcune porzioni del trasposto del grafo del web. Si definisce dunque grafo di contesto di profondità k di una pagina web rilevante l'albero di visita per ampiezza sul trasposto radicato sulla pagina rilevante e di massima profondità k. Si definisce inoltre grafo di contesto generale[1] di profondità k la sovrapposizione di un insieme di grafi contestuali di profondità k dove i vertici rappresentati una stessa pagina web vengono fatti collassare assieme. Il grafo di contesto generale viene generato per un insieme di pagine rilevanti fornite inizialmente al sistema dopodichè le pagine vengono suddivise in sottoinsiemi aventi la stessa distanza da una pagina rilevante, detti *strati*. L'algoritmo di visita si avvale di un classificatore bayesiano naive leggermente modificato avente $k+1$ classi addestrato sui k strati del grafo contestuale generale, più una classe extra che rappresenta la distanza $\geq k+1$, a cui vengono attribuiti i documenti la cui percentuale di classificazione per una qualsiasi altra classe non supera una soglia data. In questo modo la visita evolve con $k+1$ code di visita parallele: ogni volta che una nuova pagina viene recuperata, viene classificata e tutti i link uscenti vengono inseriti (o spostati, se già visitati e la cosa si rivela vantaggiosa) nella coda di distanza subito inferiore; subito dopo il crawler estrae la nuova pagina dalla coda non vuota con indicatore di distanza più bassa.

1.3.4 Esperimenti della Cornell University

Il gruppo dedito allo studio delle biblioteche digitali della Cornell University, nel 2002 presenta un lavoro [4] che, pur non proponendo sostanziali novità dal punto di vista algoritmico, esegue alcune ricerche comunque interessanti alla ricerca di alcuni punti cruciali che debbono essere considerati nella progettazione della propria strategia di visita nel caso si voglia costruire contemporaneamente per vari argomenti un elenco di 25-50 URL rilevanti. Innanzitutto, gli autori individuano quello che è il problema del *tunnelling*, ovvero la necessità del crawler di seguire lunghi percorsi di nodi non rilevanti per poter accedere ad aree di nodi rilevanti. Esperimenti su un crawl di circa 500.000 pagine e susseguenti statistiche provano a dare le seguenti risposte:

- Qual è la lunghezza media di un tunnel? Nel 90% dei casi, sicuramente minore di 10. Inoltre, è possibile trovare pagine rilevanti a qualsiasi distanza da un'altra pagina rilevante

[1] Il termine originale è Merged Context Graph, la cui traduzione letterale è grafo di contesto fuso.

- Nodi con alta valore di aderenza hanno figli con altri valori di aderenza? La risposta è genericamente si: questo è anche dovuto alla tendenza dei siti web su un argomento ad essere distribuiti su varie pagine. Non è invece valido il contrario: nodi con basso valore possono avere figli con alto valore.
- Se dunque il punteggio di un nodo in se non è indicativo sull'aderenza del figlio, può esserlo il percorso del crawler fino alla pagina? Facendo alcune stime sui 3 elementi predecessori, risulta che anche nel caso il genitore di un nodo abbia bassa aderenza, l'eventuale alta aderenza del nonno è un indicatore migliore, mentre nel caso di un padre ad alta aderenza, l'alta aderenza del nonno è soltanto un rafforzativo del valore del padre.

Al termine delle indagini statistiche, il lavoro prosegue sfruttando questi tre punti e confrontando il crawl contro alcune delle metodologie standard.

1.3.5 Modelli probabilistici

Nel 2004 Liu ed altri [19] propongono l'utilizzo di metodi probabilistici quali modelli markoviani nascosti e campi stocastici condizionali unitamente a tecniche di apprendimento non supervisionato al fine di creare delle strategie di crawl performanti. [2] Il lavoro è motivato dal tentativo di sfruttare la dipendenza semantica tra le varie pagine, tentando di sintetizzare asserzioni del tipo "Le pagine di Sport puntano sicuramente più facilmente a pagine di Vela che a pagine sulle Neuroscienze". Innanzitutto, l'utente genera dati per l'apprendimento fornendo un campione di pagine di interesse unitamente alla struttura del web con cui ha navigato dall'una all'altra (indipendentemente dal grado di interesse). Sul solo contenuto delle pagine del campione viene compiuta la Latent Semantic Analysis per ridurre la dimensionalità degli elementi e le pagine vengono clusterizzate utilzzando l'algoritmo K-means in 5 cluster differenti. Successivamente si crea un modello markoviano nascosto il cui spazio degli stati è un insieme $T_0, ..T_{n-1}$ che rappresenta le n distanze da una pagina di interesse e il cui spazio delle osservazione è costituito dall'appartenenza ad uno dei cluster di cui sopra. In questo modo il modello cerca di ottenere la probabilità che, data l'appartenenza della pagina ad un cluster, essa sia ad una certa distanza. I parameteri del modello markoviano vengono stimati utilizzando gli archi di navigazione del campione dell'utente e l'appartenenza delle rispettive pagine ai vari cluster. Il modello viene dunque utilizzato nella seguente maniera: si sceglie una pagina e la si clusterizza, generando l'osservazione del modello markoviano; successivamente dal modello si estrae la più probabile sequenza di stati del modello data la seguenza di osservazioni composta dal percorso che finisce sulla pagina appena estratta (questo è fatto facilmente con un riadattamento dell'algoritmo di

[2] Questo lavoro verrà ripreso nel capitolo 2.

Viterbi). In questo modo si può aggiornare (o dare) la previsione di distanza da una pagina rilevante dei figli della pagina attuale. Il crawler possiede dunque una coda con priorità, i cui valori sono dettati dal modello markoviano, da cui estrarre la nuova pagina da esplorare. Questo modello risulta essere particolarmente ben performante secondo un paio di esperimenti a confronto con il Best-First-Search.

1.4 Modelli a confronto

La letteratura finora presentata risulta eterogenea e difficilmente confrontabile, sia per i diversi tipi di dati necessari all'addestramento dei modelli quando esistenti, sia per il fatto che non sono disponibili implementazioni delle varie metodologie che permettano di produrre uno studio comparativo usando gli stessi parametri di partenza per la visita e utilizzando direttamente Internet. Inoltre, le metodologie di valutazione dell'efficienza della visita sono anch'esse eterogenee, dovuto anche alla necessità di stimare la bontà del modello non avendo a disposizione avversari ottimali.

Capitolo 2

Due modelli d'interesse

Questo capitolo si occupa di descrivere dettagliatamente i due modelli di principale interesse per la discussione della tesi, ovvero quello dei grafi contestuali e quello basato su modelli probabilsitici (in particolare, su modelli markoviani nascosti).

Al fine di poter comprendere chiaramente gli algoritmi con grafi di contesto e con modello markoviano, questa sezione presenta un riassunto degli elementi necessari alla costruzione delle strategie di visita. I Classificatori bayesiani vengono utilizzati da entrambi i modelli, mentre clustering, modelli markoviani nascosti e indicizzazione della semantica latente sono caratteristici soltanto del lavoro di Liu ed altri.

2.1 Rappresentazioni di documenti

Ogni documento testuale può essere rappresentato come un vettore nello spazio vettoriale le cui componenti è il dizionario (finito) contenente tutti i termini con cui si può descrivere un documento. Destrutturando il documento tramite una fase di *parsing* è possibile ridurlo ad una rappresentazione comunemente detta TF (*term frequency*): una sequenza di indici di termini (univoci all'interno del documento) a cui viene associata il conteggio del numero di apparizioni all'interno dello stesso documento. Dunque il documento

```
<html><body>Sopra la panca la capra campa,<br>
sotto la campa la capra crepa</body></html>
```

eliminando la struttura HTML e assegnando gli indici ai termini in ordine alfabetico a partire da 0, risultato rappresentato da

$$0{:}2,\ 1{:}2,\ 2{:}1,\ 3{:}4,\ 4{:}1,\ 5{:}1,\ 6{:}1$$

o, vettorialmente, da $\boldsymbol{d} = (2, 2, 1, 4, 1, 1, 1)$.

In altre parole, sia W la dimensione del nostro vocabolario e si supponga di porre in biiezione i lessemi contenuti nel vocabolario con gli interi (permettendoci di indicarli come $w_1, ..., w_W$); un documento è un elemento di $\mathbb{R}^W$ e il valore di ogni componente rappresenta il conteggio delle occorrenze del termine all'interno del documento.

Una seconda rappresentazione, supponendo che i nostri documenti appartengano ad una collezione di dimensione M è identica alla precedente ma il valore di d_i è ottenuto come

$$d_i = \frac{\text{count}(w_i, d)}{\sum_{j=1}^{W} \text{count}(w_j, d)} \log \frac{M}{f_i}$$

dove f_i è la *frequenza* del termine w_i, ovvero il numero di documenti in cui ha conteggio non nullo.

2.1.1 Classificatori Bayesiani

Si supponga di avere un insieme finito di *classi* $(c_1, .., c_n)$ in cui si vogliono distinguere i nostri documenti e di voler trovare un meccanismo automatico che, addestrato su un campione di documenti di cui è conosciuta la classe, sia in grado, preso un altro elemento dell'insieme dei documenti, di associarvici una classe. Sia $\tilde{\boldsymbol{d}}$ un generico documento in rappresentazione TF e sia $\boldsymbol{d}$ il vettore a componenti binarie dove d_i è 1 quando $\tilde{d}_i$ è non-zero e zero altrimenti. Un classificatore bayesiano affronta questo tipo di problematica nel seguente modo, dato che:

$$\Pr[C = c_j | D = \boldsymbol{d}] = \frac{\Pr[C = c_j] \cdot \Pr[D = \boldsymbol{d} | C = c_j]}{\Pr[D = \boldsymbol{d}]}$$

è ragionevole assegnare $C(\boldsymbol{d}) = \arg\max_c \Pr[C = c | D = \boldsymbol{d}]$ Notiamo immediatamente che, per quanto riguarda il massimo, la produttoria al denominatore è indipendente dalla classe, pertanto può essere omessa durante il calcolo della classe.

Nella realtà la probabilità che $\Pr[D = \boldsymbol{d} | C = c_j]$ è descritta da una qualche distribuzione congiunta di tutti i W termini, che però non è facilmente modellizzabile. Si introduce quindi il cosiddetto classificatore bayesiano *naive*, che propone di considerare tutti gli attributi come stocasticamente indipendenti. In altre parole,

$$\Pr[C = c_j | D = (d_1, \ldots, d_W)] = \frac{\Pr[C = c_j] \cdot \prod_{i=1}^{W} \Pr[D_i = d_i | C = c_j]}{\prod_{i=1}^{W} \Pr[D_i = d_i]}$$

avendo quindi W variabili bernoulliane indipendenti che rappresentano il modello dei documenti. Questa semplificazione è estremamente generalista ed è quasi sempre violata in pratica, tuttavia il classificatore bayesiano naive

ha delle performance sorprendentemente buone nonostante la semplcità del modello. Una prima giusitificazione di tale successo è descritta in [33].

L'utilizzo in pratica del classificatore richiede però che siano già disponibili i valori di $\Pr[C = c_j]$ e $\Pr[D_i = d_i | C = c_j]$. Si supponga dunque che T_j sia l'insieme dei documenti di *training* appartenenti alla classe c_j. L'articolo originale propone le due seguenti stime:

$$\Pr[C = c_j] = \frac{|T_j|}{\sum_{i=1}^{|C|} |T_i|}$$

e

$$\Pr[D_i = d_i | C = c_j] = \frac{\sum_{\tilde{\boldsymbol{t_j}} \in T_j} \tilde{t}_{ji}}{\sum_{\tilde{\boldsymbol{t_j}} \in T_j} \sum_{k=1}^{W} \tilde{t}_{jk}}$$

Il classificatore in teoria è completo ma diffiiclmente utilizzabile. Quando infatti W è un numero sufficientemente grande il prodotto delle probabilità diventa un numero difficile da trattare con un calcolatore, pertanto si suggerisce di sostiuire i prodotti con l'esponenziazione delle somme dei logaritmi in una base a scelta. Collateralmente, questo implica che non vi possano essere probabilità pari esattamente a zero. A seguito di ciò si suggerisce l'utilizzo dello *smoothing* Laplaciano, riscrivendo dunque le stime come

$$\Pr[C = c_j] = \frac{1 + |T_j|}{|C| + \sum_{i=1}^{|C|} |T_i|}$$

e

$$\Pr[D_i = d_i | C = c_j] = \frac{1 + \sum_{\tilde{\boldsymbol{t_j}} \in T_j} \tilde{t}_{ji}}{W + \sum_{\tilde{\boldsymbol{t_j}} \in T_j} \sum_{k=1}^{W} \tilde{t}_{jk}}$$

2.1.2 Modelli markoviani nascosti

Un processo stocastico è una famiglia di variabili casuali $\{X_t\}$ tutte definite sull'insieme degli *stati* che il sistema descritto può avere. La nostra discussione sarà concentrata sui processi a tempo discreto, dove $t = \{1, 2, \dots\}$ e a stati finiti. Un processo è detto *markoviano* quando:

- La probabilità di transizione del sistema verso un nuovo stato dipende soltanto dallo state attuale dello stesso
- Esiste una distribuzione di probabilità per lo stato iniziale del sistema (X_1).

Ovvero, sia S l'insieme degli stati finiti che il sistema può assumere e t il tempo a cui si osserva il sistema, per una qualsiasi sequenza di stati $s_a, \dots$ vale che

$$\Pr[X_{t+1} = s_a | X_t = s_b | X_{t-1} = s_c | \dots | X_0 = s_z] = \Pr[X_{t+1} = s_a | X_t = s_b]$$

Qualora i valori di probabilità siano indipendenti dal tempo di osservazione del sistema il modello viene detto *omogeneo* e le sue probabilità di transizione possono essere dunque descritte da una semplice matrice $P^{|S|\times|S|}$ dove $P_{ij} = \Pr[X_{t+1} = S_j | X_t = S_i]$. La distribuzione di probabilità iniziale viene invece descritta come un vettore stocastico $\boldsymbol{\pi}$ dove $\pi_i = \Pr[X_0 = S_i]$. Un qualsiasi vettore stocastico π e una matrice a colonne stocastiche P, costruite su uno stesso insieme degli stati S formano la tripla $(S, P, \boldsymbol{\pi})$ e rappresentano una catena di Markov a tempo discreto omogenea.

Un modello markoviano nascosto (a tempo discreto ed omogeno) è invece una quintupla $(A, B, S, Q, \boldsymbol{\pi})$ dove $(A, S, \boldsymbol{\pi})$ è una catena di markov omogenea a tempo discreto che però non è osservabile direttamente, mentre Q e B vengono descritti in seguito. In un modello markoviano nascosto (in breve HMM - hidden markov model), non è possibile osservare il cambiamento di stato del sistema, ovvero non si ha a disposizione i risultati delle estrazioni delle variabili casuali del processo X_t né si conosce la distribuzione iniziale. L'unica informazione disponibile del processo sottostante è S. Al contrario, però, ad ogni cambiamento di stato del processo è possibile osservare una nuova variabile casuale Y_t definita sull'insieme delle *osservazioni* Q (che si suppone finito). La proprietà del modello markoviano nascosto è quella di possedere una matrice $B^{|S|\times|O|}$ dove $B_{ij} = \Pr[Y_t = o_j | X_t = s_i]$, ovvero esista una distribuzione di probabilità delle osservazioni per ogni stato del sistema. I modelli markoviani nascosti, descritti meglio in [24], sono estremamente utili in pratica poichè esiste un algoritmo che risolve ognuno dei tre problemi tipici associati ad un HMM:

1. Sia $O = o_0, o_2, \ldots, o_n$ una sequenza di osservazioni generate da un HMM $(A, B, S, Q, \boldsymbol{\pi})$ di cui si conoscono tutti i parametri (compresi quindi A, B e $\boldsymbol{\pi}$). Qual'e' la più probabile sequenza di stati $X_1, \ldots, X_n$ corrispondenti a tale osservazione?

2. Sia O la stessa sequenza di osservazioni e HMM un modello già parametrizzato, qual'e' la probabilità che la sequenza sia stata generata proprio da quel modello?

3. Sia O una sequenza di osservazioni e si abbia un modello di cui si conscono solo S e Q, quali sono i valori per A, B e $\boldsymbol{\pi}$ tali per cui la probabilità di osservare tale sequenza dato il modello è massima?

Trascurando il secondo e il terzo problema, risolvibili usando gli algoritmi forward-backward e di Baum-Welch rispettivamente, ci focalizzeremo su quello di nostro interesse: il primo, risolvibile tramite l'algoritmo di Viterbi. Si noti innanzitutto che il concetto di alta probabilità di sequenza di stati può essere descritto in vari modi, ma è ampiamente adottato il concetto tale per cui ad ogni elemento della sequenza di osservazione sia associato lo stato che è *globalmente* il più probabile ovvero, data una sequenza di osservazioni

$o_1, \ldots o_t$ si cerca

$$\arg \max_{r_1, s_2, \ldots, r_t \in S^t} \Pr\left[X_1 = r_1, \ldots, X_t = r_t | O_1 = o_1, \ldots, O_t = o_t | \lambda\right]$$

da cui si può ricavare l'algoritmo di Viterbi.

Si rappresenti sinteticamente con λ il modello e sia ancora $r_1, \ldots, r_n$ una sequenza di stati associata alla sequenza di osservazioni $o_1, \ldots, o_n$ di interesse, definiamo ora:

$$\delta_t(i) = \max_{r_1, \ldots, r_{t-1}} \Pr\left[X_1 = r_1, \ldots, X_{t-1} = r_{t-1} X_t = i | O, \lambda\right], \forall i \in S$$

che rappresenta il miglior percorso composto da esattamente t passi che finisce nello stato i, per ogni stato. Ora, data la proprietà del modello sottostante di essere markoviano, se si ha una sequenza di osservazioni lunga k e la corrispondente miglior sequenza di stati, aumentando la sequenza al $k+1$-esimo passo allora la sequenza dei primi $k-1$ stati calcolata per ogni valore del k-esimo rimane sempre ottima e può essere aumentata introducndo il $k+1$-esimo stato. Pertanto è possibile descrivere il processo di calcolo per induzione:

$$\delta_1(i) = \pi_i B_{i(o_1)}$$

e

$$\delta_t(i) = \max_{j \in S} \left[A_{ji} \delta_{t-1}(j)\right] B_{i(o_t)}$$

Parallelamente è necessario mantenere traccia dello stato attraverso cui si arriva a quello ottimo, per cui ci avvialiamo di

$$\rho_t(i) = \arg \max_{j \in S} \left[\delta_{t-1}(j) A_{ij}\right].$$

Presa dunque la sequenza O l'algoritmo di Viterbi utilizza la programmazione dinamica, ovvero calcola e memorizza le funzioni $\delta_1(\cdot), \ldots, \delta_t(\cdot)$ per tutti gli stati in ordine temporale crescente. Terminata la prima fase di si può fare *backtracking* della sequenza degli stati corrispondente ai massimi locali, ovvero costruire la sequenza $r_0^*, \ldots, r_n^*$ come

$$\begin{aligned} r_n^* &= \arg \max_{i \in S} \delta_n(i) \\ r_t^* &= \rho_{t+1}(r_{t+1}^*) \qquad t = n-1, n-2, \ldots, 1 \end{aligned}$$

2.1.3 Indicizzazione della semantica latente

L'indicizzazione della semantica latente (LSI) è una tecnica strettamente correlata alla tecnica statistica denominata analisi delle componenti principali. Si abbia una collezione documentale rappresentata da D documenti su un vocabolario di W termini. Scegliendo una qualsiasi rappresentazione di un documento come vettore (colonna) sullo spazio dei termini a coefficienti

reali, l'intera collezione documentale può essere rappresentata da una matrice composta dalla giustapposizione dei documenti in forma vettoriale; ci si riferisce alla matrice appena composta come "matrice termini-documenti" X di dimenioni $W \times D$ e con $X_{(i)}$ ci si riferisce all'i-esima riga di X, che rappresenta l'occorrenza di un termine all'interno dei vari documenti. Come è noto, ogni matrice reale può essere espressa come la sua decomposizione a valori singolari, per cui $X = U\Sigma V^T$ dove U e V sono due matrici ortogonali e Σ è la matrice diagonale dei valori singolari. Supporremo inoltre che i valori di Σ nella decomposizione d'interesse siano ordinati per valore assoluto decrescente.

La teoria alla base di LSI asserisce che, decomponendo X come sopra e ricostruendo il valore $X_k = U_k \Sigma_k^T V_k^T$ dove Σ_k è una riduzione di Σ in cui soltanto i primi k valori della diagonale sono mantenuti, si ottiene la migliore (secondo la norma di Frobenius $\sum_{i,j=1}^{m,n} X_{ij}^2$) riduzione di rango k della matrice originale.

La matrice X_k è dunque una matrice che associa ad ogni termine un vettore di al più k elementi non nulli. Le k colonne sono, secondo l'ipotesi di LSI tale per cui esista una semantica latente all'interno della matrice X, dei *concetti* che sono espressi dalla co-occorrenza dei termini.

Si può mostrare empiricamente che avendo eliminato gli ultimi valori singolari in ordine di grandezza la riduzione di rango preserva le direzioni su cui le co-occorrenze dei termini si distribuiscono con maggior varianza, dunque non solo si ottiene una riduzione di dimensionalità degli spazi, che facilita le computazioni, ma si mantengono anche le associazioni maggiormente caratteristiche.

Pertanto, data una matrice termine-concetti è possibile trasportare un nuovo documento $\boldsymbol{q}$ nello spazio dei concetti per poi operare in quello spazio ridotto, ad esempio, il calcolo della similarità di due documenti. Per eseguire tale operazione basta calcolare $\tilde{\boldsymbol{q}} = \Sigma_k (U_k)^T \boldsymbol{q}$.

2.1.4 Clustering con K-Means

Il clustering è una tecnica di apprendimento non supervisionato che cerca di raggruppare i dati in input (che supponiamo essere documenti in rappresentazione vettoriale) in aree tali per cui elementi "simili" siano nello stesso cluster. Ogni elemento viene associato ad uno ed un solo cluster. Gli algoritmi di clustering sono molteplici e sono quasi tutti caratterizzati da avere in ingresso soltanto i dati e il numero K dei cluster in cui organizzare gli elementi di uno spazio vettoriale R^d, $\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}$. Uno dei più diffusi algoritmi di clustering e' il K-Means, che si basa sul concetto di *centroide* del cluster, ovvero dell'elemento dello spazio costituito dalla media matematica di tutti gli elementi del cluster. Indicheremo il centroide di ogni cluster come $\boldsymbol{c_j}$ e il

cluster associato ad ogni elemento $l(\boldsymbol{x}_i)$ Lo scopo dell'algoritmo K-means è infatti quello di minimizzare la *varianza totale* dei dati, ovvero la funzione obbiettivo

$$\sum_{j=1}^{K} \sum_{\boldsymbol{x} \mid l(\boldsymbol{x})=j} \|\boldsymbol{x} - \boldsymbol{c}_j\|^2$$

il cui gradiente è zero quando ogni cluster ha come centroide la media aritmetica dei dati ad esso associato. Il K-means opera dunque così:

1. Si selezionano K elementi a caso tra gli $\boldsymbol{x}$ e li si assegna come centroidi.
2. Si associa ognuno degli elementi rimanenti al cluster il cui centroide è più vicino, secondo una qualche metrica, all'elemento dato.
3. Si ricalcola il centroide di ogni cluster facendo la media dei suoi elementi, dopodichè si torna al punto 2.
4. Nel caso i centroidi non si spostino più di una data soglia dopo il ricalcolo delle medie l'algoritmo termina.

Si noti che le performance dell'algoritmo dipendono fortemente dal tipo di distanza scelta, dalla distribuzione soggiacente ai dati e da K; solitamente la distanza scelta è quella euclidea e K viene scelto empiricamente dall'utente.

Una volta addestrato K-means su un set di dati, è possibile mantenere soltanto la posizione dei centroidi e classificare un nuovo documento semplicemente calcolando il centroide più vicino.

2.2 Grafi contestuali

Con i prerequisiti finora descritti, siamo in grado di dare una descrizione più dettagliata del metodo dei grafi contestuali, cercando di inquadrare il metodo all'interno della letteratura antecedente e susseguente.

Il grafo del web è complesso e difficilmente mantiene una struttura regolare, esprimendo spesso molte componenti connesse e pochissime fortemente connesse; l'incapacità di percorrere gli archi del grafo all'indietro risulta quindi uno svantaggio per un crawler: in questo senso, la qualità della visita dipende fortemente dal seme di crawl, che determina automaticamente il grado massimo di copertura del grafo. Gli autori si pongono dunque il problema di come recuperare in qualche forma elementi del trasposto del web, per poter effettuare del crawling all'indietro. Sotto l'assunzione che i grandi motori di ricerca abbiano a disposizione una base di dati molto maggiore, si decide di utilizzare la limitata capacità di motori di ricerca esterni (ad es. Google o AltaVista) di permettere agli utenti di ottenere le pagine riferenti a quella inserita.

I grafi contestuali, già descritti nel capitolo precedente, sono in realtà delle visite per ampiezza svolte fino ad una massima profondità K sul trasposto del grafo del web a partire da un gruppo di pagine iniziali a cui ci riferiremo con il nome *radici*, per non confonderle con il seme della visita.; sottolineiamo immediatamente che l'utente si limita ad addestrare il sistema fornendogli le radici, un insieme di pagine di sicuro interesse. Negli esperimenti degli autori i grafi vengono costruiti con al più 300 elementi per strato e per semplificazione si assume che ogni nodo rimanga sempre associato al suo strato più vicino alla radice. Lo scopo della costruzione di un grafo di contesto è il tentativo di raccogliere conoscenza a proposito dei contenuti che si trovano a una certa distanza da elementi interessanti come le radici, sotto l'assunzione che per un documento molto simile ad una radice, i contenuti delle pagine a distanza 1 dalla radice conosciuta e quelli del nuovo documento saranno simili.

I vari grafi di contesto vengono poi organizzati in un unico singolo grafo contestuale generale al fine di combinare le informazioni riguardanti la struttura inter-radice e poter addestrare dei classificatori per determinare a posteriori lo strato a cui una pagina possa verosimilmente appartenere. Si supponga che il grafo generale abbia massima profondità N. Esistono quindi $N+1$ classi a cui un documento può appartenere: le distanze da 0 a $N-1$ comprese e la distanza "N o più". Gli autori fanno notare come mentre sia semplice costruire un modello da esempi delle prime N classi di distanza, l'ultima non può essere affrontata nella stessa maniera. A seguito di ciò si decide di utilizzare un classificatore bayesiano addestrato e che classifica sulle prime N classi aggiungendo però un insieme di soglie σ_j tale per cui se per un generico documento $\boldsymbol{d}$

$$\Pr[C = c_k | D = \boldsymbol{d}] \leq \sigma_k \qquad \text{dove } k = \arg\max_c \Pr[C = c | D = \boldsymbol{d}]$$

allora $\boldsymbol{d}$ è da considerarsi nella classe extra. La soglia di ogni strato viene determinata come

$$\sigma_j = \min_{\boldsymbol{d} \in D_j} \Pr[C = c_j | D = \boldsymbol{d}]$$

dove D_j è l'insieme degli elementi di training allo strato j.

I classificatorii bayesiani naive vengono addestrati e applicati a documenti che hanno subito una prima fase di preprocessing; innanzitutto si rappresentano i docuemnti in TF-IDF, eliminando però alcuni termini inutili quali le *stopwords* (ad es. ciao, che) e raggruppando secondo regole linguistiche (*stemming*, meglio descritto nel prossimo capitolo) che ad esempio eliminano i plurali o portano i verbi all'infinito. Al termine, solo le 40 componenti con maggior valore TF-IDF vengono mantenute e le altre vengono azzerate. Il classificatore dello strato zero è dunque, secondo la dicitura di Chakrabarti, il *classificatore* di pagine interessanti.

Il *distillatore* è invece l'elemento che estrae dalle diverse code: si ricorda infatti che ogni elemento classificato in uno degli $N+1$ strati comporta la

classificazione delle nuove pagine figlie come subito appartenenti allo strato più interno. All'interno di una stessa coda ha maggiore precedenza il nodo cono verosimiglianza verso la propria classe più alta.

La pubblicazione originale non indica se le stime di nodi già in coda vengano abbassate o meno. Essa indica però che il modello dei grafi di contesto viene continuamente aggiornato: ogni qualvolta si sfila un elemento della coda 0, i classificatori vengono aggiornati con il nuovo grafo di contesto. Data la natura lineare e la semplicità dei parametri dei classificatori naive bayes questa operazione è computazionalmente poco costosa.

2.3 Modello probabilistico

Il modello probabilistico di Liu è più complesso dal punto di vista architetturale e dal punto di vista della tipologia di dati necessaria all'addestramento. Sotto questa visione il crawler è modellato come un visitatore del grafo del web a cui viene in ogni momento proposta la scelta tra tutti gli elementi uscenti dalla frontiera. Secondo una sua policy interna il crawler seleziona la pagina web e provoca un'osservazione da parte del modello markoviano nascosto (il cluster cui si associa la pagina) e contemporaneamente una transizione da uno stato di distanza ad un'altro.

Più in dettaglio, l'HMM è costruito con $Q = \{q_1, \ldots, q_K\}$ dove K è il numero di cluster, $S = \{T_0, \ldots, T_{n-1}\}$, che rappresentano le n distanze da una pagina interessante, una distribuzione degli stati inziale uniforme e le due matrici A e B, i cui valori vengono dalla fase di apprendimento ora descritta.

Inizialmente, l'utente viene monitorato durante la sua fase di navigazione del web e per ogni pagina che egli visita viene proposta la possibilità di annotarla come pagina di interesse. In questo modo si costruisce un piccolo sottografo del web, denotato dai link esplorati e dalle pagine di interesse e non essi associate. Questo sottografo (denominato anche grafo dei concetti), viene esplorato a partire dalle pagine interessanti con una visita per ampiezza, marcando ogni pagina con un valore di distanza dalla più vicina pagina interessante.

La fase successiva implica la generazione del modello markoviano e del clustering. A priori, tutti i contenuti delle pagine di learning vengono utilizzati per generare la matrice di LSI, attraverso cui vengono generalizzati tutti i documenti. Applicando la fase di *learning* del K-Means ai documenti generalizzati, si generano solitamente 5 cluster (dove il valore è scelto completamente empiricamente).

Le probabilità del markov model vengono stimate avvalendosi dei valori di distanza ricavati dalla visita del grafo dei concetti, ovvero

$$A_{ij} = \frac{|L_{ij}|}{\sum_{k=0}^{n-1} |L_{kj}|}$$

dove L_{ij} è l'insieme degli archi nel grafo dei concetti che passano da un elemento di distanza i ad uno di distanza j, e

$$B_{ij} = \frac{|N_{ij}|}{\sum_{k=0}^{K} |N_{kj}|}$$

dove N_{ij} è l'insieme delle pagine nel grafo dei concetti di distanza j associate al cluster i. Come è intuitivo, per avere delle buone stime di probabilità è necessario avere un grafo dei concetti abbastanza vasto. Gli esperimenti originali usano almeno 200 pagine di cui 30 a distanza 0.

Terminato l'addestramento il crawler parte dal seme di visita (che si suppone essere a distanza n), e inserisce tutti i suoi figli come se fossero di distanza $n-1$. Una volta estratta e visitata una nuova pagina w, ad esso viene associato un cluster, tutte le variabili parziali δ_w e, utilizzando l'algoritmo di Viterbi, uno stato, che determina la distanza dei figli che vengono inseriti in coda.

Si noti che data la natura del problema di crawling l'algoritmo di Viterbi non viene applicato alla reale sequenza di visita, bensí agli antenati (all'interno della visita) del nuovo nodo appena estratto. Ad esempio, se si percorrono gli archi $(1,2);(1,4);(2,3);(4,5)$, quando si visita la pagina 5 la sequenza di osservazioni usata per Viterbi sarà quella associata alle pagine $\{1,4,5\}$.

L'ultimo parametro mancante è la massima distanza (ovvero il numero di stati) considerati dal modello. Su questo valore gli autori pongono una particolare attenzione in quanto tanto maggiore è la profondità raggiunta tanto le limitate capacità di generalizzazione di HMM e soprattutto clustering hanno effetto sulla visita: si discute della variazione delle performance del crawl con lo stesso numero di stati su argomenti diversi, asserendo che per ogni istanza del crawl, intesa come grafo di concetto, esiste una profondità massima ottimale. Negli esperimenti presentati i valori di profondità massima variano tra 4 e 7.

Capitolo 3

Collezione per gli esperimenti

Con l'eccezione di [20], il cui scopo è appunto di porre a confronto secondo varie metriche le differenti strategie di crawling, le pubblicazioni scientifiche riportanti esperimenti di focused crawling sono tra di loro difficilmente comparabili, come è difficile confrontare un proprio crawler con uno di quelli descritti. Il motivo di base è semplice: gli esperimenti vengono condotti o su archivi virtuali su cui si simula la visita dalla struttura non perfettamente definita oppure direttamente su Internet (il che rende completamente impossibile la riproduzione dei tentativi fatti). Per quanto esperimenti su Internet diano al lettore poi l'idea della reale efficacia dei metodi di crawling, si tralascia sempre la possibilità di fornire risultati ottenuti collezioni su cui simulare la visita disponibili a coloro che vogliono confrontarsi. Scopo di questo capitolo è dunque descrivere la teoria, la fase implementativa e gli effettivi dati utilizzati per la costruzione di una collezione (che chiameremo grafo documentato). pubblicamente accessibile e rappresentativa della reale situazione di Internet. Il risultato è un pacchetto software per il linguaggio Java per la costruzione e l'accesso a un grafo del web inglese collezionato da un crawler generico nel mese di aprile 2007 il cui contenuto e struttura sono stati appositamente compressi per permettere esperimenti di focused crawling realistici in ambienti invarianti (e quindi per dare riproducibilità scientifica ad eventuali esperimenti). Nel affrontare questo percorso è di fondamentale interesse la scelta di quali tipologie di dati inserire, come la sorgente di dati e come architettare la struttura software e organizzare i dati su disco in modo da ottenere un impronta di dimensioni contenute (sufficiente da essere distribuita su normali supporti ottici).

All'interno della collezione dovranno essere presenti i due elementi base necessari ad un focused crawler: il grafo della rete e i contenuti dei singoli nodi. Per quanto riguardo il primo elemento l'infrastruttura per l'accesso, l'archiviazione compressa dei grafi del web è già stata sviluppata in linguaggio Java en un pacchetto opensource di nome WebGraph [28] (si veda anche [6]).

Riguardo il secondo elemento, supporremo per semplicità di eseguire un

crawl soltanto tra pagine (X)HTML e tralasceremo gli elementi altri tipi di contenuto che sono privi di connessioni uscenti e che quindi sono comunque degli elementi di termine dei percorsi disegnati dal crawler. Pertanto, lo scopo sarà quello di creare una struttura dati su disco con buoni fattori di compressione e grande rapidità d'accesso e la cui architettura d'accesso in lettura permetta di allineare i nodi un grafo semplice costruito con Webgraph e gli elementi di quello che denominiamo ora *archivio (diretto)*, ovvero la collezione documentale delle pagine web debitamente compressa e coronata di strutture accessorie per il recupero.

3.1 Archivio diretto

Un archivio diretto contiene una versione destrutturata e già elaborata del contenuto testuale di documenti, in formato opportuno. Si supponga di avere la collezione di tutti i termini (detta *dizionario*) della collezione documentale di nostro interesse e di associare ad ogni termine un indice numerico, tralasciando a dopo il problema di come creare lo schema di assegnamento.

Un archivio diretto contiene principalmente la ricostruzione in formato TF di un insieme di documenti costruiti su un qualche dizionario. Il file di dati contiene, per ogni documento, un indice numerico univoco all'interno dell'archivio, seguito dal numero di termini univoci contenuti nel documento e dalla rappresentazione TF dello stesso; ai fini della costruzione dell'indice, un documento è almeno 2 numeri interi seguiti da K coppie di numeri interi. La giustificazione all'utilizzo della rappresentazione TF del documento è data dalla semplice osservazione che al fine di confrontare due sistemi di classificazione testuale che devono comunque destrutturare i documenti, le fasi di *parsing* e conteggio delle frequenze devono essere assolutamente identiche, altrimenti gli esperimenti si svolgeranno su data set intrinsecamente differenti.

Ancor peggio capita qualora, come nel nostro caso, si scelga di applicare tecniche di riduzione e rimanipolazione dei termini, quali *stemming* o eliminazione delle *stopword*: cambiare l'elenco delle stopword o variare lo stemmer può portare inavvertitamente a profondi cambiamenti nei risultati della classificazione.

3.2 Distribuzione dei termini

Un buon metodo per comprimere la sola rappresentazione TF del documento è il seguente. Innanzitutto si rinumerano tutti i termini in ordine decrescente di frequenza[1] Quando è necessario comprimere un nuovo documento le coppie $\langle t, f \rangle$ rinumerate vengono ordinate per t crescente e al posto degli indici dei

[1]La frequenza è il numero di documenti nell'archivio in cui il termine appare almeno una volta.

termini vengono codificate le loro differenze iterate. Quindi, al posto di $1, 2, 4, 7$ vengono codificati i valori $1, 1, 2, 3$. Le frequenze invece vengono codificate esattamente come sono. Visti gli attenti studi già fatti sui codici di compressioni per sequenze di interi, è utile considerare la distribuzione di probabilità dei valori degli scarti per scegliere il codice migliore.

Il processo da analizzare è dunque il seguente: si prenda la distribuzione di probabilità dei termini all'interno della collezione documentalee si estraggano senza ripetizione k termini (corrispondenti a un documento). Sia $X_1, \ldots, X_k$ il campione (non iid) rappresentate i termini estratti. Si definiscono $Y_1 < Y_2 < \cdots < Y_k$ le k statistiche d'ordine corrispondenti alla sequenza di termini riordinata. La distribuzione di interesse è uindi, quella delle variabili $Z_i = Y_i - Y_{i-1}$.

La distribuzione iniziale dei termini, per quanto dipendente dalla collezione documentale in sé, può essere approssimata, come descritto da molti lavori e confermato anche in questo caso da evidenza empirica, con una distribuzione Zipfiana, dove N è il numero di termini nel vocabolario, s è un parametro reale positivo ed $H_{n,s}$ rappresenta il n-esimo numero armonico generalizzato di esponente s,

$$\Pr[X = i] = \frac{i^{-s}}{H_{n,s}}$$

La zipfiana può essere sostituita dalla distribuzione zeta, portandola all'infinito, che per un parametro $s > 1$, vale:

$$\Pr[X = i] = \frac{1}{\zeta(s)} i^{-s}$$

Sempre assumendo la semplificazione dell'indipendenza dell'apparizione dei termini è dunque possibile scriverela funzione di densità di una coppia di statistiche d'ordine (Y_a e Y_b con $a < b$) e della funzione di ripartizione di una singola statistica:

$$F_{Y_a}(y) = \sum_{i=a}^{k} \left(\begin{array}{c} k \\ i \end{array} \right) F_X(y)^i (1 - F_X(y))^{(k-i)}$$

e

$$\begin{aligned} f_{Y_a,Y_b}(x,y) &= \frac{k!}{(a-1)!(b-a-1)!(n-b)!} F_X(x)^{a-1} (F_X(y) - F_X(x))^{b-a-1} \times \\ &\times \quad (1 - F_X(y))^{k-b} f(x) f(y) \end{aligned}$$

Per statistiche adiacenti, come quelle di interesse, si arriva dunque a:

$$f_{Y_a,Y_{a+1}}(x,y) = \frac{k!}{(a-1)!(k-a-1)!} F(x)^{a-1} (1 - F(y))^{n-a-1} f(x) f(y)$$

la cui forma specializzata per la Zipf o per la Zeta non è facilmente maneggiabile.

Seguendo un processo leggermente alternativo si può ricavare un'altra formulazione per le probabilità delle singole statistiche d'ordine, anche discrete, considerando perÃšche data la proporzione della taglia del campione (circa 200, nella realtà) rispetto al numero di masse della distribuzione (circa 100 milioni) il campionamento può essere considerato con reimmissione. In particolare considerando di lavorare sulle variabili del campione:

$$\Pr[Y_a = x] = \Pr[\text{a-1 di n-1 variabili} < \text{k, 1 uguale a x}] =$$
$$= n \left(\begin{array}{c} n-1 \\ a-1 \end{array} \right) F(x-1)^{a-1}(1-F(x))^{n-a} f(x)$$

Da questo si può operare su coppie di statistiche considerando che l'evento $P(Y_a = x, Y_{a+1} = y)$ ha probabilità non nulla se e solo se $x < y$, $Y_a = x$, $Y_{a+1} = y$ e nessuna variabile del campione ha valori compresi tra x ed y. Ovvero il campione di n variabili si divide in: 1 di valore x, 1 di valore y, $a-1$ minori di x, $n-a-1$ maggiori di y. Da cui:

$$\Pr[Y_a = x, Y_{a+1} = y] = \left(\begin{array}{c} n \\ 2 \end{array} \right) \left(\begin{array}{c} n-2 \\ a-1 \end{array} \right) F(x-1)^{a-1}(1-F(y))^{n-a-1} f(x) f(y)$$

Da notare che se è di nostro interesse la distribuzione degli scarti essa è indipendente dalla coppia di statistiche d'ordine considerata. Da cui $\Pr[Y_b - Y_a = d] = \sum_{x=1}^{N-d} \sum_{i=1}^{n-1} \Pr[Y_i = x, Y_{i+1} = x+d]$, ovvero:

$$\Pr[G = d] = \sum_{x=1}^{N-d} \sum_{i=1}^{n-1} \left(\begin{array}{c} n \\ 2 \end{array} \right) \left(\begin{array}{c} n-2 \\ i-1 \end{array} \right) f(x) f(x+d) F(x-1)^{i-1}(1-F(x+d))^{n-i-1}$$

che ancora si rivela essere un'espressione difficile da maneggiare.

Una semplificazione utilizzata sovente quando si affronta il problema degli scarti consiste nel ignorare il processo di estrazione dei dati e limitarsi a considerare la probabilità che l'estrazione di una coppia di elementi abbia distanza k, ovvero

$$\Pr[G = d] = \frac{1}{H_{n,s}^2} \sum_{i=0}^{W} i^{-s}(i+d)^{-s}$$

Non essendo stata approfondita ulteriormente la teoria, anche data la lateralità del problema rispetto al tema centrale della tesi, si è passati a dare una evidenza empirica della seguente:

Congettura. *La distribuzione della variabile G che rappresenta la larghezza degli scarti tra indici di termini in un documento in rappresentazione TF, i cui termini sono numerati in ordine decrescente di frequenza di apparizione globale e che empiricamente si sa seguire una power-law, è una power-law*
A supporto di ció é stato creato un primo eperimento utilizzando il software R [26]. La collezione vera e propria confermerà ulteriormente questa congettura. L'esperimento prevede i seguenti passi:

```
1. Genera una zipfiana su N elementi di esponente k
2. Crea S campioni senza reimmissione di taglia n
3. Ordina ognuno degli S campioni
4. Sostituisci i valori di ogni campione con gli scarti
5. Conteggia le frequenze di ognuno dei valori degli scarti
6. Disegna su scala log-log la f.d.p. empirica degli scarti
7. Stima l'esponente della power-law
```

Data la particolarità della distribuzione, per il passo 7 si utilizza lo stimatore di massima verosimiglianza descritto in [3] (equazione 26). Supposto di conoscere la dimensione della distribuzione (W) e denotando con $x_1, .., x_m$ il campione di m elementi estratti dalla zipfiana di interesse, l'MLE per l'esponente $\hat{k}$ risolve la seguente equazione:

$$\frac{1}{m}\sum_{i=1}^{m}\ln x_i + \frac{\frac{\partial}{\partial \hat{k}}H_{W,\hat{k}}}{H_W,\hat{k}} = 0$$

che deve essere risolta numericamente. In questo caso si decide di usare il metodo di Newton-Rhapson. Innanzitutto si definiscano le derivate prime e seconde dei numeri armonici generalizzati in funzione dell'esponente k:

$$H'_{W,k} = \frac{\partial}{\partial k}\sum_{i=1} W i^{-k} = \sum_{i=1}^{W}\frac{\partial}{\partial k}e^{-k\ln i} = -\sum_{i=1}^{W}\ln(i)i^{-k}$$

e

$$H''_{W,k} = \frac{\partial^2}{\partial k^2}\sum_{i=1}^{W} i^{-k} = \sum_{i=1}^{W}\ln i\frac{\partial}{\partial k}e^{-k\ln i} = \sum_{i=1}^{W}\ln^2(i)i^{-k}$$

Da questo si ricava che il metodo di Newton $\hat{k}_{n+1} = \hat{k}_n - \frac{f(\hat{k}_n)}{f'(\hat{k}_n)}$ per la nostra

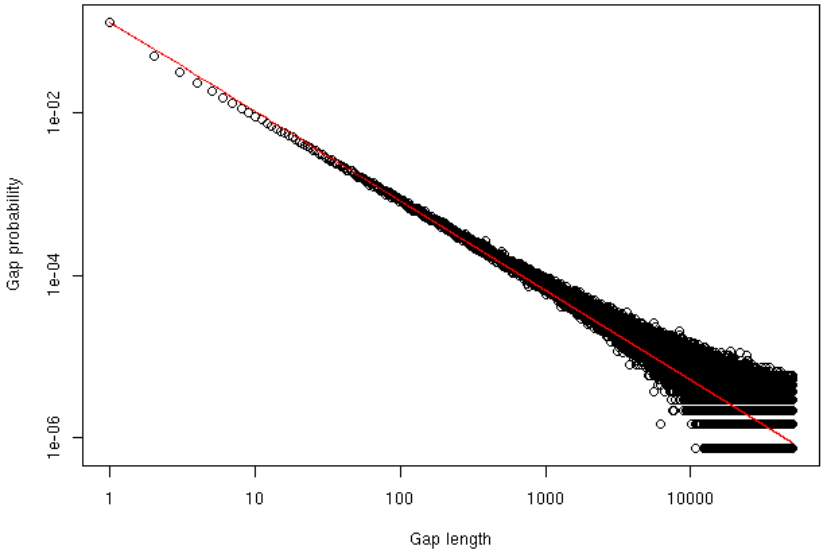


Figura 3.1: La distribuzione degli scarti simulata (scala logaritmica) e la zipfiana di esponente stimato, in rosso.

istanza è, sostituendo $\frac{1}{m}\sum_{i=1}^{m}\ln x_i$ con α

$$\begin{aligned}
\hat{k}_{n+1} &= \hat{k}_n - \left(\alpha + \frac{H'_{W,\hat{k}_n}}{H_{W,\hat{k}_n}}\right)\frac{1}{\frac{\partial}{\partial \hat{k}_n}\left(\alpha + \frac{H'_{W,\hat{k}_n}}{H_{W,\hat{k}_n}}\right)} = \\
&= \hat{k}_n - \left(\alpha + \frac{H'_{W,\hat{k}_n}}{H_{W,\hat{k}_n}}\right)\frac{H^2_{W,\hat{k}i_n}}{H''_{W,\hat{k}_n}H_{W,\hat{k}_n} - H'^2_{W,\hat{k}_n}} \\
&= \hat{k}_n - \frac{H_{W,\hat{k}_n}(\alpha H_{W,\hat{k}_n} + H'_{W,\hat{k}_n})}{H''_{W,\hat{k}}H_{W,\hat{k}} - H'^2_{W,\hat{k}_n}}
\end{aligned}$$

In particolare, la dimensione media di un dizionario su pagine web N è di circa 10 milioni di termini con un esponente tra 1 e 2 circa (empiricamente provato); per ogni documento simulato si estraggono un numero fisso di elementi univoci. Il primo esperimento parte dalla zipfiana usata nella collezione reale, che ha dimensione 11793136 ed esponente stimato su 5 milioni di termini pari a 1.0375 circa e documenti di dimensione media 300. Il risultato di diecimila esperimenti è espresso nel grafico di figura 3.1 e denota la buona aderenza del modello dato ai dati empirici. L'esponente della power-law risultante è stimato essere circa 1.1021.

Per avere una panoramica più generale della situazione, la figura 3.2

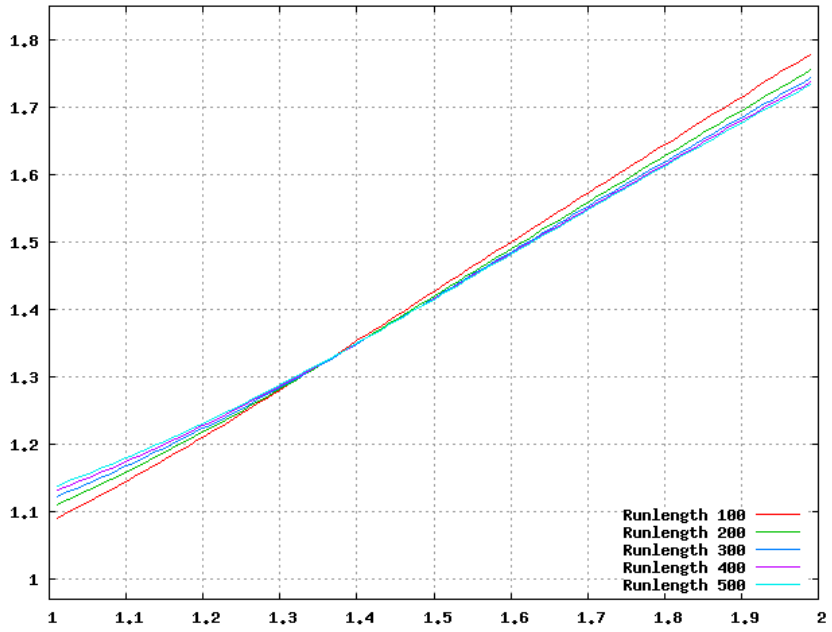


Figura 3.2: Relazione tra l'esponente della power-law generatrice (ascisse) e quella degli scarti.

descrive la legge empirica che lega l'esponente della zipfiana descrivente la distribuzione dei termini e quello riguardante gli scarti con documenti di dimensione differente, lunghezza della distribuzione pari a 10 milioni e 20 mila documenti estratti. Si evince che maggiore è la dimensione del documento minore è invece la salita dell'esponente degli scarti, come che l'esponente degli scarti è maggiore di quello di partenza solo per valori inferiori a circa 1.35. Nel caso di collezioni molto grandi l'esperienza denota che l'esponente della power-law di partenza raramente arriva a 1.2, quindi possiamo congetturare che in pratica gli scarti saranno sempre più piccoli dei valori di partenza.

3.3 Codici istantanei per interi

Come già descritto, è nostro interesse comprimere sequenze di interi dalle differenti distribuzione di probabilità, principalmente zipfiana con esponente inferiore a 2 od uniforme. Un codice istantaneo per interi distribuiti zipfianamente è la famiglia di codici parametrici ζ_k, descritta inizialmente in [7].

Sia $k \in \mathbb{N}^+$ il cosiddetto *fattore di riduzione*, allora un generico $x \in \mathbb{N}^+$, può essere scomposto nella forma $2^h + r$ con $h = \lfloor \log_{2^j} x \rfloor$. In ogni caso, h ed r sono interi. La codifica di x è composta dalla scrittura in codifica unaria di $h + 1$ seguita dalla codifica minaria minimale di r. Per codifica binaria

minimale di un intero $y \in [0, z-1]$ intenderemo l'y-esima parola binaria di lunghezza $\lfloor \log_2 z \rfloor$ se $y < 2^{\lceil \log_2 z \rceil} - z$, altrimenti la $x - z + 2^{\lfloor \log_2 z \rfloor}$.

I codici ζ_k hanno come distribuzione intesa delle zeta e, a seconda dell'esponente s, è possibile selezionare il valore di k che offre la miglior compressione. Unica eccezione è il codice ζ_1, che corrisponde al più famoso Elias γ-code, la cui distribuzione intesa è la power-law $\frac{1}{x^2}$.

Un altro codice per gli interi molto utile è quello proposto da Golomb [16] , anch'esso caratterizzato da un parametro intero b. In questo caso, ogni intero positivo x viene decomposto nella forma $qb + r$, con $q = \lfloor \frac{x-1}{b} \rfloor$. La codifica di x avviene esprimendo q in unario e r in binario minimale.

Un codice di Golomb è ottimo per codificare gli scarti tra elementi di un insieme equiprobabile: si supponga che ogni elemento x dell'insieme di partenza abbia probabilità di apparire nella sequenza da codificare pari a p, allora la probabilità di ottenere uno scarto di lunghezza d è equivalente a trovare una sequenza di prove bernoulliane di peso p di con $d-1$ fallimenti seguiti da un successo, che si distribuisce geometricamente con parametro p. La scelta del parametro b che garantisce la codifica più compatta in media è dato dalla formula (approssimata)

$$b = \frac{\log 2}{\log \frac{1}{1-p}}$$

.

3.4 Compressione su disco

I dati devono essere organizzati su disco in forma compatta e rapidamente accessibile. Sono di interesse 2 principali metodologie d'accesso: l'iterazione su tutta la collezione e l'accesso random al singolo documento dato il suo identificativo univoco. La prima modalità può essere utile in caso l'archivio venga utilizzato in modalità *offline*, mentre la seconda è quella principalmente usata durante il crawling: si esplora il grafo e dal nodo del grafo, che identifica univocamente il documento, si cerca di risalire al contenuto all'interno dell'archivio.

Ispirandosi anche alle tecniche consolidate già presenti in Information Retrieval il modo più semplice per garantire rapidità di accesso mantenendo la struttura di appoggio leggera è la seguente. I documenti all'interno dell'archivio vengono memorizzati in un unico file (ma il metodo è facilmente estedibile) detto *data file* e sono inseriti in ordine di ID [2] di documento crescente. Si sceglie poi un fattore di *skip* (f), di default 32, e in base a quello, durante la costruzione, ogni f documenti inseriti viene scritta in un file di

[2] Si ricordi che comunque gli identificativi dei documenti sono degli interi univoci, ma non necessariamente consecutivi, per tanto 32 documenti non sono 32 interi consecutivi.

supporto la posizione all'interno del data file a cui si trova il documento corrente, insieme con il suo identificativo. In questo modo, il file di supporto viene caricato in memoria all'apertura dell'archivio e con una ricerca binaria si può rispondere rapidamente a quale blocco di f elementi il documento, se presente, appartiene. Ci si posiziona quindi sul datafile all'inizio del blocco e lo si scandisce linearmente, fino a quando non si trova il documento oppure non si trova un identficativo maggiore. In questo modo la complessità in tempo computazionae di accesso al documento è $O(\log n - \log f)$ ma, ancora più importante,vi è soltanto 1 *seek* su disco seguita da una lettura sequenziale (molto rapida grazie all'ormai enorme capacità di read-ahead dei dischi attuali). Il fattore f determina quindi il bilanciamento tra la dimensione della struttura di supporto e la rapidità di accesso.

3.5 Valutazione della compressione

Una valutazione teorica del metodo di compressione finora usata, seguendo un percorso simile a quello di [2] può essere data avvalendosi del calcolo dell'entropia l-aria associata al nostro schema di compressione:

Definizione 3 *Sia $\{p_1, \ldots, p_l\}$ un insieme di valori tali che $p_i > 0$; la loro l'entropia empirica l-aria è definita come*

$$\mathcal{H}(p_1, \ldots, p_l) = -\sum_{i=1}^{l} p_i \log_2 p_i$$

con la convenzione che $0 \log_2 0 = 0$.

Si noti che la definizione di entropia empirica data qui è particolarmente generale. L'aggettivo "empirica" deriva dall'eliminazione del contesto probabilistico tipico dell'entropia di Shannon (basta infatti richiede che gli elementi p_i vengano dalla probabilità di una variabile casuale discreta - la sorgente - ed è possibile tornare alla definizione shannoniana di entropia). Contemporaneamente, questa definizione ingloba quella di entropia empirica su stringhe data da Manzini ed altri in [18], dove si utilizzano le frequenze dei simboli di un alfabeto componenti una stringa come valori di p_i. Il punto di forza delle definizioni di entropie di stampo empirico stà nell'assenza di necessità di un modello generativo dei dati ottenuti: mentre nel modello shannoniano si definisce la sorgente e vi si associa l'entropia di tutte le stringhe che essa può generare, nel modello empirico, data un'effettiva rappresentazione di una stringa si calcola l'entropia come se esistesse una sorgente che ha come stringa tipica proprio quella osservata.

Il prezzo da pagare con questa generalizzazione è dato dall'impossibilità di dimostrare come intrinseche della misura di entropia alcune proprietà, come ad esempio quella riguardante l'estensione n-esima di una sorgente. Per mantenere la stessa operatività diamo quindi la seguente

Definizione 4 *[2] Per una collezione di n elementi dotati di entropia empirica*
$\mathcal{H}_1, \ldots, \mathcal{H}_n$ l'entropia empirica della concatenazione degli elementi è la somma delle entropie l-arie dei singoli elementi: $\mathcal{H} = \mathcal{H}_1 + \mathcal{H}_2 + \cdots + \mathcal{H}_n$

Sia ora A un insieme di n elementi e sia B un suo sottoinsieme di $m \leq n$ elementi; per poter descrivere l'entropia empirica del nostro archivio è necessario dare una definizione di entropia empirica proprio per questo sottoinsieme, mantenendo la massima generalità possibile, ovvero non volendo considerare proprietà intrinseche dell'insieme universo di generazion ma limitandosi a considerare le modalità di costruzione del sottoinsieme. Il nostro scopo è infatti quello di poter estendere ai concetti di insiemi e sottoinsiemi la nozione di entropia, come se esistesse un modello generativo dei sottoinsiemi stessi. Ad ogni sottoinsieme B di A si può associare la funzione caratteristica $\chi_B(a) \in 0, 1 \forall a \in A$; quest'ultima è rappresentabile a sua volta come una stringa binaria di n elementi [3] con esattamente m valori a 1. Possiamo quindi costuire una variabile casuale discreta $\boldsymbol{Y}$ che associa ad ognuna delle 2^n stringhe caratteristiche una probabilità di generazione; siccome è nostra volontà non considerare relazioni tra gli elementi dell'insieme possiamo considerare stocasticamente indipendente l'apparizione degli elementi all'interno del sottoinsieme:

$$\Pr\left[\boldsymbol{Y} = (y_1, y_2, \ldots, y_n)\right] = \prod_{i=1}^{n} \Pr\left[Y_i = y_i\right]$$

dove Y_i risulta essere una variabile bernoulliana che rappresenta la probabilità che $a_i \in B$.

Di conseguenza, l'entropia empirica di generazione di un sottoinsieme di dimensione m viene valutata in base alle probabilità di ottenere stringhe caratteristiche con esattamente m valori ad uno. Scomponendo sugli elementi di B la probabilità della stringa e utilizzando la tecnica di identificare l'entropia di una stringa osservata come se fosse generata da una sorgente che tipicamente (ovvero in media) produce stringhe come quella osservata, diamo la seguente:

Definizione 5 *Sia $A = \{a_1, a_2, \ldots, a_n\}$ un insieme di n elementi e si consideri un sottoinsieme $B \subseteq A$ di m elementi. Si associ ad ogni elemento $a_i \in A$ la variabile casuale X_i bernoulliana che rappresenta l'appartenenza di a_i a B. Si costruisca la famiglia di variabili casuali $Y_i = g(X_i)$ secondo la condizione che*

$$\mathbb{E}\left[\sum_{i=1}^{n} Y_i\right] = m.$$

[3] Si supponga ordinare totalmente secondo un qualche criterio arbitrario gli elementi di A.

L'entropia empirica di B è data da

$$\mathcal{H} = \sum_{i=1}^{n} \mathcal{H}(p_i, 1 - p_i)$$

dove $p_i = \Pr[Y_i = 1]$.

Possiamo estendere la definizione di cui sopra anche ai multi-sottoinsiemi di dimensione m; infatti in tal caso, la funzione caratteristica viene sostituita dalla funzione di molteplicità, ovvero ogni elemento può apparire da 0 a m volte all'interno del multi-sottinsieme che denominiamo C. Notiamo quindi che creare un multi-sottoinsieme di dimensione m da A è equivalente a produrre un sottoinsieme di

$$\tilde{A} = \{(1,1); (1,2); \cdots (1,m); (2,1); \cdots (2,m); \cdots (n,m)\}$$

di cui poi si considera solo il primo elemento di ogni coppia ordinata. Creiamo quindi la famiglia di variabili casuali X_i per $1 \leq i \leq n$ che rappresenta la probabilità di selezionare l'i-esimo elemento di A come facente parte di C un'altra volta ancora o meno. Possiamo quindi definire la famiglia $\tilde{X}_{ij}$ dove $\tilde{X}_{i1} \sim \tilde{X}_{i2} \sim \cdots \sim \tilde{X}_{im} \sim X_i$ e dare la seguente

Definizione 6 *Sia $A = \{a_1, a_2, \ldots a_n\}$ un insieme di n elementi e si consideri un multisottoinsieme C di A di esattamente m elementi. Si associ ad ogni elementi $a_i \in A$ la variabile casuale X_i bernoulliana che rappresenta la possibilità che a_i compaia in C un'altra volta. Si costruisca la famiglia di variabili casuali $\tilde{X}_i j$ come descritto sopra per $1 \leq j \leq m$ e a sua volta la si estenda come $Y_{ij} = g(X_{ij})$ tale per cui*

$$\mathbb{E}\left[\sum_{i=1}^{n}\sum_{j=1}^{m} Y_{ij}\right] = m$$

.

L'entropia empirica di C è dunque data da

$$\mathcal{H} = \sum_{i=1}^{n}\sum_{j=1}^{m} \mathcal{H}(p_{ij}, 1 - p_{ij})$$

dove $p_{ij} = \Pr[Y_{ij} = 1]$.

Date le definizioni di cui sopra possiamo operativamente definire le entropie empiriche di vari tipi di sottoinsiemi e multisottoinsiemi, dipendentemente dal tipo distribuzione che si associa alla scelta di ogni elemento di A come appartenente al sottoinsieme.

Lemma 1 *[2] Sia A un insieme di n elementi e si generi un suo sottoinsieme B di m elementi selezionando gli elementi di A equiprobabilmente, allora l'entropia*[4] *del sottoinsieme è*

$$\mathcal{H} = m \log_2 \frac{n}{m} + (n-m) \log_2 \frac{n}{n-m}$$

Dimostrazione Le variabili X_i rappresentanti l'appartenenza di a_i a B hanno parametro $p_i = \frac{1}{n}$. Se si pone $Y_i = g(X_i) = mX_i$ si ottiene che

$$\mathbb{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbb{E}\left[X_i\right] = m \sum_{i=1}^{n} \frac{1}{n} = m$$

L'entropia delle varie prove bernoulliane, che è $\mathcal{H}(\frac{m}{n}, 1-\frac{m}{n})$, si somma per definizione, da cui $\mathcal{H} = \sum_{j=1}^{n} \mathcal{H}(\frac{m}{n}, 1-\frac{m}{n}) = nH(\frac{m}{n}, 1-\frac{m}{n})$. Il lemma segue applicando la definizione di entropia 2-aria.

Lemma 2 *Sia A un insieme di n elementi e si consideri un suo multi-sottoinsieme B di dimensione totale m, senza vincoli sul numero di elementi univoci di A inseriti in B e si consideri la scelta di ogni elemento di A come componente di B equiprobabile, allora l'entropia di B vale:*

$$\mathcal{H} = m \log_2 m + (mn - m) \log_2 \frac{n}{n-1}$$

Dimostrazione Abbiamo che $\Pr\left[X_i = 1\right] = \frac{1}{n}$ da cui $\Pr\left[X_{ij} = 1\right] = \frac{1}{n}$. Si ponga $Y_{ij} \sim X_{ij}$, da cui

$$\mathbb{E}\left[\sum_{i=1}^{n}\sum_{j=1}^{m} Y_{ij}\right] = \sum_{i=1}^{n}\sum_{j=1}^{m} \mathbb{E}\left[Y_{ij}\right] = \frac{nm}{n} = m$$

. Da questo si ricava che l'entropia vale $nm\mathcal{H}(\frac{1}{n}, 1-\frac{1}{n})$. Applicando la definizione di entropia 2-aria si arriva al lemma.

Lemma 3 *Sia $A = \{a_1, a_2, \ldots, a_n\}$ di n elementi e si consideri un suo multi-sottoinsieme B di dimensione totale m e si consideri che la scelta del t-esimo elemento di A avvenga con probabilità pari a $\frac{t^{-s}}{H_{n,s}}$, per $s > 1$ reale. L'entropia del multisottoinsieme è limitata da*

$$\mathcal{H} \leq m \log_2(eH_{n,s}) + \frac{m}{H_{n,s}} \sum_{i=1}^{n} i^{-s} \log_2 \frac{1}{i^{-s}}$$

.

[4] D'ora in poi ometteremo la dicitura l-aria o empirica quando non necessario

Dimostrazione Si consideri $Y_{ij} \sim X_{ij}$, nel cui caso:

$$\mathbb{E}\left[\sum_{i=1}^{n}\sum_{j=1}^{m} X_{ij}\right] = \sum_{j=1}^{m}\sum_{i=1}^{n} \mathbb{E}\left[X_{ij}\right] = \sum_{j=1}^{m}\sum_{i=1}^{n} \frac{i^{-s}}{H_{n,s}} = m$$

come richiesto. Abbiamo quindi che l'entropia vale esattamente $m\sum_{i=1}^{n}\mathcal{H}(\frac{i^{-s}}{H_{n,s}}, 1-\frac{i^{-s}}{H_{n,s}})$ ovvero

$$m\sum_{i=1}^{n}\frac{1}{H_{n,s}}\left(i^{-s}\log_2\frac{H_{n,s}}{i^{-s}} + (H_{n,s}-i^{-s})\log_2\frac{H_{n,s}}{H_{n,s}-i^{-s}}\right)$$

Consideriamo ora il secondo addendo della sommatoria soltanto e che $1+x \leq e^x$ per $x > 0$,

$$(H_{n,s}-i^{-s})\log_2\frac{H_{n,s}}{H_{n,s}-i^{-s}} = (H_{n,s}-i^{-s})\log_2\frac{H_{n,s}-i^{-s}+i^{-s}}{H_{n,s}-i^{-s}} =$$
$$(H_{n,s}-i^{-s})\log_2(1+\frac{i^{-s}}{H_{n,s}-i^{-s}}) \leq (H_{n,s}-i^{-s})\frac{i^{-s}}{\ln 2(H_{n,s}-i^{-s})} = \frac{1}{\ln 2}$$

Da cui possiamo limitare l'entropia di cui sopra, spezzando invece il primo logaritmo, con

$$\mathcal{H} \leq m\left[\frac{1}{H_{n,s}}\sum_{i=1}^{n}\left(i^{-s}\log_2\frac{1}{i^{-s}}\right) + \frac{\log_2 H_{n,s}}{H_{n,s}}\sum_{i=1}^{n} i^{-s} + \frac{1}{\ln 2 H_{n,s}}\sum_{i=1}^{n} i^{-s}\right]$$

Sostituendo $\frac{1}{\ln 2} = \frac{\ln e}{\ln 2}$ e considerando che $\sum_{i=1}^{n} i^{-s} = H_{n,s}$ si arriva poi al limite del lemma.

Torniamo ora al nostro archivio, composto da D documenti identificati da valori tra 1 e d_{max} ognuno dei quali ha una lunghezza compresa tra 0 ed l_{max} e che contiene termini estratti da un dizionario composto da W termini. Secondo la definizione di cui sopra l'entropia empirica dell'archivio è la somma delle entropie della parte riguardante gli identificativi, di quella riguardante le lunghezze e delle liste dei termini e per tanto può essere calcolata nella maniera più realistica:

$$\mathcal{H}_{dir} = \sum_{i=1}^{d_{max}}(\frac{f(i)}{D}\log_2\frac{D}{f(i)}) + \sum_{j=0}^{l_{max}}(\frac{g(j)}{D}\log_2\frac{D}{g(j)}) + \sum_{k=1}^{D}\sum_{t=0}^{W}(\frac{N(t,k)}{h(k)}\log_2\frac{h(k)}{N(t,k)})$$

dove le funzioni f e g rappresnetano la frequenza di apparizione dell'elemento all'interno dell'insieme dei valori cui appartiene, $N(t,k)$ è il conteggio d'occorrenza del termine t all'interno del k-esimo documento e $h(k)$ è la lunghezza del k-esimo documento espressa come somma delle occorernze.

Con gli strumenti contenuti nei lemmi di cui sopra possiamo, utilizzando alcune evidenze empiriche riguardanti l'archivio, dare una stima teorica ragionevole dell'entropia dell'archivio, in particolare per poi derivare da essa indizi sul tipo di compressione da scegliere per le varie componenti. Il vantaggio di modellare esattamente come dettato dalla valutazione empirica delle proprietà della collezione è quello di ridurre al minimo la parte di approssimazione analitica e contemporaneamente fornire un *upper bound* allo spazio occupato dall'archivio, valutato come

$$D\mathcal{H}_D + D\mathcal{H}_L + \sum_{i=0}^{D} h(i)\mathcal{H}_{list}(h(i))$$

. Per fare questo diamo per vere le seguente assunzioni:

- L'archivio contiene D documenti i cui identificativi vengono dall'insieme $\{1, \ldots, d_{max}\}$ e hanno eguali probabilità di apparire all'interno della collezione.
- Per ogni documento è descritta la sua lunghezza appartenente all'universo $\{1, \ldots, l_{max}\}$ dove la probabilità che il singolo documento abbia lunghezza i è distribuita secondo una zipfiana di esponente $\alpha > 1$, noto.
- Per ogni documento sono descritte le occorrenze dei termini appartenenti ad un dizionario di cardinalità W. Ogni elemento ha possibilità di apparire più volte all'interno di un documento, ma la singola apparizione è regolata da una legge zipfiana distribuita con esponente $s > 1$, noto.

in questo modo infatti possiamo, come descritto sotto, dare una valutazione analitica valida per ogni archivio esistente, basata su un numero molto limitato di parametri. La prima assunzione è forzata in quanto l'assegnazione degli identificativi è un processo aggiunto in fase di creazione ed è completamente impossibile da modellare. Per quanto riguarda la distribuzione dei termini la sezione precedente descrive ampiamente la validità di questa assunzione. Le lunghezze sono purtroppo difficili da modellare, anche su base dei dati empirici. Infatti, la figura 3.3 mostra su scala logaritmica la distribuzione delle varie lunghezze di una collezione reale; la figura 3.4 mostra invece la distribuzione delle lunghezze minori di 150, che formano il quantile 0.6, dando idea della difficoltà di trovare una curva analitica che abbia un buon livello di *fitting* in quella zona. Decidiamo quindi di utilizzare una generica power-law per modellare l'intera distribuzione anche in previsione del fatto che un codice che comprime bene una power-law utilizza pochi bit per valori piccoli, quali quelli che rappresentano buona parte delle probabili occorrenze, è un buon candidato per la compressione delle lunghezze.

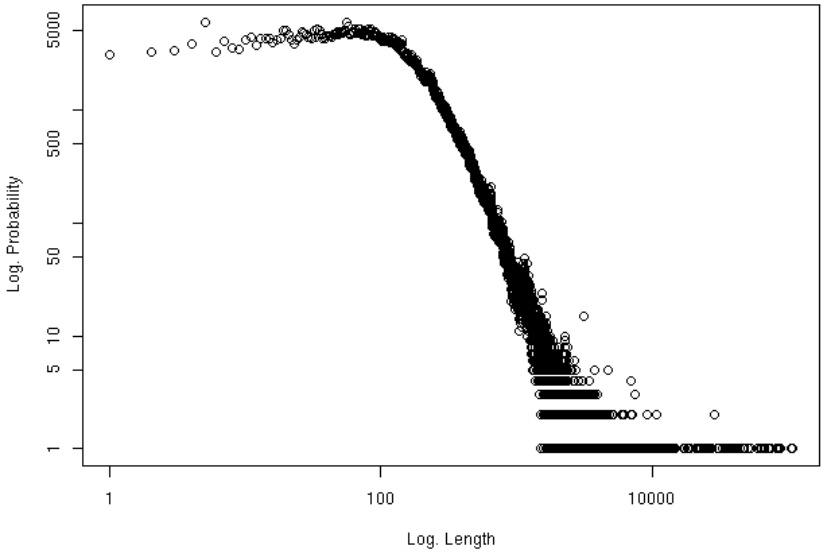


Figura 3.3: Grafico di distribuzione delle lunghezze dei documenti tratto da una collezione di 5 milioni di documenti, scala log-log.

Dando per valide le assunzione di cui sopra possiamo giungere al seguente:

Teorema 1 *Sia A un archivio diretto con D documenti ad identificativi in $\{1, \ldots, d_{max}\}$ distribuiti equiprobabilmente tra di loro, le cui lunghezze sono distribuite secondo una zipfiana di parametro $\alpha > 1$ e dove il j-esimo documento contiene l_j estratti con reimmisisone da un vocabolario di W termini distribuiti secondo una zipfiana di parametro $s > 1$. Allora, definendo $L = \sum_{i=1}^{D} h(i)$, l'entropia empirica dell'archivio rispetta la seguente disuguaglianza:*

$$\begin{aligned} \mathcal{H}_{dir} &\leq \frac{2D + D(\ln H_{l_{max},\alpha} + \ln d_{max}) + L \ln H_{W,s}}{\ln 2} + D \log_2 \frac{1}{D} + \\ &+ \frac{D}{H_{l_{max},\alpha}} \sum_{i=1}^{l_{max}} \left(i^{-\alpha} \log_2 \frac{1}{i^{-\alpha}} \right) + \frac{L}{H_{W,s}} \sum_{j=1}^{W} \left(j^{-s} \log_2 \frac{1}{j^{-s}} \right) \end{aligned}$$

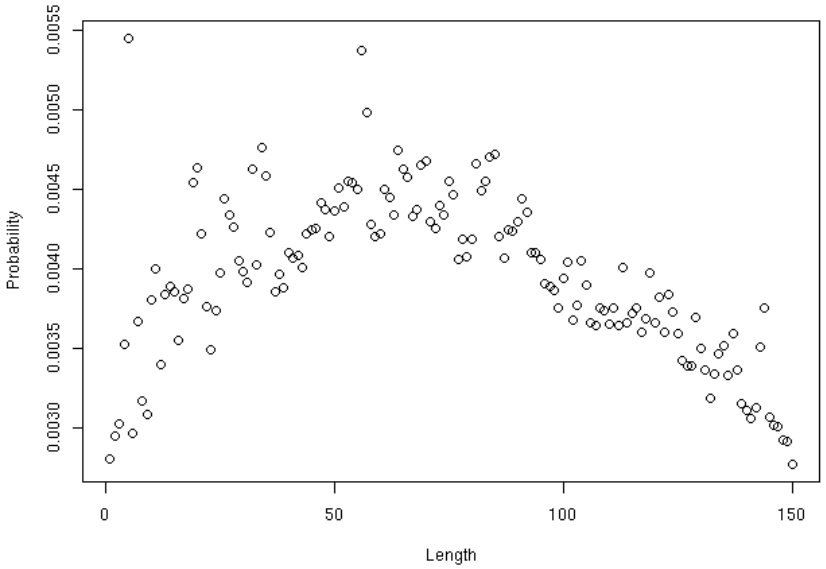


Figura 3.4: Particolare del grafico di figura 3.3, in scala reale, per valori sotto il 150.

Dimostrazione Applicando i lemmi precedenti si trova che la reale espressione dell'entropia l-aria è limitata da

$$\begin{aligned}\mathcal{H}_{dir} &\leq D\log_2\frac{d_{max}}{D} + (d_{max}-D)\log_2\frac{d_{max}}{d_{max}-D} + \\ &+ \frac{D}{\ln 2} + D\log_2 H_{l_{max},\alpha} + \frac{D}{H_{l_{max},\alpha}}\sum_{i=1}^{l_{max}} i^{-\alpha}\log_2 i^{\alpha} + \\ &+ \frac{L}{\ln 2} + L\log_2 H_{W,s} + \frac{L}{H_{W,s}}\sum_{j=1}^{W} j^{-s}\log_2 j^{s}\end{aligned}$$

Approssimando, come nella dimostrazione del Lemma 3, $(d_{max}-D)\log_2\frac{d_{max}}{d_{max}-D}$ con $\frac{D}{\ln 2}$ e riorganizzando i termini si ottiene il teorema.

Da questo limite possiamo trarre un prezioso risultato:

Proprietà 1 *Se gli identificativi dei documenti sono codificati con un codice di Golomb, le lunghezze sono codificate con un codice ottimo per una power law di esponente α (ad esempio lo ζ-code), e le liste di termini sono codificate con un codice ottimo per una power law di esponente s allora la compressione arriva molto vicina all'entropia calcolata analiticamente.*

Per quanto lo stesso tipo di argomentazione sia valida con le liste dei termini generiche, prive di scarti, è banale rendersi conto che, siccome la massa di

probabilità della zipfiana degli scarti, che ha s maggiore, è maggiormente concentrata verso i valori bassi rispetto alla zipfiana delle liste semplici, la codifica degli scarti è sicuramente non peggiore di quella delle liste, anche considerando il fatto che tutti i termini di una sequenza di scarti non possono essere più grandi dei valori originali.

3.6 Implementazione

L'architettura software creata per gli archivi diretti è stata scritta in linguaggio Java con le specifiche versione 1.5 e si concretizza in una libreria contenente classi e primitive per la creazione e l'interrogazione di grandi collezioni in modo efficiente. La creazione della collezione, in particolare, deve poter avvenire da sorgenti diverse e, per supportare moli di dati molto grandi, supportare il più possibile processi off-line anche a discapito della velocità di esecuzione del processo. La fase di numerazione, eliminazione e raggruppamento (*stemming*) dei termini viene separata dalla procedura di creazione dell'archivio stesso, che lavora quanto più possibile con oggetti di tipo solo numerico.

Inizialmente, si concretizza il concetto di *documento di archivio* come l'insieme dei termini univoci e delle frequenze loro assegnate ed un identificativo numerico univoco all'interno dell'archivio e lo si rappresenta attraverso l'interfaccia `ArchiveDocument` e l'implementazione astratta `AbstractArchiveDocument` che fornisce quasi tutte le primitive per inglobare due array paralleli di interi (termini e frequenze) e l'identificativo numerico all'intero di un documento d'archivio. La più semplice implementazione concreta di `AbstractArchiveDocument` è `GenericArchiveDocument` che introduce un campo aggiuntivo contenente un eventuale URI associato al documento.

Di supporto, si fornisce la classe `ASCIIArchiveDocument` la quale è in grado di interpretare come documento d'archivio una linea di un file ASCII costruita nella seguente maniera:

```
1459 6 1:45 2:43 7:99 8:1000 1042:4 9999:9
```

dove il primo campo rappresenta l'identificativo del documento, il secondo il numero di coppie seguenti e ogni coppia è la coppia $< t, f >$ rappresentante l'identificativo del termine secondo il vocabolario dell'archivio e la sua frequenza contenuta all'interno del documento. Le coppie sono ordinate per identificativo dei termini crescente e ogni termine compare una ed una sola volta. Questa rappresentazione testuale funge da protocollo di comunicazione tra un programma che trasforma la sorgente di documenti originale in formato ASCII ed il programma che dalla sorgente ASCII crea un archivio compresso binario.

Il primo elemento di questa catena viene detto *scanner* mentre il secondo è denominato *writer*. Il pacchetto software realizzato contiene un solo *scanner*, nella forma della classe `DirectScan`, che trasforma una `DocumentSequence`

del progetto MG4J [21], in un file di documenti di archivio in formato ASCII. I risultati del *writer* sono dei file su disco che vengono interpretati per ricostruire i documenti d'archivio sotto forma di istanza di un'implementazione dell'interfaccia generica `Archive<T>`, dove `T` è una classe che implementa `ArchiveDocument`. Questa interfaccia permette di accedere all'archivio solo in lettura attraverso le seguenti primitive:

- getDocumentByID che dato l'identificativo numerico di un documento all'interno dell'archivio lo decodifica e lo ritorna sotto forma di istanza della classe `T`. Se l'identificativo corrisponde ad un documento non esistente ritorna `null`.
- getDocumentByIndex che invece accede sequenzialmente ai documenti, ricevendo in input la posizione del documento rispetto all'ordine con cui sono inseriti nel file di achivio. Se l'identificativo è invalido , ritorna `null`..
- numberOfDocuments che ritorna il numero di documenti all'intero dell'archivio.
- numberOfTerms che ritorna invece la dimensione del vocabolario.
- iterator che ritorna un iteratore di sola lettura sull'archivio secondo l'interfaccia `Iterable<T>`.

Le due implementazioni di `Archive` fornite sono `UnsortedBitstreamArchive` e `SortedBitstreamArchive` di cui la seconda è un'estensione della prima. Un archivio bitstream non ordinato è costruito codificando i dati come esposto nelle precedenti sezioni senza porre attenzione all'ordine con cui i documenti vengono inseriti all'interno dell'archivio. Oltre ad alcuni file di supporto (di cui si discuterà in seguito) i dati sono contenuti in un unico enorme file. I documenti sono codificati sequenzialmente in un *bitstream* (ovvero non allineando i dati al byte) nella seguente maniera:

1. Identificativo del documento, in *golomb coding*
2. Numero di coppie termini/frequenza conteute, in codifica ζ_3, già esposta.
3. Ogni coppia viene descritta prima dall'identificativo del termine codificando il valore della differenza dal termine precedente (sempre positivo) in ζ_2, a seguito delle spiegazioni di cui sopra.
4. La frequenza di ogni coppia viene codificata in Elias γ-coding dopo aver sottratto 1.

In un archivio ordinato, invece, i documenti sono inseriti secondo l'ordine crescente dettato dall' identificativo del documento, per tanto l'identificativo di documento viene codificato attraverso lo scarto dal precedente. Gli archivi ordinati inoltre utilizzano un file ausiliario contenete gli *skip*, nella forma dell'ID del documento di inizio del blocco con un codice di Golomb seguito dallo scarto della posizione del precedente, codificato in ζ_3-code.

3.7 Descrizione della collezione

L'implementazione descritta viene applicata ad un crawl generico di 10^8 pagine univoche provenienti dai domini *.uk* collezionato con le infrastrutture universitarie durante l'aprile del 2007, discendendo al massimo di 7 passi all'interno di ogni host e collezionando non più di 15000 pagine per host. Il grafo, che si struttura in 107 milioni di nodi e 3.8 miliardi di archi, viene codificato usando il framework di WebGraph e occupa circa 2 GiB. I contenuti delle pagine (X)HTML occupano nel formato originale (il WARC), compresso con gzip, circa 560 GiB. Le pagine vengono sottoposte ad un processo di destrutturazione tramite parsing dei contenuti, dopodichè ogni termine viene prima posto tutto in minuscolo per poi venir sottoposto ad un processo di stemming per le parole di lingua inglese. Successivamente si eliminano le stopword e i termini quasi-hapax legomena, identificandoli empiricamente come i lessemi (stemmati) presenti in almeno il 12% della collezione e in meno di 1000 documenti. La collezione risultante, ordinata, viene costruita con circa 110 ore di tempo macchina (su macchine Intel XEON 2.8 Ghz con 4 GB di RAM) e risulta in 25 GiB di dati compressi contenenti l'archivio diretto, la mappa che dato un ID di documento ne recupera l'URL, la mappa che dato un qualsiasi termine ne fornisce (nel caso vi sia) la sua numerazione all'interno dell'archivio. Volendo includere anche la mappa da numeri dei termini al loro lessema *stemmed* l'occupazione sale intorno a 30 GiB.

Unitamente alla collezione di grandi di dimensioni vengono create 3 sottocollezioni che rappresentano 3 sotto-grafi del grafo originario rispettivamente di dimensione 10^5, 10^6 e 10^7, ottenuti simulando un crawl superificiale con la seguente tecnica: si esplora il grafo a partire dal seme, considerando per ogni hostname all'interno della collezione un massimo di (rispettivamente) 2500, 3000 e 5000 pagine. Inoltre, partendo dall'home page vengono visitate al massimo (rispettivamente) fino a profondita' 4, 5 o 7, ovvero si discende al più di k nodi all'interno di ogni sito. La visita all'interno del singolo host avviene per profondità mentre tra i vari host ci si muove per ampiezza. Le 3 sottocollezioni sono comunque create con gli oggetti `ImmutableSubgraph` e `Subarchive`, per cui la loro occupazione è di circa 100 MiB in totale, dovendo comunque far riferimento alla collezione originale.

Capitolo 4

Classificazione semisupervisionata

Come descritto nel primo capitolo, un *focused crawler* deve comprendere al suo interno un classificatore in grado di sostituire e/o interagire con l'utente per selezionare quali pagine durante la visita del Web siano effettivamente interessanti. Restringendo la nostra attenzione ai classificatori non interattivi, le tecniche presentate solitamente utilizzano classificatore di tipo bayesiano, non supervisionati o supervisionate. Queste metodologie, se pur efficaci in pratica, prevedono addestramenti o soltanto con un campione di pagine pertinenti o con una coppia di campioni di pagine pertinenti e non pertinenti. Il primo scenario è comunque efficace seppur limitato alle poche indicazioni forinite come campione dall'utente: è impensabile, anche ricorrendo all'analisi della storia di visita, riuscire a ottenere campioni oltre il migliaio di pagine. Il secondo scenario invece richiede una interazione con l'utente in fase di addestramento particolarmente fastidiosa, perchè richiede la presentazione di pagine a ripetizione fino alla raccolta di un numero sufficientemente grande di pagine non interessanti (che per l'utente medio risulta essere tempo sprecato). Nello scenario in cui il crawler sia utilizzato per la costruzione di collezioni orientate ad argomenti generici, la prassi è quella di utilizzare un gruppo di pagine preclassificate positivamente (ad esempio contenute in una categoria dell'Open Directory Project [15]).

Una via ulteriore, poco esplorata anche poiché introdotta negli anni 90 con l'articolo [22], è quella della classificazione semi-supervisionata, in cui invece di utilizzare una coppia di campioni etichettati rispettivamente positivo e negativo, si utilizza un campione di positivi afficanto da uno non etichettato dell'*universo*, ovvero della collezione di tutti i possibili esempi, unione dei positivi e dei negativi. Nel nostro caso ci restringeremo ad algoritmi di classificazione semisupervisionata in cui vengono forniti soltanto un campione etichettato positivo ed uno non etichettato dell'universo. Per quanto possa sembrare assurdo che l'utilizzo di dati non etichettati possa aiutare

l'apprendimento, assumendo i modelli generativi corretti, è possibile integrare l'informazione non etichettata all'interno del processo di apprendimento. Questo tipo di tecnica ha il vantaggio di non richiedere all'utente tempo aggiuntivo e si adatta molto bene al nostro problema: il Web è un enorme bacino da cui è facile prendere un campione non etichettato eterogeneo.

Scopo di questo questo capitolo è introdurre tecniche di classificazione con positivi e non etichettati adatte per la classificazione di testo, di cui se ne selezionano due tra le più promettenti, illustrarne i retroscena teorici ed estendere i lavori di confronto già fatti nei rispettivi articoli scientifici di presentazione, al fine di costruire una tabella comparativa su alcune collezioni testuali realistiche. Il classificatore con il miglior comportamento verrá poi inserito all'interno del crawler.

4.1 Specifiche del problema

Introduciamo dunque più rigorosamente il nostro problema. Si abbia un universo di documenti testuali Z, diviso in due classi non conosciute e disgiunte, C_+, dei positivi, e C_-, dei negativi (o non interessanti). Si supponga di essere in uno scenario plausibile in cui la cardinalitá di C_+ è di svariati ordini di grandezza inferiore a C_- e si generi, con l'aiuto dell'utente, un campione etichettato positivamente di C_+ chiamato P e un campione non etichettato , ben più grande del precedente, del generico Z, denominato U. Scopo delle tecniche presentate è generare un classificatore in grado di distinguere l'appartenenza a C_+ o C_- di un documento dopo aver operato l'apprendimento soltanto su P e U.

A tale scopo si tradurrà, tramite parsing, ogni documento in un corrispondente vettore sui reali in $|V|$ dimensioni, dove V è l'insieme di tutti i termini su cui sono costruiti i documenti di P e U. Ogni elemento del vettore non a zero, ad esempio $\boldsymbol{x}_w$ descrive dunque la frequenza (TF) del termine $w \in V$ all'interno del documento $\mathbf{x}$. L'ordine di numerazione dei termini è completamente arbitrario e non rilevante. Ogni elemento del vettore a zero determina l'assenza del termine. Supporremo inoltre di avere a disposizione la misura $IDF_D(w)$, per $w \in V$ detta Frequenza documentale inversa, che rappresenta il numero di documenti all'interno della collezione D che contiene almeno una occorrenza del termine w.

Per riconoscere la bontà dei nostri classificatori utilizzeremo principalmente 2 misure, l'*accuratezza*, definita come:

$$\text{Accurancy} = \frac{\#\text{ di esempi classificati correttamente}}{\#\text{ di esempi classificati}}$$

classificati, e la *F-measure*. Quest'ultima è una rielaborazione di altri 2 indici

$$\text{Precision} = \frac{\#\text{ di elementi classificati correttamente come positivi}}{\#\text{ di elementi realmente positivi}}$$

$$\text{Recall} = \frac{\#\text{di elementi classificati correttamente come positivi}}{\#\text{ di elementi realmente positivi}}$$

$$F_1 - \text{measure} = \frac{\text{2*Precision*Recall}}{\text{Precision+Recall}}$$

Si può notare abbastanza facilmente che le misure di Precision e Recall sono influenzate rispettivamente dal quantitativo di falsi positivi e falsi negativi. La F-measure semplicemente le sintetizza assieme e si preferisce alla classica accuratezza in quanto un classificatore costante positivo o costante negativo, dato lo scenario in cui le due classi hanno dimensioni molto sbilanciate, potrebbe paradossalmente risultare molto più accurato di un classificatore reale che invece distribuisce il proprio errore più uniformemente tra i falsi positivi e i falsi negativi.

4.2 Macchine a vettore di supporto

4.2.1 ν-support vector machine

Uno degli approcci considerati per risolvere il nostro problema fa uso delle Support Vector Machine nella loro formulazione ν-SVM. Si supponga di avere un problema di classificazione binaria classico, con esempi di entrambi le classi, e che gli esempi appartengano ad uno stesso spazio vettoriale, allora è possibile costruire un classificatore che cerchi di separare le 2 classi tramite un iperpiano. Nella forma lineare un classificatore di questo tipo utilizza come funzione

$$f(\mathbf{x}) = \text{sgn}(\boldsymbol{w} \cdot \mathbf{x} + b), \boldsymbol{w}, \mathbf{x} \in \mathbb{R}^{|V|}, b \in \mathbb{R}$$

dove $\mathbf{w}$ è un vettore descrivente il cosiddetto iperpiano separatore, mentre b descrive ulteriormente la posizione dell'iperpiano. Tra tutti gli iperpiani separatori, per motivi inerenti la teoria della complessità di Vapnick-Chervonenkis, si va a cercare l'iperpiano che massimizza la seguente quantità (detta margine):

$$\min_{\mathbf{x}}\{||\mathbf{x} - \boldsymbol{x_i}|| : \boldsymbol{x_i} \in \text{L}earning, (\mathbf{x} \cdot \boldsymbol{w} + b) = 0\}$$

che si può dimostrare essere unico. Trovare l'iperpiano ottimale si riconduce, dopo varie [11] riformulazioni matematiche, a risolvere il seguente problema di ottimizzazione vincolata:

$$\max_{\boldsymbol{\alpha} \in \mathbb{R}^m} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

$$\text{avendo } \alpha_i \geq 0, 1 \leq i \leq m \text{ e} \sum_{i=1}^{m} \alpha_i y_i = 0$$

dove y_i vale -1 se l'esempio $\mathbf{x}_i$ è negativo, $+1$ altrimenti, m è la taglia del campione usato per il training e le variabili α_i sono variabili di supporto introdotte per poter risolvere il problema.

Data questa formulazione si riscrive anche la formula di classificazione che diventa:

$$f(\mathbf{x}) = \text{sgn}(\sum_{i=1}^{m} y_i \alpha_i (\mathbf{x}_i \cdot \mathbf{x}) + b)$$

da cui si può intuire il perchè del nome Support Vector Machine: avendo riscritto la funzione di classificazione come espansione dei vettori di training, quelli tali per cui vale $\alpha_i > 0$ definiscono implicitamente l'iperpiano (ovvero lo supportano); proprio tali vettori vengono chiamati *vettori di supporto*.

Immediatamente ci si pone una domanda riguardo la possibilità di trovare questo iperpiano: nella formulazione data, l'iperpiano esiste solo se i dati sono linearmente separabili e pochi campioni potrebbero non essere in grado di supportare completamente l'iperpiano. Per non incorrere nel primo problema sarebbe necessario trovare una funzione non lineare per separare i dati. E' dimostrabile però che se essa esiste allora esiste uno spazio allargato[1] (chiamato feature space) in cui si possono trasportare i vettori in input tramite una iniezione $\Phi : \mathbb{R}^a \to \mathbb{R}^b$ nel quale è possibile trovare un iperpiano separatore. In pratica, la nostra funzione di classificazione diventerebbe:

$$f(\mathbf{x}) = \text{sgn}(\sum_{i=1}^{m} y_i \alpha_i (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x})) + b)$$

e si modifica relativamente il problema di ottimizzazione. Allargare molto la dimensione del feature space ha però lo svantaggio di rendere infattibile la computazione. Si può però notare che l'unica operazione che deve essere fatta sui vettori mappati in feature space, sia in apprendimento che in classificazione, è il prodotto interno. Pertanto,esistono una serie di spazi detti *kernellizzabili* per cui è possibile definire una funzione detta kernel $k(\mathbf{x}, \boldsymbol{y}) = \Phi(\mathbf{x}) \cdot \Phi(\boldsymbol{y})$, rendendo dunque implicito il feature space e dando spazio all'implementazione di classificatori SVM che operino efficientemente su spazi anche ad infinite dimensioni.

Il secondo problema di interesse è inerente la possibilità di definire l'iperpiano utilizzando i soli esempi forniti. Nella pratica, infatti, vista anche la necessità di scegliere manualmente il feature space in cui si preferisce eseguire la computazione, trovare un iperpiano separatore ottimale con i soli dati di training può non essere fattibile. A tal scopo si introduce la seguente variante del problema di ottimizzazione, che permette al classificatore di determinare un iperpiano tale per cui alcuni esempi positivi risultino nella metà negativa e viceversa:

[1] Lo spazio allargato però non è determinabile analiticamente: la scelta è dunque empirica.

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^m} \frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y_i y_j(\mathbf{x}_i\cdot\mathbf{x}_j)$$

$$\text{dove } 0\leq\alpha_i\leq\frac{1}{\nu m}, \sum_{i=1}^{m} y_i\alpha_i = 0, \sum_{i=1}^{m}\alpha_i > \nu$$

Il prezzo da pagare per aggiungere questa funzionalità è l'introduzione di un ulteriore parametro libero $\nu \in [0,1]$ (da cui ν-SVM) che rappresenta contemporaneamente il limite inferiore della frazione di vettori che vengono utilizzati come vettori di supporto e il limite superiore della frazione di esempi di apprendimento che possono essere classificati erroneamente. Questo tipo di classificatori viene detto a margine flessibile, in contrapposizione a quelli descritti precedentemente detti a margine fisso.

4.2.2 One-class SVM

In un recente lavoro ([27]) ci si pone il problema di costruire una macchina in grado di stimare i confini del sottospazio degli esempi positivi all'interno della collezione vera e propria, senza avere a disposizione esempi negativi o non etichettati. Praticamente, il problema posto è quello di trovare l'ipersfera di raggio minimo contenente tutti e soltanto gli esempi positivi, in un feature space altamente non lineare. Solitamente si utilizza lo spazio con kernel RBF, dove

$$k(\mathbf{x},\boldsymbol{y}) = -\gamma e^{||\mathbf{x}-\boldsymbol{y}||^2}$$

dove $\gamma \in (0,1)$ è un parametro del kernel.

Anche in questo caso è possibile scrivere il tutto come un problema di ottimizzazione: sia R il raggio dell'ipersfera in feature space, allora si cerca

$$\min_{\boldsymbol{\alpha}} \sum_{i,j=1}^{m}\alpha_i\alpha_j k(\boldsymbol{x_i},\boldsymbol{x_j}) - \sum_{i=1}^{m}\alpha_i k(\boldsymbol{x_i},\boldsymbol{x_i})$$

$$\text{avendo } 0\leq\alpha_i\leq\frac{1}{\nu m}, \sum_{i=1}^{m}\alpha_i = 1$$

per poter utilizzare la funzione di classificazione

$$f(\mathbf{x}) = \operatorname{sgn}\left(R^2 - \sum_{i,j=1}^{m}\alpha_i\alpha_j k(\boldsymbol{x_i},\boldsymbol{x_j}) + 2\sum_{i=1}^{m}\alpha_i k(\boldsymbol{x_i},\mathbf{x}) - k(\mathbf{x},\mathbf{x})\right)$$

Per kernel, come quello RBF, della forma $k(\mathbf{x},\boldsymbol{y}) = h(\mathbf{x}-\boldsymbol{y})$, il termine lineare del problema di ottimizzazione diventa costante, riducendo per tanto l'ottimizzazione nuovamente a quello delle SVM classiche. Può essere inoltre

dimostrato che la funzione di decisione diventa ugualmente simile a quella di una SVM normale. Questo tipo di classificatore si chiama One-class SVM e pur essendo un oggetto particolarmente interessante dal punto di vista teorico i risultati pratici sono utilizzabili principalmente per cercare esempi mal classificati secondo un altro classificatore: utilizzare direttamente una OSVM per l'apprendimento porta a scarsissimi risultati data anche la sua enorme suscessibilità ai parametri γ e ν, totalmente empirici.

Si noti comunque, ai fini delle discussioni successive, che se esiste uno spazio espanso in cui tutti gli elementi positivi possono essere contenuti in un ipersfera di raggio finito, la distanza dal centro dell'ipersfera di un esempio negativo può essere una ragionevole misura di negatività dell'esempio stesso: un esempio quasi a contatto con la superficie dell'ipersfera può essere denominato come blandamente negativo mentre un esempio molto lontano è sicuramente più negativo di quello preso in considerazione prima.

4.3 Approcci considerati

Per il progetto sono state messe a confronto due approcci completamente diversi per la classificazione con esempi positivi e non etichettati, il primo è un modello di tipo discriminativo, ovvero che ignora la struttura interna dei dati utilizzati per lavorare e si chiama Support Vector Mapping Convergence (successore del più conosciuto PEBL - Positive Example Based Learning). Il secondo è stato descritto ultimamente e non è ancora stato vagliato ampiamente dalla comunità scientifica ed è un modello di tipo generativo (ovvero che tenta invece di tararsi sulla struttura interna dei dati forniti e utilizzarli integralmente) che può essere riassunto con il nome di Biased PrTFIDF (Biased Probabilistic TermFrequency-InverseDocumentFrequency).

Sono presenti in letteratura anche altri approcci (quali Co-training [5]) che però sono stati scartati in partenza o perché già superati da SVMC o B-PrTFIDF o perché computazionalmente troppo onerosi per essere poi inseriti in un crawler.

4.3.1 SVMC

L'approccio di Support Vector Mapping Convergence è stato descritto per la prima volta in [31] e poi rifinito in [30] e si basa su un procedimento in due fasi: la primo è il Mapping, tramite cui un weak learner genera una prima e grossolana approssimazione degli esempi negativi all'interno di U, seguito dalla fase Convergence, in cui un procedimento iterativo raffina la divisione di U in positivi e negativi.

Mapping. Si supponga al nostro scopo di definire una funzione $h{:}U \rightarrow \mathbb{R}$, positiva per gli elementi di C_+ e negativa o al più zero per gli altri, tale per cui $|h(\mathbf{x})| > |h(\boldsymbol{y})|$ se e soltanto se $\mathbf{x}$ è definibile come maggiormente positivo (o negativo) di $\boldsymbol{y}$; gli elementi corrispondenti ai massimi e ai

minimi di h possono essere denominati *fortemente positivi* e *fortemente negativi*, rispettivamente. Dare una definizione operativa di questa h è difficile e forzatamente empirica, ma è possibile fare il seguente ragionamento: se lo scopo è quello di ottenere soli e soltanto i fortemente negativi allora una definizione estremamente generosa di positivo, tale da includere anche molti degli elementi che potrebbero essere debolmente negativi, è complementare alla nozione che cerchiamo. In altre parole, un classificatore che sicuramente non genera falsi negativi su elementi di P e che etichetta come positivi tutti gli elementi non fortemente negativi, è in grado di fornire un primo, approssimativo, nucleo degli elementi negativi contenuti in U.

Gli autori descrivono due metodi differenti per creare questo classificatore. Nella forma più semplice (1-DNF) si categorizzano innanzitutto i termini contenuti nelle collezioni documentale P e U come discriminanti dei positivi o meno: t è discriminante positivo se $\frac{f_t^P}{f_t^U} > \theta$, ovvero la frequenza[2] con cui compare all'interno di P è almeno θ volte la frequenza con cui compare in U, dove θ è una soglia aggiustabile. In altre parole, se un termine compare nel 90% dei documenti positivi e nel 10% di quelli non etichettati, allora è molto probabile che la sua presenza contribuisca ad una maggior attinenza del documento con la classe dei positivi. A questo punto il classificatore

$$f(\mathbf{x}) = \text{sgn}(\sum_{w \in \text{DISCR}} \mathbf{x}_w)$$

è anche la nostra funzione h e descrive i fortemente negativi come quei documenti non contenenti alcun discriminante positivo. Per completare la descrizione, si sceglie θ in modo che il classificatore risultante applicato agli elementi di P dia quasi 100% di recall (ovvero non generi falsi negativi), utilizzando una ricerca lineare. Questo è coerente con i nostri desideri: la fase di Mapping deve generare una approssimazione grossolana e molto restrittiva dei negativi, tale per cui essi possano essere certamente un punto di partenza valido nel caso si voglia ingrandire monotonicamente l'insieme dei negativi.

Successivamente gli autori propongono di utilizzare una One class SVM, addestrata sui soli elementi di P ed applicata a U, tarando i parametri della One class SVM in modo da ottenere 100% recall sugli elementi di P (ovvero azzerando i falsi negativi, senza preoccuparsi dei falsi positivi) e prendendo come fortemente negativi quelli che la OSVM etichetta come negativi. In altre parole, si forza la OSVM, a comporre una sfera coerente con gli esempi dati a discapito della minimalità del raggio. Per fare questo è sufficiente utilizzare dei valori di ν molto piccoli, dell'ordine del 10^{-2}, forzando il classificatore a non accettare errori sugi esempi forniti se non in

[2]La frequenza di un termine all'intero di una collezione documentale è supposta essere il conteggio dei documenti all'intero di cui compare. Purtroppo il lavoro originale non specifica come intepretare questa locuzione.

casi estremi. Anche qui la ricerca di ν nell'intervallo desiderato viene fatta linearmente, selezionando poi la OSVM che ha prodotto recall più alto.

Convergence. La seconda fase dell'algoritmo è un procedimento iterativo. Si inizializzi N con l'insieme dei fortemente negativi generato al passo di Mapping. Al passo i dell'iterazione si addestra una ν-SVM su P e N e si classificano gli elementi di $U \backslash N$ in negativi e positivi. I soli elementi negativi risultanti vengono aggiunti a N mentre i positivi non vengono ulteriormente toccati. Il procedimento si arresta al passo in cui la SVM non classifica più alcun elemento di $U \backslash N$ come negativo.

L'intero algoritmo SVMC è dunque una sequenza di azioni atta ad incrementare monotonicamente l'insieme dei negativi contenuti in U utilizzando la sola conoscenza dei positivi indotta dalle label e i dati negativi inizializzati dal Mapper che vengono pian piano costruiti. Al termine del processo di apprendimento, l'ultima SVM creata dalla fase di Convergence è utilizzabile per la classificazione di documenti.

SVMC è un algoritmo lento (solitamente richiede una ventina di addestramenti) ma ha solide basi teoriche e si basa su una tecnologia già ampiamente affermata, quella delle SVM; ha l'ovvio svantaggio che se il classificatore di Mapping classifica erroneamente come negativi degli elementi positivi l'intero processo potrebbe convergere ad un classificatore costante positivo o negativo. Un altro svantaggio è dovuto all'utilizzo di un sottoclasfficatore binario che ha dei parametri liberi ma soprattutto richiede un bilanciamento intorno al 50% dei negativi e dei positivi usati in apprendimento per poter trovare un buon iperpiano. Le implementazioni di SVMC devono dunque fare fronte all'esiguità dei negativi iniziali e man mano dei positivi poi, alterando l'algoritmo in modo che a ogni passo di convergence estragga un campione uniforme da P ed N della grandezza del più piccolo dei 2 insiemi e faccia addestramento su di questo. Si può scoprire facilmente con qualche prova che il campionamento purtroppo condiziona in modo pesante la rapidità e la probabilità di convergenza di SVMC.

In quanto si verificherà più avanti che le prestazioni del DNF e dell'OSVM Mapper sono assai scarse, sperimentalmente e al di fuori delle specifiche dell'articolo originale è stato introdotto un Cosine Mapper. Siccome in un qualsiasi spazio vettoriale una buona misura di distanza tra i documenti, usata anche in Information Retrieval, è il coseno dell'angolo tra due vettori, un Mapper ragionevole può essere costruito creando il vettore medio m tra tutti gli elementi di P, calcolando il coseno minimo raggiunto tra m e un elemento di P, per poi definire fortemente negativo un documento che abbia coseno più piccolo di questa soglia. In più siccome $\cos(\mathbf{x}, \boldsymbol{y}) = \frac{\mathbf{x} \cdot \boldsymbol{y}}{\sqrt{(\mathbf{x} \cdot \mathbf{x})(\boldsymbol{y} \cdot \boldsymbol{y})}}$ è possibile applicare alla misura del coseno un qualsiasi kernel trick per lavorare in uno spazio allargato implicitamente. Per poter calcolare la distanza dal vettore medio $m = \frac{1}{|P|} \sum_{\mathbf{x} \in P} \Phi(\mathbf{x})$ nel numeratore della formula del coseno si

sfrutta le proprietà di linearità del prodotto interno tali per cui:

$$\Phi(\boldsymbol{y}) \cdot m = \Phi(\boldsymbol{y}) \cdot \frac{1}{|P|} \sum_{\mathbf{x} \in P} \Phi(\mathbf{x}) = \frac{1}{|P|} \sum_{\mathbf{x} \in P} k(\mathbf{x}, \boldsymbol{y})$$

4.3.2 B-PrTFIDF

Un approccio completamente diverso utilizza un metodo di tipo generativo per la creazione di un modello di P e di U che sia in grado di distinguere positivi e negativi ed è descritto in [32]. In particolare, l'assunzione iniziale, confermata dalle indagini contenute in [13], consiste nel considerare che un elemento di training di C_+ ha proabilità p di essere lasciato non etichettato all'interno di U e probabilità $1 - p$ di essere invece all'interno di P. Si consideri infatti questo classificatore ipotetico, semplice ma molto sensato:

$$f(\mathbf{x}) = \text{sgn}(\Pr[\mathbf{x} \in C_+] - \Pr[\mathbf{x} \in C_-])$$

Alla luce della nostra assunzione si può scrivere:

$$\begin{aligned} \Pr[\mathbf{x} \in P] &= (1-p) \Pr[\mathbf{x} \in C_+] \\ \Pr[\mathbf{x} \in U] &= p \Pr[\mathbf{x} \in C_+] + \Pr[\mathbf{x} \in C_-] \end{aligned}$$

da cui:

$$\begin{aligned} & \Pr[\mathbf{x} \in C_+] - \Pr[\mathbf{x} \in C_-] = \\ = \quad & \frac{1}{1-p} \Pr[\mathbf{x} \in P] - \Pr[\mathbf{x} \in U] + p \Pr[\mathbf{x} \in C_+] \\ = \quad & \frac{1+p}{1-p} \Pr[\mathbf{x} \in P] - \Pr[\mathbf{x} \in U] \end{aligned}$$

Per cui, il nuovo classificatore può essere descritto, ponendo $b = \frac{1+p}{1-p}$:

$$f(\mathbf{x}) = \text{sgn}(b \Pr[\mathbf{x} \in P] - \Pr[\mathbf{x} \in U])$$

riducendo il problema a stimare b e le due probabilità. Si noti inoltre che è facilmente possibile ottenere anche una stima della probabilià di appartenenza alla classe dei positivi, tale per cui se questa supera 0.5 allora il documento viene classificato positivamente:

$$\begin{aligned} & \Pr[x \in C_-] = 1 - \Pr[x \in C_+] \\ \iff \quad & \Pr[x \in C_+] - \Pr[x \in C_-] = 2 \Pr[x \in C_+] - 1 \\ \iff \quad & \frac{1}{2}(1 + b \Pr[x \in P] - \Pr[x \in U]) = \Pr[x \in C_+] \end{aligned}$$

Avendo a disposizione la decomposizione di documenti in parole, per un generico insieme di documenti S, è ragionevole porre, dando la generalistica semplificazione che l'apparizione dei termini sia stocasticamente indipendente, per il teorema delle probabilità condizionate:

$$\begin{aligned}
\Pr[\mathbf{x} \in S] &= \sum_{w \in V} \Pr[\mathbf{x} \in S | w \in \mathbf{x}] \Pr[w \in \mathbf{x}] \\
\Pr[w \in \mathbf{x}] &= \frac{\mathbf{x}_w}{\sum_i \mathbf{x}_i} \\
\Pr[\mathbf{x} \in S | w \in \mathbf{x}] &= \frac{\sum_{\boldsymbol{y} \in S} \Pr[w \in \boldsymbol{y}]}{\sum_{\boldsymbol{z} \in Z} \Pr[w \in \boldsymbol{z}]}
\end{aligned}$$

dove l'ultima equazione, nel nostro caso può essere ridotta a:

$$\Pr[\mathbf{x} \in S | w \in \mathbf{x}] = \frac{1}{1 + \frac{\sum_{\boldsymbol{y} \in \bar{S}} \Pr[w \in \boldsymbol{y}]}{\sum_{\boldsymbol{z} \in S} \Pr[w \in \boldsymbol{z}]}}$$

in quanto Z è diviso in due soli sottoinsiemi.

Per completare lo schema è necessario introdurre un modo per avere una stima di p; il modo operativo introdotto è quello di utilizzare un validaion set esterno di documenti per eseguire delle valutazioni dopo aver costruito il classificatore sul training set. E' importante che il rapporto $\frac{|P|}{|U|}$ e quello $\frac{|P_{val}|}{|U_{val}|}$ siano pressochè identici e che provengano da una sorgente con lo stesso valore di p. Il metro di valutazione teoricamente migliore sarebbe la F-measure descritta in precedenza, ma non essendo possibile calcolare la Precision si preferisce una sua rielaborazione. Tenendo conto che, probabilisticamente, la Precision è $\Pr[\mathbf{x} \in C_+ | f(\mathbf{x}) = 1]$ e il Recall è $\Pr[f(\mathbf{x}) = 1 | \mathbf{x} \in C_+]$, la misura $\frac{rp}{\Pr[\mathbf{x} \in C_+]}$ è alta solo quando sia Precision che Recall sono alte. Si noti ora che:

$$\begin{aligned}
&\Pr[f(\mathbf{x}) = 1 | \mathbf{x} \in C_+] \Pr[\mathbf{x} \in C_+] = \Pr[\mathbf{x} \in C_+ | f(\mathbf{x}) = 1] \Pr[f(\mathbf{x}) = 1] \Leftrightarrow \\
&\frac{\Pr[f(\mathbf{x}) = 1 | \mathbf{x} \in C_+]}{\Pr[f(\mathbf{x}) = 1]} = \frac{\Pr[\mathbf{x} \in C_+ | f(\mathbf{x}) = 1]}{\Pr[\mathbf{x} \in C_+]} \Leftrightarrow \\
&\frac{r}{\Pr[f(\mathbf{x}) = 1]} = \frac{p}{\Pr[\mathbf{x} \in C_+]} \Leftrightarrow \\
&\frac{rp}{\Pr[\mathbf{x} \in C_+]} = \frac{r^2}{\Pr[f(\mathbf{x}) = 1]}
\end{aligned}$$

Essendo il Recall stimabile sul validation set come $\Pr[f(\mathbf{x}) = 1 | \mathbf{x} \in P_{val}]$ e similmente $\Pr[f(\mathbf{x}) = 1]$, questa nuova misura conserva le stesse proprie-

tà della F-measure e preidilige comunque un classificatore più conservativo nell'elargire positivi a parità di falsi negativi.

Di conseguenza, usando un validation set e valutando la *Biased F-measure* appena descritta, si può costruire una stima di p costruendo le probabilità necessarie al classificatore su P ed U. Questo algoritmo ha un evidente vantaggio: richiede al più 2 passate lineari sui documenti e una ricerca lineare computaizonalmente poco impegnativa per produrre un classificatore. Inoltre non richiede di ritrovare all'interno di U dei negativi.

4.4 Architettura software

Al fine di sperimentare i due algoritmi sopra descritti è stata scritto un pacchetto software in Java 1.5 utilizzando due package ausiliari: MG4J e LibSVM [10] Il primo è sviluppato all'interno dell'università ed è un set di classi per la creazione e manipolazione di indici inversi e di collezioni documentali. Una collezione documentale è un semplice insieme di file o pagine web accettate da MG4J per la creazione di un indice, ovvero di una struttura dati residente su disco che, previa numerazione dei documenti della collezione, crea delle liste, una per ogni termine contenuto all'interno della collezione, contenenti coppie della forma $\langle d, f \rangle$ dove d è il numero del documento in cui compare il termine in questione e f è la frequenza del termine all'interno di d. A loro volta, le liste sono numerate e su disco vengono depositate separatemente le liste e una mappa che associa ad ogni numero di termine la parola stessa. L'utilizzo di MG4J è motivato dalla necessità di dover fare più volte una scansione dei termini nei documenti sequenzialmente sui termini. Utilizzando la collezione è invece possibile accedere direttamente ai documenti per ottenere le liste di termini che contengono (il cosiddetto indice diretto).

LibSVM è invece un pacchetto disponibile anche per Java che implementa le ν-SVM e le OSVM su formati di dati vettoriali ponendo a disposizione la canoninca scelta di kernel (lineare, polinomiale di grado arbitrario, RBF, logistico). Comprende sia la classica architettura per training e testing che alcuni extra quali ad esempio il k-fold cross validation, una tecnica che suddivide il training set in k sottoinsiemi egualmente dimensioniati e, in assenza di un test set, produce una stima, in termini di accurancy, della bontà del classificatore semplicemente addestrandolo su $k-1$ insiemi e testandolo sul rimanente, facendo poi la media di tutti i k addestramenti.

Classi di supporto:

- *StopWordTermProcessor* è una classe che viene utilizzata insieme con MG4J per permettere di eliminare dall'indicizzazione alcuni termini comunemente riconosciuti come Stopword, quali i pronomi, le preposizioni, i verbi essere, etc.. che non sono di alcun aiuto ai processi

di classificazione in quanto presenti universalmente nei documenti e in ampie quantità. L'elenco dei file viene fornito tramite la system property *StopWordTermProcessor.words*

- *SVMUtils* si occupa di tradurre le liste invertite e dirette nel formato di vettori sparsi necessari a LibSVM per lavorare.
- *Kernel, RBFKernel, PolynomialKernel* sono rispettivamente un'interfaccia e due implementazioni che contengono un Kernel di tipo RBF e uno polinomiale, vengono utilizzati dal Cosine Mapper.

SVMC:

- *Mapper* è una classe astratta da cui derivano tutte le implementazioni che eseguono la fase di Mapping. Ogni Mapper riceve in input indici inversi e collezioni di P ed U, eventuali parametri, e classifica gli elementi di U come fortemente negativi o meno. Le label finali vengono poi depositate su un file pronti per la fase di convergence.
- *DNFMapper* è l'implementazione del Mapper 1-DNF: non riceve altri parametri in input e dopo aver calcolato le frequenze determina la miglior soglia riclassificando gli elementi di P e contorllando il recall, prediligendo la θ più bassa a parità di recall.
- *OSVMMapper* è l'implementazione del Mapper con One-class SVM: ricevuto in input il γ del kernel RBF, addestra una OSVM sugli elementi di P con una ricerca lineare di $\nu \in (0.005, 0.1)$ prediligendo il classificatore con massimo recall quando applicato a P stesso.
- *CosineMapper* è l'implementazione del Mapper sperimentale basato su coseno: una volta avuto il parametro del kernel RBF che utilizza, esegue il calcolo del vettore medio
- *MappingConvergence* contiene il codice della fase di Convergence: presi in input collezioni, indici inversi e risultati della fase di Mapping, itera la fase di addestramento delle SVM e ampliamento dei negativi fino ad un numero massimo (ritoccabile) di 50 iterazioni, per poi produrre un modello di SVM da utilizzare in test.
- *SVMCClassify* classifica due collezioni (non indicizzate) rispettivamente di elementi positivi e elementi negativi per testare l'efficacia del metodo (la collezione di negativi viene quindi inserita volontariamente e manualmente per testare non solo il recall ma anche la precisione), utilizzando il modello generato di MappingConvergence.

B-PrTFIDF:

- PTITrain è la classe che dati indici e collezione dei documenti di P, U, P_{val}, U_{val} si occupa di costruire e scrivere su disco $\Pr[\mathbf{x} \in P | w \in \mathbf{x}]$ e $\Pr[\mathbf{x} \in U | w \in \mathbf{x}]$ in un oggetto serializzato da Java, insieme con la stima ottenuta per b. I valori di $\Pr[w \in \mathbf{x}]$ per $\mathbf{x} \in P \cup U$ non vengono memorizzati in quanto non sono necessari poi in fase di classificazione.

- PTIClassify esattamente come *SMVCClassify* classifica due collezioni utilizzando il modello creato da *PTITrain*, producendo i valori di F-measure, Accurancy, Precision e Recall.

4.5 Esperimenti

Per valutare la bontà dei due metodi si sono scelti 2 dataset su cui metterli a confronto in situazioni differenti. Il primo è 20NG [1], un dataset che contiene un campionamento di alcuni newsgroup pubblici su internet e si tratta di documenti in puro testo senza struttura, molto simili alle email. Il secondo è WebKB [29], un insieme di pagine web di 3 diverse università americane, presuddivisi in cateogrie da dei classificatori umani, contenenti quindi poco rumore ma testo di tipo strutturato.

In ogni esperimento sono stati messi a confronto la Accuratezza, la F-measure e il numero di falsi positivi e negativi dei due metodi. Ogni esperimento con SVMC è stato riprodotto con 3 diversi Mapper mentre gli esperimenti di B-PrTFIDF sono stati effettuati con e senza eliminazione delle stopword, tutti con un valore di p prefissato.

Nel dataset 20NG sono stati usati come positivi gli elementi del newsgroup *rec.sport.hockey*, e l'universo è stato il dataset stesso. Il training è stato effettuato su 300 elementi positivi e 3000 elementi non etichettati, di cui il 60% circa erano positivi (dunque $p = 0.6$). Il set di validazione per B-PrTFIDF è di 100 positivi più 1280 non etichettati. Il set di test è invece composto da 590 positivi e 1780 negativi.

Differentemente, in WebKB le pagine che sono classificate manualmente come appartenente alla classe *faculty* sono i positivi. Il set di training è composto da 350 elementi positivi e 9650 non etichettati, con probabilità $p = 0.3$. Il set di validazione è di 140 positivi contro 3860 non etichettati, mentre quello di test è 300 positivi contro 5800 negativi.

I risultati sono espressi nella tabella sottostante, dove la dicitura *+SW* indica l'eliminazione delle stopword durante il processo di indicizzazione

Metodo	20NG	WebKB	20NG+SW	WebKB+SW
SV 1DNF	$0.31(\theta = 1.0)$	$0.22(\theta = 3.0)$	$0.31(\theta = 1.4)$	$0.31(\theta = 3.1)$
SV OSVM	$0.32(\nu = 0.155)$	$0.24(\nu = 0.018)$	$0.21(\nu = 0.009)$	$0.33(\nu = 0.01)$
SV Cosine	0.31	0.26	0.21	0.5
B-PrTFIDF	0.85	0.52	0.87	0.66

Salta subito all'occhio che le performance di B-PrTFIDF, indipendentemente dal metodo di Mapping, superano di gran lunga quelle del metodo SVMC.

Capitolo 5

Un nuovo crawler

Il metodo dei grafi di contesto pone l'accento sulla problematica di stimare la distanza dei nodi da quelli rilevanti e assolve tale compito in modo strettamente contenutistico. Il modello markoviano invece utilizza un metodo strutturale con lo svantaggio di dipendere fortemente dall'apprendimento delle aree semantiche (ad esempio, il numero di cluster): l'algoritmo tenta di determinare la distanza usando una generica appartenenza semantica e la distanza dell'osservazione precedente.

Gli esperimenti con il metodo basato su grafi dimostrano la fattibilità di utilizzare dei semplici classificatori testuali per stimare la classe di distanza di ogni pagina; quelli con il metodo basato su modelli probabilistici invece dimostrano che la struttura del grafo inter-contestuale può essere utilizzata efficacemente per determinare le priorità di visita.

In questo capitolo si fornisce dunque un nuovo modello, applicabile al seguente scenario,che fonde assieme le caratteristiche di quello markoviano e di quello con grafi di contesto, sfruttando anche i risultati del capitolo 4.

Si supponga dunque che l'utente possa istruire il crawler sui propri interessi soltanto all'inizio del processo di crawl e non possa più essere interepellato. Lo scopo è produrre un algoritmo di visita che sfrutti le capacità di entrambi i modelli sopra descritti, ovvero si basi sia sulla struttura che sul contenuto, con apprendimento continuato, riaggiornando quindi il modello ogni qual volta sia possibile introdurre nuovi elementi. Il crawler dev'essere progettato in modo che sia in grado di aggiornare il proprio modello e di dare dei valori di importanza alle nuove pagine efficientemente sia in termini di spazio che di tempo, senza poter utilizzare elementi esterni al processo di crawl stesso.

Il risultato è il modello descritto sotto che viene poi sperimentato su un'architettura di visita simulata scritta in linguaggio Java sulla collezione descritta nel capitolo 2.

5.1 Modello markoviano per i grafi di contesto

Per giungere al nuovo modello è utile cercare di integrare il modello dei grafi di contesto all'interno di quello markoviano. Nel contesto markoviano il crawler si comporta come un navigatore probabilistico sul grafo degli stati osservabile solamente tramite le pagine web che visita. L'irreplicabilità intrinseca dell'osservazione di una pagina web (intesa come vettore nello spazio dei termini) introduce, al fine di poter generare delle stime per le probabilità di emissione delle osservazioni, la necessità di aggregare gli elementi dello spazio dei termini in un insieme di cardinalità molto più piccola. La scelta originale è quella di raggruppare in K aree semantiche una versione ridotta delle pagine web. Quindi il modello markoviano nascosto associa le aree semantiche degli antenati del nodo nella visita e gli stati precedenti ad uno stato successivo tramite l'algoritmo di Viterbi.

In senso lato, è possibile considerare il modello dei grafi di contesto come una istanza di un modello markoviano nascosto degenere. Infatti, se le osservazioni fossero le d classi di distanza emesse dal classificatore bayesiano, allora l'HMM dovrebbe associare direttamente e sempre lo stato all'ultima osservazione ottenuta. Questo esempio può anche essere letto in maniera differente. Preso un classificatore bayesiano a d classi e un modello HMM a d stati addestrati costantemente durante il crawl , se fosse possibile giungere ad una situazione in cui il modello markoviano associa direttamente stato ed osservazione e lo stato è prevalentemente uguale alla distanza, allora sarebbe sufficiente utilizzare il testo delle pagine per effettuare focused crawling. Ci preoccuperemo di dare una risposta alla congettura di cui sopra nel capitolo successivo.

Il nuovo modello proposto è ispirato alla discussione di cui sopra e viene definito *modello markoviano per i grafi di contesto statico*. Sia d la distanza massima coperta, si crea un modello markoviano con d stati e d osservazioni. Un classificatore a d classi (di cui una sogliata, come descritta nel capitolo 2) produce, alla visita di una nuova pagina, un'osservazione della classe di distanza basata sul contenuto. Il modello markoviano "corregge" la stima di distanza integrando la conoscenza strutturale e genera una nuova stima di distanza. La priorità dei figli delle pagine visitate non ancora in frontiera viene quindi assegnata uguale a quella della pagina genitore; più formalmente, il genitore di ogni nodo v è quel nodo u tale che $(u, v) \in E$ tale per cui $u \in F$ e $v \notin F$. Siccome ogni volta che questo tipo di archi viene percorso il crawler inserisce v in F allora la definizione è ben posta. Per gli elementi del seme di visita, il genitore è il nodo stesso.

Ogni qualvolta il crawler incontra una nuova pagina ritenuta di interesse esegue una procedura di aggiornamento dei modelli, eseguendo una fase di back-crawling sulle pagine antenate che ha visitato e riassegnando loro nuovi valori di stato. Contemporaneamente, i gruppi di pagine per grado di distanza vengono utilizzati per riaddestrare il classificatore bayesiano. A

fronte del riaddestramento bayesiano vengono ricalcolate le probabilità della matrice di transizione degli stati e di emissione delle osservazioni.

Ponendosi in un contesto di apprendimento *online* e volendo sfrutture al massimo la conscenza possibile l'ultimo arbitro sul grado di interesse della pagina è un classificatore di tipo B-PrTFIDF addestrato sul campione iniziale dell'utente. L'utilizzo di questo classificatore aiuta particolarmente nelle prime fasi del crawl: quando il modello markoviano e quello bayesiano non sono ancora stati addestrati almeno una volta, la visita denegera in una Best-First search utilizzando come priorità la probabilità di appartenenza alla classe dei positivi data dal modello. Ulteriormente, durante la visita, secondo un parametro impostabile $0 \leq \alpha \leq 1$, la priorità di una pagina viene ereditata dalla pagina genitore e vale il suo *punteggio*:

$$\sigma = \alpha \frac{(d - \delta)}{d} + (1 - \alpha)\gamma$$

dove δ è la stima di distanza data dal modello markoviano nascosto e γ è il grado di verosimiglianza proposto dal classificatore B-PrTFIDF. Il crawler si limita a estrarre ogni volta dalla coda la pagina con massima priorità per determinare l'elemento di aumento della frontiera.

Schematicamente, considerando il grafo G , il seme di visita che costituisce inizialmente la frontiera F, il modello markoviano nascosto $HMM(S, O, \pi, A, B)$ con d stati e d osservazioni, il classificatore B-PrTFIDF che determina $\Pr[\boldsymbol{x} \in C_+]$ per ogni documento x ed il classificatore Bayesiano sogliato a d classi che determina $1 < class(\boldsymbol{x}) \leq d$, si ottiene il seguente algoritmo di vista, in cui inizialmente gli elementi del seme hanno priorità 0.

1. Preleva $u = \arg\max_{x \in F} p_x$
2. Se $\Pr[u \in C_+] > 0.5$ allora esegui il riaddestramento del modello:
   (a) Esegui una visita in ampiezza su G^T con u come unico seme, limitata ad una distanza da u pari a d, senza considerare nodi non toccati dalla visita originaria.
   (b) Assegna agli elementi di questa visita un nuovo stato, e poni $\delta_u = 0$.
   (c) Aggiorna le stime di probabilità dei termini e delle classi del modello bayesiano con i risultati del back-crawl e riclassifica tutti quei documenti, rigenerando le osservazioni.
   (d) Aggiorna il modello markoviano, ricalcolando le matrici di transizione e emissione sulle nuove osservazioni.
3. Altrimenti, sia *class(parent(parent(⋯ parent(u)))), . . . , class(parent(u)), class(u)* la sequenza di osservazioni riguardanti il cammino di visita sul grafo degli stati nascosti dell'HMM. Utilizza Viterbi per calcolare esclusivamente per u la stima di distanza più verosimile, denotata δ_u.

4. Calcola il punteggio $\sigma_u = \alpha\frac{d-\delta_u}{d} + (1-\alpha)\Pr[u \in C_+]$.
5. Per ogni v tale che $(u,v) \in E \wedge v \notin F$, assegna *parent(v)* $= u$ e $p_v = \sigma_u$, dopodichè $F = F \cup v$.
6. Torna al passo 1.

5.2 Riassegnamento dei genitori

La scelta di assegnare come genitore di ogni nodo (e conseguente priorità) il nodo che lo fa inserire in frontiera, ovvero il primo nodo visitato da cui è raggiungibile, ha l'ovvio vantaggio di non porre alcun costo computazionale aggiuntivo ma si tratta di una scelta puramente arbitraria che può portare ad una visita poco oculata. Per l'algoritmo statico vale infatti la seguente

Proprietà 2 *Sia $v \in F$ e sia $I(v) = \{u_1, \ldots, u_n\}$ l'insieme dei nodi da cui v è raggiungibile. Vale quindi la seguente:*

$$p_v \leq \max_{x \in I(v)} s_x.$$

Nel caso $I(v) = \varnothing$, vale invece che $p_v = 0$.

Se v viene visitato e $I(v) = \varnothing$ allora v è forzatamente nel seme, quindi per definizione $p_v = 0$. Per gli altri nodi l'asserto deriva dal fatto che il genitore viene scelto in base alla priorità dello stesso all'interno della coda, che non ha alcun legame con il punteggio del nodo stesso (infatti se $p_{u_1} > p_{u_2}$ e $s_{u_1} < s_{u_2}$ la disuguaglianza vale in modo stretto).

Quindi man mano che i nodi vengono estratti dalla coda e ne si calcola il punteggio questi potrebbero fornire una maggiore priorità per i discendenti del nodo: riassegnando la priorità e i genitori in qualche modo si renderebbe la visita più efficace. Per poter operare delle scelte in tale senso è necessario considerare che il riassegnamento dei genitori varia la stringa di osservazioni che vengono utilizzate per il calcolo del punteggio delle pagine scaricate, che dipende dallo stato generato dall'HMM e dalla stima di rilevanza ($\Pr[x \in C_+]$). Decidiamo quindi di voler riassegnare i genitori in modo da ottenere il minimo assegnamento di stato finale per ogni sequenza di nodi.

Più formalmente, denotiamo con $o_{(v)}$ l'osservazione associata alla pagina v e estendiamo il concetto di *forward variable* δ usato dall'algoritmo di Viterbi, attraverso le seguenti definizioni:

Definizione 7 *Sia E l'insieme degli archi di un grafo G, e siano E^n, $n \geq 2$, l'insieme contente le n-uple rappresentanti i percorsi di lunghezza n sul grafo G; si definisca*

$$\chi_m(v_1, \ldots, v_m) = \begin{cases} 1 & \text{se } (v_1, \ldots, v_m) \in E^m \\ -\infty & \text{altrimenti} \end{cases}$$

Definizione 8 *Si supponga di operare su un grafo aciclico e diretto e siano* $v \in V$ *e* $s \in S$ *e siano* $v_1, \ldots, v_{t-1}$ *variabili a valori in* V *ed* $s_1, \ldots, s_{t-1}$ *variabili a valore in* S*, con* $t \geq 1$*. Si denoti con* $o_{(u)}$ *l'osservazione associata al nodo generico nodo* u*. Si definisce* stato ammissibile a t passi *(t-SA)i date le pagine* $v_1, \ldots, v_{t-1}$ *come la seguente generalizzazione delle* forward variable*:*

$$\phi_t(v, s) = \max_{s_1,\ldots,s_{t-1}} \Pr\left[s_1, \ldots, s_{t-1}, s; o_{(v_1)}, \ldots, o_{(v_{t-1})}, o_{(v)}\right] \chi_t(v_1, \ldots, v_{t-1}, v)$$

Si definisce inoltre minimo stato ammissibile a t passi *(t-MSA) rispetto ad un generico HMM il valore*

$$\theta_t(v) = \min_{v_1,\ldots,v_{t-1}} \arg\max_i \phi_t(v_t, i)$$

Calcolare il t-MSA di un nodo v secondo la definzione richiede, secondo gli studi volti finora, di calcolare tutte le possibili stringhe di osservazioni legate ai percorsi terminanti sul nodo v. Ci limitiamo quindi a calcolare la sua versione ridotta all'ultimo passo:

Definizione 9 *Con la stessa notazione della definizione precedente sia*

$$v^* = \hat{\theta}_t(v) = \min_{v_{t-1}} \arg\max_{s_t} \max_{s_1,\ldots,s_{t-1}} \Pr\left[s_1, \ldots, s_t; o_{(v_1^*)}, o_{(v_2^*)}, \ldots, o_{(v_{t-1})}, o_{(v)}\right]$$

dove i primi $t-2$ *nodi sono assegnati secondo la definizione di* $\hat{\theta}$ *per i vari* $v_{t-1} \in I(v)$*.*

Tuttavia per effettuare questo tipo di computazione correttamente sarebbe necessario visitare il grafo per ampiezza, dopo aver scelto di spezzare i cicli in qualche modo, obbiettivo differente da quello del focused crawler. Cerchiamo quindi di avvicinarci quanto più possibile alla definizione di cui sopra con il *modello stocastico markoviano dinamico* sostituendo il passo 5 del modello statico con il seguente:

5. Per ogni v tale che $(u, v) \in E$:
   (a) Se $v \notin F$ allora $F = F \cup v$, $p_v = \sigma_u$ e parent$(v) = u$,
   (b) altrimenti se $\sigma_u > p_v$, $p_v = \sigma_u$ e parent$(v) = u$.

5.3 Architettura per le visite

L'implementazione dell'algoritmo è stata fatta in linguaggio Java (specifiche versione 1.5) basandosi sul codice del package `it.unimi.dsi.law.wga` per le visite simulate, ovvero che permettono di sperimentare differenti strategie di visita su grafi del web compressi con *WebGraph*. Una visita di questo tipo

è composta da tre oggetti principali: un `Dispenser`, un `Visitor` ed un `AbstractWebGraph`.

Le implementazioni dei primi, che discendono dalla classe astratta `Dispenser` decidono l'effettivo ordine di visita degli archi. Posseggono infatti principalmente due metodi, `int get()` e `void put(u,v)`, che rispettivamente restituiscono il prossimo nodo da visitare e inseriscono nella struttura la coppia di nodi (u, v). Se v è un nodo del seme allora vale -1, altrimenti (u, v) è l'arco che attraversa la frontiera, dove $u \in F$, che per la prima volta porta ad una visita del nodo v. Infatti, ogni nodo compare come secondo argomento di `put` una volta soltanto. I tipici dispenser già forniti con il pacchetto implementano la visita per ampiezza, quella randomica, quella per profondità.

I secondi posseggono un solo metodo di largo utilizzo che è `int visit(v)`, che viene richiamato su v quando questo nodo viene emesso dal dispenser. Dopo aver operato sul nodo in se (solitamente una qualche statistica) il risultato di `visit` istruissce l'architettura sul fatto se gli archi uscenti da v debbano essere considerati o meno o se addirittura la visita debba interrompersi. I `visitor` più utilizzati sono ad esempio quello che stampa i nodi mentre vengono visitati o quello che produce una curva della qualità dei nodi pian piano che vengono visitati.

Le implementazioni di `AbstractWebGraph` collegano gli elementi necessari a questo tipo di viste inglobando gli `ImmutableGraph` di *Webgraph* e altre strutture dati corollarie al grafo, come il suo trasposto, la mappa tra indici dei nodi ed URL e viceversa, o una generica nozione di qualità di un singolo nodo (espresso come valore decimale).

Il pacchetto `it.unimi.dsi.law.pebl4j` contiene le classi Java che implementano l'algoritmo descritto; esso include anche il codice descritto nel Capitolo 4. Data la natura sperimentale della tesi, il codice è stato pensato per poter essere quanto più efficiente e rapido possibile, permettendo di ottenere i risultati degli esperimenti in tempo breve anche a discapito della quantità di memoria utilizzata.

La classe `AugmentedNaiveBayesClassifier` implementa il classificatore naive bayesiano multiclasse sogliato descritto nel Capitolo 2 e qui riutilizzato. Tutti i conteggi delle probabilità sono sostituiti in pratica dalel stesse operazioni sui loro logaritim; in particolare, essendo necessaria soltanto la molitplicazione di probabilità esse vengono sostituite da somme sui logaritmi, in modo da evitare problematiche di underflow. Dovendo riaddestrare il modello bayesiano ogni volta, oltre alla mappa delle log-probabilità di apparizione dei termini, vengono mantenuti (con la stessa sintassi del capitolo 2) $|T_j|$ e le somme dei conteggi sui singoli termini come le occorrenze generiche di ogni documento. In questo modo è possibile ricostriure rapidamente la probabilità semplicemente aggiungendo i valori del nuovo set di addestramento e ricalcolando il logaritmo della frazione.

La classe `HiddenMarkovModel` implementa il modello markoviano nascosto, utilizzando lo stesso accorgimento riguardante i logaritmi e il contenimento dei conteggi da cui derivano le probabilità per accelerare il riaddestramento. Ogni qualvolta è disponibile una nuova osservazione il metodo `observeAndInfer` permette di inserirla nell'HMM e far calcolare, secondo la sequenza di genitori coinvolta, lo stato finale della sequenza di stati globalmente più probabile.

Infine la classe `HMMDispenser`, utilizzabile direttamente per la visita, combina il modello markoviano nascosto, il classificatore B-PrTTFIDF e il classificatore naive bayes sogliato per ottenere la classe di algoritmi descritta in questo capitolo. Per permettere la maggiore flessibilità ottenibile durante gli esperimenti la classe, che esegue pedissequamente l'algoritmo di modello statico o dinamico; ulteriormente il dispenser supporta i seguenti parametri

- *useTFIDF* : utilizza o meno la rappresentazione TF-IDF (piuttosto che TF) per la classificazione bayesiana. Di default vero.
- *hmmStoreFile*: permette di scrivere su file i dati del modelo markoviano una volta terminata la visita. Contiene il nome del file.
- *hmmLoadFile*: permette di effettuare la visita utilizzando un modello markoviano precalcolato, evitando quindi di aggiornarne le probabilità.
- *bayesStoreFile*: simile a hmmStoreFile.
- *bayesLoadFile*: simile a hmmLoadFile.
- *maxDistance*: la massima distanza coperta dall'algoritmo. Solitamente, 6.
- *alpha*: il valore di α nel calcolo delle priorità. Di default 0.5.
- *useLiveClassifier*: Vero (di default falso) nel qual caso si debba utilizzare l'opzione ptiDataFile per caricare i dati al fine di utilizzare un classificatore B-PrTFIDF online. Se falso utilizza il vettore di qualità dei nodi di `AbstractWebGraph` come valori di probabilità di appartenenza a C_+.
- *ptiDataFile*: Datafile serializzato della soluzione del problema di training di B-PrTFIDF.
- *progressFile*: Indica un eventuale file di testo dove in ogni riga porre il nodo visitato (numerico), la sua probabilità, l'osservazione connessa e lo stato generato dall'HMM.
- *allEdges*: Vero (di default vero) quando il modello da applicare è quello dinamico.

Capitolo 6

Risultati

Il presente capitolo introduce i risultati sperimentali ottenuti con gli algoritmi presentati nel capitolo 5 confrontando le varie versioni sia tra di loro sia con altri algoritmi di visita.

Per valutare le prestazioni degli algoritmi dati si utilizza la collezione descritta nel capitolo 3 nei tagli da 100.000, 1.000.000, e 10.000.000 nodi; qualora non venga specificato altrimenti tutte le visite si intendono essere proposte con distanza massima 6.

L'insieme dei nodi rilevanti viene considerato essere quello per cui $\Pr[\boldsymbol{x} \in C_+] > 0.5$ secondo il classificatore B-PrTFIDF precedentemente addestrato. Poichè la collezione creata nel capitolo 3 non è dotata di etichette è stato necessario creare manualmente un training set al fine di concretizzare l'istanza del crawler mirato. A tal scopo, è stato creato un solo set di 300 pagine classificate manualmente come rilevanti per il training set, più altri 200 per il validation set, estratti dalla collezione come parte di siti di interesse politico e governativo. Inoltre, il classificatore è stato addestrato con un campione estratto casualmente dalla collezione di 3000 indrizzi per il training set e 2000 per il validation set.

La prima metrica di confronto è l'area sottostante la curva di F_1-measure che ricordiamo essere espressa come $\frac{2pr}{p+r}$ dove p, r indicano precisione e richiamo. Sia nel nostro caso $B(t)$ il numero di nodi rilevanti collezionati durante la visita all'istante di visita t, $|R|$ la cardinalità dell'insieme dei rilevanti. Possiamo dunque riscrivere la F_1-measure come

$$F_1 = \frac{2 \times B(t)/t \times B(t)/|R|}{B(t)(1/t + 1/|R|)} = \frac{2B^2(t)}{t|R|} \frac{t|R|}{B(t)(t + |R|)} = \frac{2B(t)}{t + |R|}$$

Calcolata puntualmente, valuteremo poi l'andamento dell'F_1-measure attraverso l'utilizzo di un grafico.

È comunque di interesse ottenere un unico valore numerico in grado di esprimere la bontà della visita. A tale scopo usiamo una variante della misura di preferenza binaria (BPref) descritta in [8], definendola su una generica

visita W:

$$\text{Pref} = 1 - \frac{1}{|V||R|} \sum_{x \in R \cap W} |\{y \in W \cap R^c | y <_r x\}|$$

dove la relazione $a <_r b$ vale se a precede b nella visita. Discorsivamente, a meno della normalizzazione, la Pref individua per ogni nodo rilevante il numero dei nodi non rilevanti che lo precedono temporalmente nella visita e somma queste quantità. Questo rende la Pref un valore compreso tra 0 e 1 che valuta sinteticamente la bontà della visita: il nostro scopo è avere una Pref molto alta al termine della visita.

6.1 Confronto tra strategie

I primi esperimenti mettono a confronto sulle tre taglie di visita algoritmi differenti. Un primo concorrente è la semplice visita per ampiezza, che seleziona i nodi secondo una coda senza priorità. Conseguentemente ci si misura contro una visita casuale che in ogni momento della visita estrae dalla frontiera un nodo scelto casualmente. Il generatore casuale per le visite sui vari tagli viene inizializzato sempre allo stesso valore.

Non avendo implementazione di riferimento per il modello con grafi di contesto o con modello markoviano, poniamo come concorrente più competitivo l'algoritmo di visita Best-First, descritto nel capitolo 1, che assegna il punteggio di $\Pr[\boldsymbol{x} \in C_+]$ di ogni pagina come priorità dei suoi figli, estraendo poi i nodi dalla frontiera con una visita con priorità [1].

Le visite sono state effettuate utilizzando l'implementazione sotto forma di Dispenser degli algoritmi di cui sopra, oltre che le versioni statica e dinamica del nuovo algoritmo proposto. Gli esperimenti sono stati condotti con JVM HotSpot VM 1.6.02beta5 a 64 bit su sistema operativo Linux Fedora su un server Sun Fire X2200M con 2 processori AMD Opteron 2200 (64-bit) a 2.8 Ghz Dual Core dotata di 32 GB di RAM e 2 dischi SATA-II da 7200 RPM. Il processo di visita si rivela in ogni caso sempre CPU-bound e i tempi di visita sono descritti nella tabella sottostante (in formato ore/minuti/secondi)

Metodo	10^5 nodi	10^6 nodi	10^7 nodi
BFS (Breadth-First)	00:00:24	00:03:15	00:30:21
Best-First	00:02:26	00:04:15	00:42:03
Casuale	00:00:26	00:03:20	00:34:56
Modello statico	00:03:00	01:55:39	-
Modello dinamico	00:03:30	02:03:15	11:46:28

Le figure 6.1, 6.2 e 6.3 descrivono l'andamento della F_1-measure e del numero di nodi rilevanti ($B(t)$) per i tre tagli di visita. Ad esse corrispondono i valori di Pref al termine della visita tabulati di seguito:

[1] La priorità dei nodi già in frontiera non viene mai riassegnata.

Metodo	10^5 nodi	10^6 nodi	10^7 nodi
Best-First	0.9870	0.9676	0.97174
BFS (Ampiezza)	0.9480	0.9491	0.98190
Modello statico	0.9818	0.9697	-
Modello dinamico	0.9839	0.9711	0.98843
Casuale	0.9512	0.9480	0.97974

Come si mostra, le prestazioni delle visite non mirate (ampiezza e casuale) sono povere e decadono subito. La visita Best-First e il modello markoviano invece hanno andamenti molto vicini sulle visite di piccole dimensioni, ma l'evidenza dei grafici mostra che con il tempo il modello markoviano si comporta molto meglio. La motivazione è anche da ricercarsi nel campione utilizzato: il grafo da 10^5 nodi contiene 5000 elementi che sono utilizzati per l'addestramento; inoltre quest'ultimo contiene una grossa componente fortemente connessa di nodi rilevanti, che si replica identica nel grafo da 10^6 elementi. In queste componenti, l'ingordigia del metodo Best-first garantisce l'esplorazione dell'intera componente appena vi si entra.

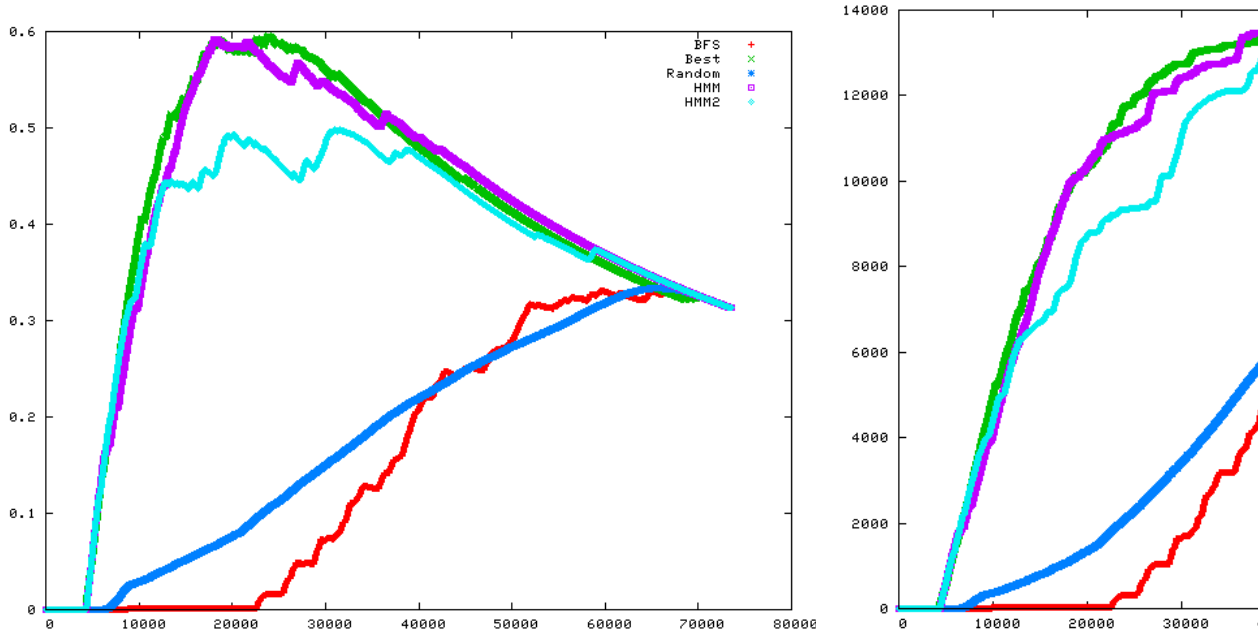
Il grafo colorato di figura 6.4 mostra visivamente il grado di connessione della componente connessa; esso è stato prodotto "riassumendo" il grafo da 10^5 nodi: ogni nodo disegnato rappresenta un gruppo di pagine a massima distanza reciproca 3 (ogni pagina appare in un solo nodo) il cui grado di colorazione è proporzionale al numero di pagine rilevanti rappresentate dal nodo stesso. Due nodi del grafo sono connessi da un arco se vi sono archi tra le pagine rappresentate. L'immagine rappresenta i soli nodi (e conseguenti archi coinvolti) che contengono almeno una pagina di interesse ed è disegnato separando le componenti connesse. Gli archi sono direzionati tuttavia per mantenere una buona chiarezza di disegno le frecce indicanti la direzione sono state rimosse.

Sul grafo da 10^6 nodi le prestazioni dell'algoritmo proposto in questa tesi sono (specialmente per la Pref) superiori, seppur in misura limitata, dell'euristica Best-First. Il grafo da 10^6 nodi risulta comunque dotato di una grossa componente ereditata da quello più piccolo e di componenti minori sparse: la moderata (poi più accentuata nel grafo da 10^7 nodi) superiorità del modello markoviano proposto indica comunque che l'idea è funzionale e funzionante. Difatti, nella visita su collezioni più grandi la visita ingorda perde prestazioni fino a posizionarsi addirittura al di sotto di visite cieche come quelle per ampiezza o casuale.

Su ogni grafo, comunque, si nota che le performance dell'algoritmo che utilizza il riassegnamento dinamico dei padri sono maggiori.

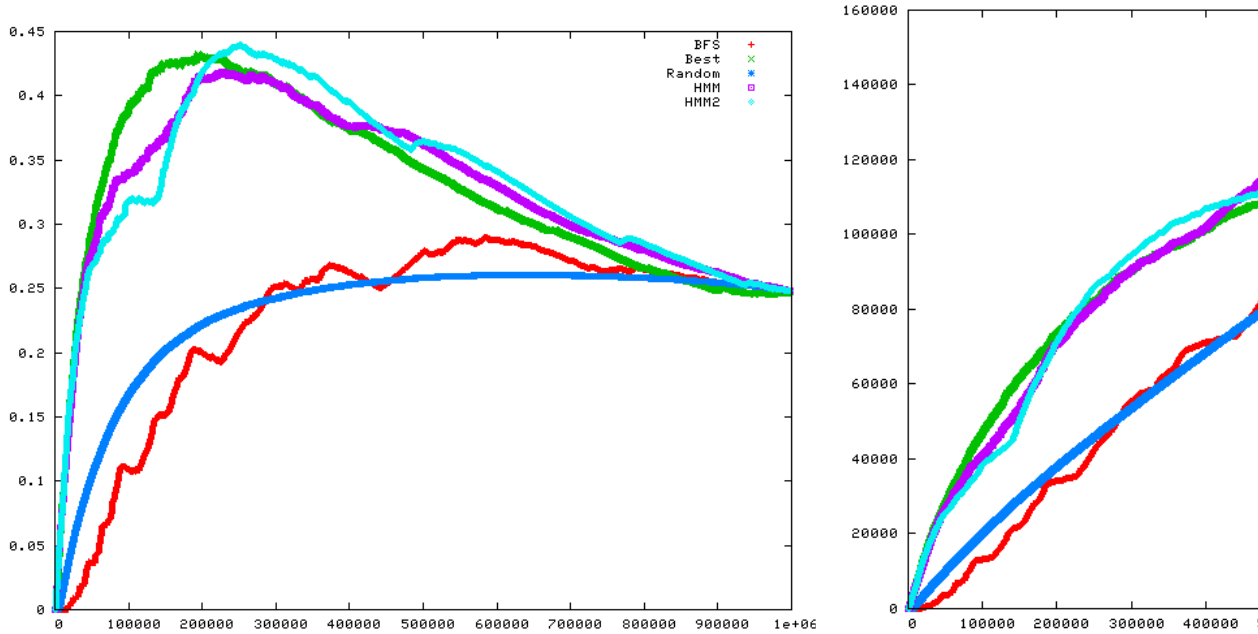
6.2 Analisi dei modelli markoviani

In questa sezione andiamo ad analizzare proprietà specifiche del gruppo di algoritmi proposti. Per confermare le funzionalità del modello specialmente



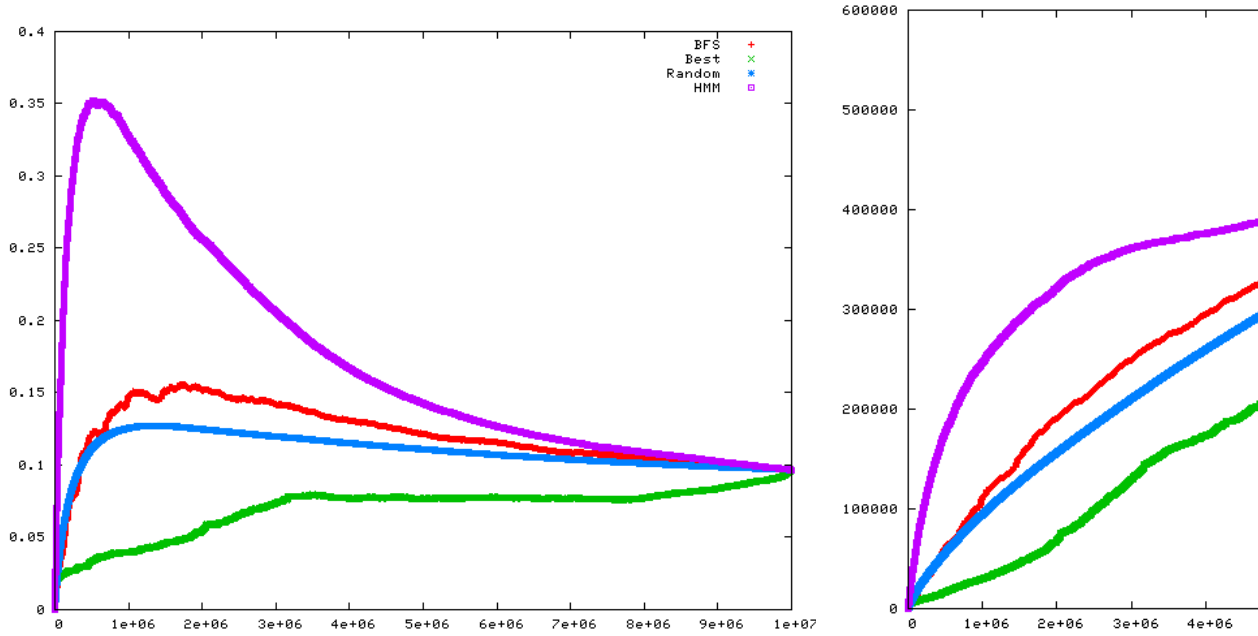
(a) (b)

Figura 6.1: (a) F_1-measure e (b) andamento dei rilevanti durante le visite su 100.000 nodi. HMM indica il modello dinamico, HMM2 quello statico.



(a) (b)

Figura 6.2: (a) F_1-measure e (b) andamento dei rilevanti durante le visite su 1.000.000 nodi. HMM indica il modello dinamico, HMM2 quello statico.



(a) (b)

Figura 6.3: (a) F_1-measure e (b) andamento dei rilevanti durante le visite su 10.000.000 nodi. HMM indica il modello dinamico.

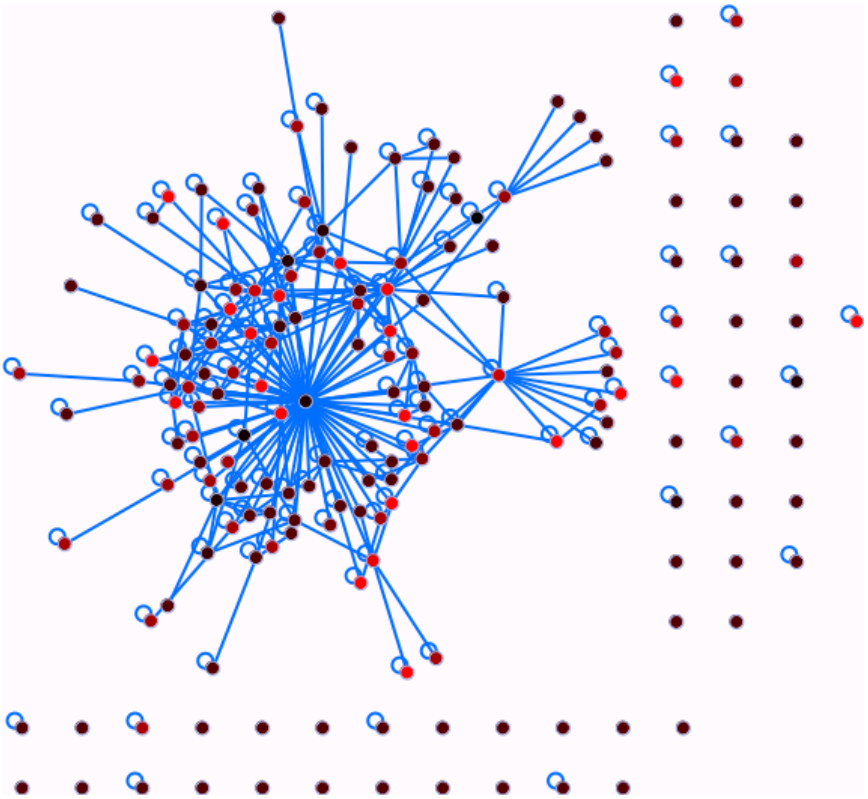
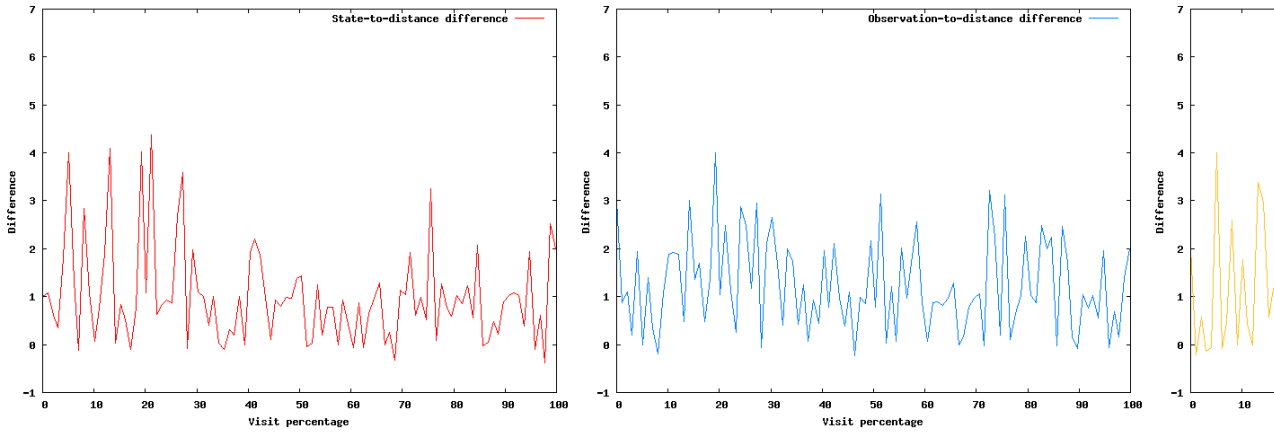


Figura 6.4: Grafo riassuntivo dell'interconnessione di gruppi di 3 pagine contenti almeno una rilevante riferita al grafo originale da 10^5 nodi. Il grado di colorazione è proporzionale al numero di pagine rilevanti rappresentate dal nodo.



(a) (b)

Figura 6.5: Differenze (a) stato-distanza, (b) osservazione-distanza e (c) osservazione-stato durante una visita con modello dinamico su 10^6 nodi

rispetto al metodo con grafi di contesto possiamo osservare i grafici di figura 6.5, costruiti dopo aver etichettato separatamente ogni nodo con la minima distanza da un nodo rilevante (entro la distanza coperta, quindi 6). I grafici mostrano una spline che interpola i punti rappresentati dalla differenza tra stato e reale distanza, osservazione e distanza ed osservazione e stato, pertanto rappresentano soltanto un andamento generico riassuntivo, tuttavia utile per alcune considerazioni. Il primo grafico mostra come con l'andamento della visita la differenza tra distanza e stato tenda comunque a restringersi, infatti i picchi della curva sono minori e più bassi. Il secondo grafico illustra invece che l'osservazione e l'effettiva distanza mantengono un grado di scorrelazione costante per tutta la visita, indica che comunque, data anche l'eterogeneità dei documenti, i classificatori bayesiano sono buoni ad ottenere una riduzione dello spazio delle pagine in classi di distanza ma che non sono buoni stimatori della distanza. Il terzo grafico tenta di rispondere alla congettura fatta in precedenza: se i classificatori bayesiani fossero perfetti, si giungerebbe ad un modello markoviano degenere, ovvero sarebbe possibile eliminare la componente strutturale? La risposta sembra essere "solo in parte", in quanto mentre si verifica un addolcimento della curva iniziale, anche dovuto all'addestramento dell'HMM la frequenza dei picchi e la loro altezza rimane pressochè invariata. Questi grafici comunque rappresentano soltanto un approssimazione dell'andamento e sono necessarie evidenze su istanze diverse del crawler e l'analisi di visite di taglia più grande per confermare l'analisi appena fatta.

Continuando l'analisi del modello, si collezionano dati su come le varie versioni dell'algoritmo e l'apprendimento *online* influiscano sull'andamento

della visita. I grafici di figura 6.6 mostrano varie istanze del modello, valutati però con valori di α pari a 0.75, per accentuare maggiormente il solo contributo dell'HMM. Per quanto già il grafico della sola F_1-measure dia una prima idea delle differenze, il grafio dell'andamento della misura Pref riportato crea una chiara classifica tra i vari metodi, esattamente come ci si aspetta. I valori di Pref al termine della visita sono:

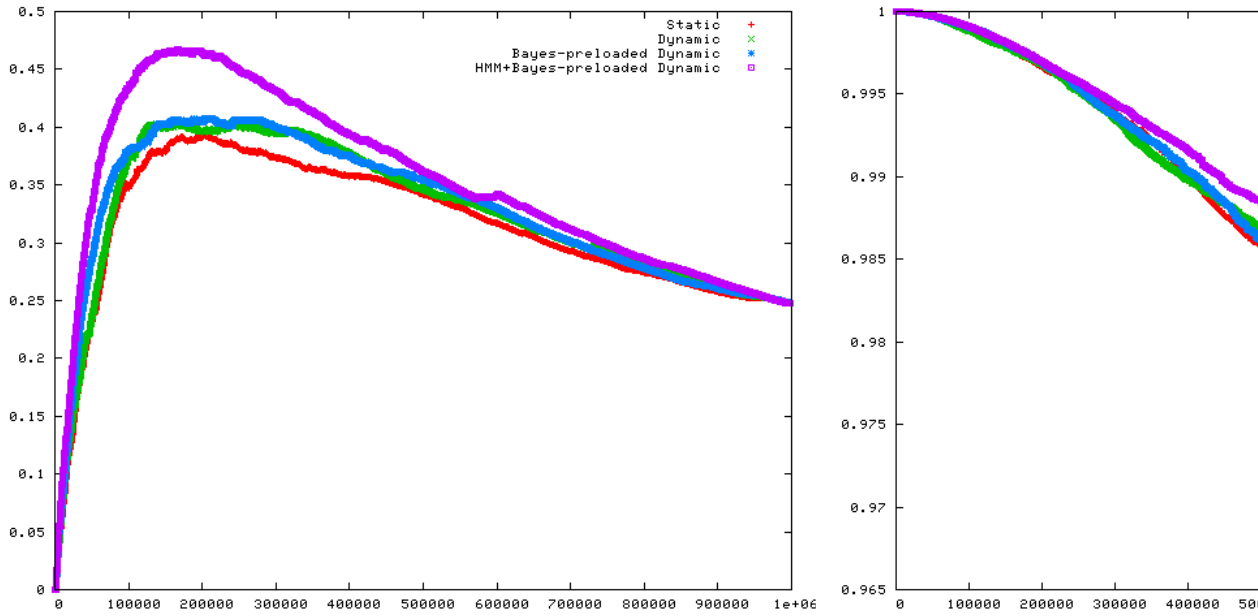
Tipo di istanza	Pref
Statica	0.96620
Dinamica	0.96903
Dinamica con Bayesiano precalcolato	0.96946
Dinamica con Bayesiano e HMM precalcolati	0.97428

Il metodo statico per quanto funzionale è comunque limitato, mentre l'algoritmo dinamico si dimostra maggiormente efficace. Le altre due visite si riferiscono a situazioni ipotetiche in cui sia già disponibile un modello bayesiano e un HMM risultanti da una visita precedente che non debbono essere ri-addestrati durante la visita. I grafici mostrano ulteriormente che i classificatori bayesiani addestrati migliorano sì la visita, ma comunque limitatamente. Al contrario, il preaddestramento del modello markoviano nascosto permette una visita nettamente migliore. Questo tipo di comporamento è comprensibile, in quanto mentre il classificatore bayesiano utilizza esempi per creare $O(d)$ gruppi di probabilità gli elementi del modello markoviano sono $O(d^2)$, poichè si addestra sulle transizione da uno stato all'altro.

Un ulteriore esperimento molto significativo da compiere è la variazione della massima distanza coperta dal modello (in questo caso useremo la versione dinamica). I valori di Pref sono:

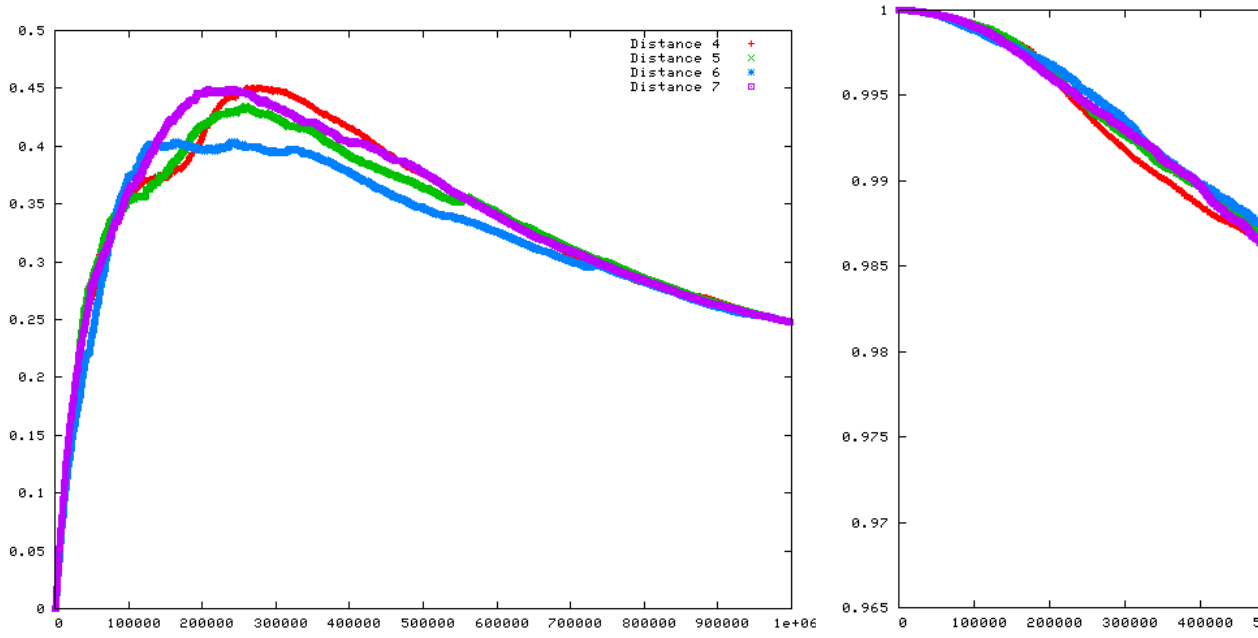
Distanza	Pref
4	0.97320
5	0.97217
6	0.97118
7	0.97335

I risultati in figura 6.7 mostrano come questo parametro sia particolarmente cruciale per ottenere una buona visita, per quanto il comportamento possa sembrare ad una prima vista controsenso. Infatti, le visite a distanza 4 e distanza 7 hanno circa la stesse capacità di previsione, mentre la visita a distanza 6 produce la Pref più bassa; questi risultati confermano le osservazioni di [19], dove si asserisce che variare la massima distanza coperta implica diminuire la capacità di previsione dell'algoritmo però garantisce una migliore capacità di apprendimento in quanto la distribuzione delle aree semantiche è più sfumata ed eterogenea tanto più si vuole valutare classi di distanza più ampie. Possiamo estendere queste considerazioni aggiungendo che il grafico



(a) (b)

Figura 6.6: (a) F_1-measure e (b) andamento della Pref durante la visita su 10^6 nodi. Modello statico, dinamico, dinamico con utilizzo di bayesiano da una visita precedente, dinamico con utilizzo di HMM e Bayesiani da una visita precedente. $\alpha = 0.75$.



(a) (b)

Figura 6.7: (a) F_1-measure e (b) andamento della Pref durante una visita su 10^6 nodi con modello dinamico a diversi distanze massime.

evidenzia come l'apprendibilità dell'appartenenza ad alcune classi di distanza sia più o meno semplice, con buona probabilità dipendentemente dalla disposizione e struttura testuale delle pagine rilevanti. In questo senso ci si propone di effettuare ulteriori esperimenti in futuro, tentando anche di tenere in considerazione le aree semantiche cui le varie pagine appartengono.

6.3 Conclusioni

La parte di compressione dati ha ottenuto dei risultati ampiamente soddisfacenti pur avvalendosi principalmente di valutazioni empiriche, cui nel futuro ci si pone l'obbiettivo di trovare motivazioni teoriche più forti.

L'algoritmo proposto ha dimostrato, specialmente nelle situazioni di ricerca di zone isolate del web, una ottima capacità di scelta delle pagine da visitare, sia in condizioni di apprendimento *offline* che *online*, nonostante i contesti utilizzati permettono all'euristica di confronto di operare vicina al massima della sua efficacia. Il modello di tipo dinamico, inoltre, risulta essere un'aggiunta importante e ben motivata. La parte sperimentale esplora, nonostante la possibilità di usare un solo argomento di visita, il comportamento del crawler nelle sue varianti, valutando anche il costo computazionale dello stesso, risultato comunque accettabile.

Le argomentazioni presentate forniscono comunque ulteriori spunti sia per altre analisi della teoria retrostante l'algoritmo sia per il progetto di variazioni dell'algoritmo presentato. In particolare, per quanto riguarda la prima osservazione, le valutazioni finora fatte risentono molto dell'impossibilità di potersi confrontare contro una visita di tipo onnisciente che sia per definizione la migliore secondo una qualche misura (esempio, la precisione). Oltretutto, il riaddestramento dei bayesiani riassegna osservazioni solo ad alcune pagine, poichè è computazionalmente improponibile di riclassificare l'intera frontiera, per tanto l'algoritmo di Viterbi agisce su una stringa distorta di osservazioni. Manca in questo senso una analisi precisa e puntuale delle possibilità di errore dovute a queste imprecisione. Un simile discorso può essere fatto sul riassegnamento dei padri, in quanto la visita permette di aggiornare i valori di probabilit'à solo per un numero ristretto di possibili predecessori.

Infine, dati i risultati della tesi, si nota come la componente strutturale sia particolarmente importante nel learning. L'assunzione che la transizione tra gli stati rappresentanti le distanze sia markoviana di primo ordine è completamente arbitraria, ed in questo senso lo studio di modelli markoviani nascosti del secondo ordine è una direzione promettente per la ricerca di algoritmi di focused crawling.

Bibliografia

[1] 20ng: A sample from 20 newsgroup, 2005, `www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/data/news20.html`.

[2] H. Bast and I. Weber. Type less, find more: fast autocompletion search with a succinct index. *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 364–371. ACM Press, 2006.

[3] H. Bauke. Parameter estimation for power-law distributions by maximum likelihood methods. *The European Physics Journal B* 58:167–173, 2007.

[4] D. Bergmark, C. Lagoze, and A. Sbityakov. Focused crawls, tunneling and digital libraries. *Proceedings of the ECDL02*, 2002.

[5] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. *COLT: Proceedings of the Workshop on Computational Learning Theory, Morgan Kaufmann Publishers*, pp. 92-100, 1998, `citeseer.ist.psu.edu/blum98combining.html`.

[6] P. Boldi and S. Vigna. The webgraph framework i: Compression techniques, 2003.

[7] P. Boldi and S. Vigna. Codes for the world wide web. *Internet mathematics* 2(4):405–427, 2005.

[8] C. Buckley and E. Voorhees. Retrieval evaluation with incomplete information. *27th Annual International ACM SIGIR (Conference on Research and Development in Information Retrieval)*, pp. 25–32, 2004.

[9] S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: a new approach to topic-specific Web resource discovery. *Computer Networks (Amsterdam, Netherlands: 1999)* 31(11–16):1623–1640, 1999, `citeseer.ist.psu.edu/chakrabarti99focused.html`.

[10] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[11] P. Chen, C. Lin, and B.Scholkopf. A tutorial on nu-support vector machines. *Proceedings of the ASMBI*, pp. 111-136, 2005.

[12] J. Cho, H. García-Molina, and L. Page. Efficient crawling through URL ordering. *Computer Networks and ISDN Systems* 30(1–7):161–172, 1998, `citeseer.ist.psu.edu/article/cho98efficient.html`.

[13] F. Denis, A. Laurent, and et al. Text classification and co-training from positive and unlabeled examp les, 1999, `citeseer.ist.psu.edu/716088.html`.

[14] M. Diligenti, F. Coetzee, S. Lawrence, C. L. Giles, and M. Gori. Focused crawling using context graphs. *26th International Conference on Very Large Databases, VLDB 2000*, pp. 527–534, 10–14 September 2000, `citeseer.ist.psu.edu/diligenti00focused.html`.

[15] Open directory project, `www.dmoz.org`.

[16] S. W. Golomb. Run-length encodings. *IEEE Transactions on Information Theory* 12(3):399, 1966.

[17] J. Kleinberg. Authoritative sources in a hyperlinked environment. *9th ACM-SIAM Symposium on Discrete Algorithms*, 1998.

[18] R. Kosaraju and G. Manzini. Compression of low entropy strings with Lempel-Ziv algorithms. *SIAM Journal of Computing* 29(3):893–911, 1999.

[19] H. Liu, E. Milios, and J. Janssen. Probabilistic models for focused web crawling. *WIDM '04: Proceedings of the 6th annual ACM international workshop on Web information and data management*, pp. 16–22, 2004.

[20] F. Menczer, G. Pant, and P. Srinivasan. Topical web crawlers: Evaluating adaptive algorithms, 2003, `citeseer.ist.psu.edu/menczer03topical.html`.

[21] Managing gigabytes 4 java, `mg4j.dsi.unimi.it`.

[22] K. Nigam, A. K. McCallum, S. Thrun, and T. M. Mitchell. Learning to classify text from labeled and unlabeled documents. *Proceedings of AAAI-98, 15th Conference of the American Association for Artificial Intelligence*, pp. 792–799, 1998, `citeseer.ist.psu.edu/nigam98learning.html`.

[23] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Tech. rep., Stanford Digital Library Technologies Project, 1998, `citeseer.ist.psu.edu/page98pagerank.html`.

[24] L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *IEEE*, vol. 77, pp. 257–285, no. 2, 1989.

[25] J. Rennie and A. K. McCallum. Using reinforcement learning to spider the Web efficiently. *Proceedings of ICML-99, 16th International Conference on Machine Learning*, pp. 335–343, 1999, `citeseer.ist.psu.edu/article/rennie99using.html`.

[26] The R project for statistical computing, `www.r-project.org`.

[27] B. Schölkopf, o Platt, J. Shawe-Taylor, A. Smola, and R. Williamson. Estimating the support of a high-dimensional distribution, 1999, `citeseer.ist.psu.edu/sch01estimating.html`.

[28] The WebGraph framework, `webgraph.dsi.unimi.it`.

[29] World wide knowledge base project, 2005, `www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-11/www/wwkb/index.html`.

[30] H. Yu. Single-class classification with mapping convergence. *Mach. Learn.* 61(1-3):49–69, 2005.

[31] H. Yu, J. Han, and K.-C. Chang. PEBL: Positive example-based learning for web page classification using SVM. *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery in Databases*, pp. 239–248, 2002.

[32] D. Zhang and W. S. Lee. A simple probabilistic approach to learning from positive and unlabeled examples. *Proceedings of the 5th Annual UK Workshop on Computational Intell igence (UKCI)*, 2005.

[33] H. Zhang. The optimality of naive bayes. *17th International FLAIRS Conference*, 2004.