IMPLEMENTAZIONE DI ALGORITMI DI LINK ANALYSIS E CAMPIONAMENTO DEL WEB PER LA LORO VALUTAZIONE

ATTIVITÀ FORMATIVA: ELABORATO

LUCIA PETTERLE

Relatore: Prof. Luca Pretto
Correlatore: Prof. Enoch Peserico

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica
Anno Accademico 2010/2011

a mia nonna Sergia,
alle mie montagne,
a Pianaz

Sommario

I primi sistemi di Information Retrieval reperivano documenti rilevanti in risposta ad interrogazioni prelevandoli da collezioni omogenee, autorevoli, e di qualità mediamente elevata. L'approccio tradizionale non è risultato adeguato alle esigenze di reperimento di informazione all'interno del web, a causa della natura eterogenea e qualitativamente variegata di quest'ultimo. Proprio in tale contesto nascono gli algoritmi di link analysis, allo scopo di cogliere la qualità delle pagine web in risposta alle interrogazioni; essi individuano e sfruttano il valore informativo insito nella struttura topologica del grafo del web, interpretando i link come raccomandazioni di qualità.

Il presente lavoro si inserisce all'interno di un progetto di valutazione dell'efficacia degli algoritmi di link analysis. Il progetto si propone di valutare la coerenza esistente tra giudizi umani raccolti mediante interviste circa la qualità di un limitato insieme di pagine web e punteggi computati dagli algoritmi sulle stesse pagine. Il nostro lavoro è consistito innanzitutto nell'implementazione dei quattro principali algoritmi di link analysis, PageRank, HITS, SALSA e Indegree; in seguito nella scelta della tecnica di campionamento più adeguata per ottenere il bacino di pagine da valutare e nella descrizione delle modalità con cui i risultati della ricerca, ottenuti sul campione prelevato, potranno essere estesi alla popolazione.

Indice

Capitolo 1

Introduzione al progetto

1.1 Progetto *Qualità*

Il progetto *Qualità*, ideato e sviluppato presso il Dipartimento di Ingegneria dell'Informazione dell'Università degli Studi di Padova e a cui il lavoro documentato in questo elaborato contribuisce, ha come scopo la valutazione dell'effettiva efficacia di funzionamento dei più importanti algoritmi di link analysis. È mirato a verificare se esiste effettivamente una forma di corrispondenza tra i giudizi umani di qualità espressi su un insieme di pagine web e i corrispondenti punteggi di qualità computati meccanicamente dai quattro più importanti algoritmi di link analysis: PageRank [2], HITS [4], SALSA [5] e Indegree. Ciò consiste nel verificare un'ipotesi che già nella letteratura scientifica sull'argomento [1, 10] ha trovato spazio nel recente passato, ossia che sia insita nella struttura ad hyperlink del web una valutazione di qualità delle sue pagine, da estrapolare mediante gli algoritmi di link analysis.

Il progetto ha richiesto, prima di tutto, la collaborazione di due gruppi di studenti di Ingegneria Informatica, che si sono occupati rispettivamente di:

1. effettuare il crawling del web italiano, formulare e realizzare la modalità migliore per costruire un grafo che ne rappresentasse adeguatamente la struttura e che fosse efficiente in termini di occupazione di spazio di memoria;

2. sviluppare un'applicazione atta alla raccolta delle interviste qualitative sulle pagine web e formulare il campionamento del web basato sulla caratteristica di rilevanza dello stesso.

La presente relazione documenta un lavoro concettualmente e cronologicamente posteriore a quello appena descritto, e che, pertanto, ha potuto usufruire dei risultati già ottenuti.

1.2 Impegno specifico all'interno del progetto

Il lavoro documentato nel presente elaborato si è articolato in due fasi ed attività distinte. Innanzitutto si è provveduto all'implementazione degli algoritmi di link analysis. In seguito ci si è occupati di scegliere e giustificare l'euristica con cui si procederà ad estrarre dal bacino di pagine web restituite da un'interrogazione un sottoinsieme ridotto di pagine in modo da rendere possibile la raccolta delle interviste qualitative. Infine, sempre in relazione all'attività di campionamento appena descritta, è stata formulata la modalità statistica con cui inferire dal risultato che si otterrà sul campione un risultato globale.

La prima fase, descritta nel capitolo 2, non si è limitata semplicemente ad un'attività di programmazione, ma ha comportato un impegno analitico molto più articolato. Questa ragione è principalmente da riferirsi alle ingenti dimensioni del grafo del web italiano, costituito indicativamente da 300 milioni di nodi, fatto che ha condotto ad effettuare delle scelte implementative strutturate ad hoc.

La seconda fase, trattata nel capitolo 3, è motivata dalla necessità di ridurre la notevole dimensione del bacino di indagine della ricerca e, al contempo, di indurre la validità generale del risultato a partire da quella ristretta al suddetto bacino considerato. Ciò ha comportato un'attività preliminare di documentazione sui temi statistici del campionamento e dell'inferenza statistica per variabili qualitative. In seguito, poi, l'impegno è stato quello di adattare gli elementi teorici acquisiti al particolare caso in esame.

Capitolo 2

Implementazione di algoritmi di link analysis

2.1 Panoramica sulle scelte di programmazione e sulle strutture dati utilizzate

Gli algoritmi implementati[1] ai fini del progetto sono stati i quattro più diffusi algoritmi di link analysis: PageRank, HITS, SALSA e Indegree. Allo scopo di apprendere le conoscenze necessarie per la loro implementazione sono stati consultati gli articoli in cui essi, eccetto Indegree – mai formalizzato – sono stati presentati.

Il linguaggio di programmazione utilizzato è stato il linguaggio Java. La scelta si è dimostrata adeguata alle necessità implementative.

L'input comune agli algoritmi di link analysis implementati è rappresentato da un file che memorizza il grafo del web italiano, che in questa sede, per semplicità, chiamiamo *file compatto del grafo del web*. Dopo aver effettuato un'operazione di crawling del grafo, si è optato per la seguente rappresentazione compatta dello stesso in un file salvato su disco fisso:

- ogni nodo è identificato univocamente da una stringa binaria, detta id, a 32 bit;
- il bit più significativo di ciascun id è un bit di 'genitorialità', vale 1 per i nodi considerati come genitori e 0 per i nodi considerati come figli;

[1] Ne verrà proposta una descrizione ad alto livello in questo capitolo, per maggiori dettagli su codice sorgente e scelte implementative si rimanda alle appendici.

- ogni nodo del grafo compare una e una sola volta nel file come genitore (con il bit più significativo impostato a 1);
- consecutivamente ad un nodo genitore vengono elencati gli id dei relativi figli (con il bit più significativo impostato a 0).

Le principali difficoltà nell'implementazione degli algoritmi si sono riscontrate nella modalità di computazione e di memorizzazione dei punteggi associati ai nodi del grafo, a causa delle ingenti dimensioni di quest'ultimo. Si sono dovute introdurre opportune scelte di programmazione che mediassero tra la necessità di disporre di tali dati in strutture che permettessero l'effettiva computazione e quella di consentire la presenza dei dati stessi in memoria centrale, essendo impensabile operare continui trasferimenti tra processore e memoria di massa.

Come suggerito in [2, 4, 5, 3], la modalità più semplice ed efficiente, in termini di tempo, per l'implementazione degli algoritmi di link analysis è quella di disporre di una matrice di adiacenza per la rappresentazione del grafo del web e di vettori per i punteggi calcolati per i nodi, in modo da permettere di effettuare le computazioni attraverso semplici operazioni di prodotto matrice per vettore. Nel nostro caso questo tipo di approccio è risultato assolutamente impossibile da attuarsi per ragioni legate allo spazio di memoria: considerate le dimensioni stimate del web italiano di circa 300 milioni di nodi e considerata la realizzazione di una matrice di booleani, lo spazio occupato supererebbe i 10000 TB. La variante di utilizzare matrici sparse è risultata anch'essa inadeguata per ragioni legate all'occupazione di spazio di memoria. Supponiamo, con una stima altamente ottimistica, di avere una media di 10 link uscenti da ogni pagina web e di utilizzare una terna di array per la memorizzazione della matrice di adiacenza normalizzata molto sparsa (nel primo array si salvano gli elementi non nulli della matrice, nel secondo l'indice di riga dei precedenti e nel terzo l'indice di colonna). L'occupazione di memoria per i soli tre array, senza tenere presente che dovrebbero essere memorizzati anche i vettori dei punteggi, supererebbe i 13 GB.

Si è pertanto concluso che la scelta più adeguata è quella di mantenere in memoria di massa il file compatto del grafo del web e di mantenere in memoria RAM il vettore (per alcuni algoritmi è necessario memorizzarne due) dei punteggi computati per tutti i nodi, aggiornandolo attraverso una lettura lineare del file compatto, con modalità appositamente pensate per ciascuno degli algoritmi implementati, in seguito specificatamente descritte.

2.2 PageRank

PageRank è un algoritmo sviluppato da Sergey Brin e Lawrence Page e presentato nell'articolo "The anatomy of a large scale hypertextual Web search engine" (In Proc. of the WWW Conference, 1998).

In quanto algoritmo di link analysis, PageRank parte dal presupposto di tenere in considerazione nella computazione dei punteggi dei nodi, il contenuto informativo insito nella struttura topologica ad hyperlink del grafo del web; esso interpreta un link come un *voto* che la pagina da cui il link esce attribuisce alla pagina in cui il link entra. L'idea di fondo dell'algoritmo è che i voti provenienti da siti importanti (autorevoli, di qualità) dovrebbero avere un peso maggiore dei voti provenienti da siti meno importanti, e l'importanza di un voto da una qualunque sorgente dovrebbe essere attenuata dal numero dei siti che tale sorgente vota (numero di link uscenti da tale pagina).

Il funzionamento di PageRank è stato dedotto logicamente dalla formulazione di un modello descrittivo dell'attività di navigazione di un utente nel web, il modello del *random surfer*. Sulla scia di tale analogia, il punteggio attribuito da PageRank ad una pagina rappresenta la probabilità con cui un utente che naviga in modo randomico nella rete si trova, in un certo istante discreto, in tale pagina.

Prima di applicare l'algoritmo di PageRank al grafo del web è necessario rilevare l'esistenza di due situazioni critiche nell'attività di navigazione modellata dal random surfer e descrivere le modalità con cui esse vengono gestite dall'algoritmo. La prima si ha in caso di accesso a pagine web prive di link uscenti, dette *nodi sink*; in questa situazione, in assenza di accorgimenti che estendano la nozione di grafo, la navigazione subirebbe uno stallo, rendendo impossibile la computazione del punteggio di PageRank. Il problema viene superato considerando l'esistenza di link fittizi uscenti dai nodi sink e diretti verso tutti i restanti nodi del grafo; la soluzione è assolutamente realistica e modella il comportamento dell'utente che accede ad una pagina priva di link in uscita immaginando che egli prosegua la navigazione in modo casuale nel web. La seconda situazione critica è rappresentata dall'esistenza di collegamenti tra nodi che forzano la navigazione a ripercorrere uno stesso insieme finito di nodi con periodicità costante. Questo problema viene risolto introducendo, nella modellazione dell'attività di navigazione, una costante, denominata *damping factor* (solitamente fissata a 0.85), che rappresenta la probabilità con cui un utente che si trova su una certa pagina web prosegue la navigazione scegliendo uno dei link uscenti dalla pagina stessa; di riflesso la componente complementare al damping factor costituisce la

probabilità con cui l'utente prosegue la navigazione accedendo casualmente ad una qualunque tra tutte le pagine del grafo del web. L'introduzione di questo secondo accorgimento, oltre ad evitare eventuali ciclicità nell'accesso alle pagine web che renderebbero impossibile il calcolo dei punteggi, permette anche di aggiungere al modello del random surfer il tipico carattere di casualità di cui è caratterizzata la reale attività di navigazione nel web.

2.2.1 Caratteristiche implementative della classe PageRank

Input:

- file compatto del grafo del web.

Output:

- file dei punteggi di PageRank di tutti i nodi del grafo in input.

L'algoritmo PageRank è di tipo query-independent, quindi il calcolo dei punteggi dovrà essere effettuato per tutti i nodi del grafo del web. Come già in precedenza anticipato, la struttura dati più adeguata e agevole per l'implementazione della funzione è l'array. Gli id consecutivi dei nodi del grafo del web vengono utilizzati, una volta tradotti da stringhe binarie nei corrispondenti interi decimali, come indici per accedere rapidamente alle celle contenenti i punteggi di PageRank di ciascuno di essi.

La struttura dell'algoritmo è di tipo iterativo. A partire da un vettore iniziale di PageRank che attribuisce a ciascuno degli N nodi del grafo un punteggio pari a $\frac{1}{N}$ (che esprime l'equiprobabilità iniziale di trovarsi in uno qualunque dei nodi del grafo del web), in ciascuna iterazione si calcola la probabilità con cui la navigazione interessa ciascuno dei nodi in funzione dell'insieme di probabilità computate al passo precedente, fino a raggiungere un certo grado di precisione imposto a priori.

Ogni iterazione, fino alla convergenza, comporta una lettura dell'intero file in input. Ogni volta che viene scandito un nodo padre ne vengono memorizzati in un array di supporto i figli. Segue un controllo per distinguere due casi da trattare separatamente: nodi con figli e nodi sink.

Vengono attribuiti contributi parziali di punteggio a tutti i discendenti dei nodi con figli: essi distribuiscono ai figli il proprio punteggio di PageRank frazionato in funzione della numerosità m dei figli stessi:

$$P'[x] \longleftarrow \sum_{y:(y,x)\in E} \frac{P'[y]}{m_y} \quad \forall x \in V$$

dove x e y sono nodi, y è un nodo con figli, (y, x) è un arco orientato da y a x, E è l'insieme degli archi e V è l'insieme dei nodi.

Nel caso dei nodi sink, i loro punteggi di PageRank vengono equidistribuiti frazionalmente a tutti gli N nodi del grafo, simulando l'aggiunta, ai nodi privi di link uscenti, di archi fittizi verso tutti i restanti nodi e verso se stessi:

$$P'[x] \longleftarrow P'[x] + \frac{P'[y]}{N} \quad \forall x \in V$$

dove y è un nodo sink.

Dopodiché i punteggi vengono ponderati in funzione del damping factor d. Viene introdotto il fenomeno del *teleporting*, lo spostamento casuale del random surfer in uno qualunque tra i nodi del grafo:

$$P[x] \longleftarrow \frac{(1-d)}{N} + d \sum_{y:(y,x)\in E} \frac{P'[y]}{m_y} \quad \forall x \in V$$

ottenendo così il vettore definitivo e normalizzato dei punteggi di PageRank.

L'occupazione di memoria RAM per i punteggi di PageRank, a causa della natura iterativa dell'algoritmo (che richiede di mantenere i punteggi di due iterazioni consecutive) e a causa del suo essere query-independent, consiste nello spazio richiesto da due array di elementi di tipo double (64 bit) di circa 300 milioni di celle: indicativamente 4.8 GB.

2.3 HITS

HITS, acronimo per Hyperlink-Induced Topic Search, è un algoritmo sviluppato da J.M. Kleinberg e presentato nell'articolo "Authoritative sources in a hyperlinked environment" (Journal of the ACM, 46(5):604-632, Sept. 1999).

L'idea che ne sta alla base è quella di coniugare un approccio semplificato come quello dell'Indegree ad una più sottile valutazione del significato della struttura ad hyperlink del web.

La validità di HITS si riferisce in particolare alla ricerca di collezioni di pagine autorevoli per le cosiddette *broad-topic query*, interrogazioni su argomenti di carattere generale. Il modello di HITS considera la relazione esistente tra le pagine autorevoli, denominate *authority*, per un'interrogazione di carattere generale, e quelle che presentano link verso numerose authority, dette *hub*; l'algoritmo identifica e sfrutta il valore informativo di entrambe le tipologie di pagine. L'obiettivo, precisamente, è quello di estrarre dal bacino di pagine su cui l'algoritmo viene applicato, quelle ad elevato punteggio di

authority sfruttando, a tal fine, le indicazioni di autorevolezza fornite dalle pagine con elevato punteggio di hub.

HITS è un algoritmo di tipo query-dependent, pertanto partiamo dall'assunzione di avere a disposizione un'interrogazione definita da una stringa σ.

A partire dall'interrogazione, per mezzo di un motore di ricerca, si ottengono le prime $t \approx 200$ pagine più rilevanti in risposta alla stessa. Esse, se considerate nel contesto di tutto il grafo del web, costituiscono una sorta di sottografo, particolarmente destrutturato, che chiamiamo $G[R_\sigma]$. Da esso si ricostruisce un sottografo $G[S_\sigma]$, un'estensione di $G[R_\sigma]$: si tratta di aggiungere ai nodi presenti in $G[R_\sigma]$ tutti i relativi nodi figli e un massimo di $d \approx 50$ nodi genitori per ogni nodo. Nel complesso il sottografo $G[S_\sigma]$ presenta tre fondamentali caratteristiche: è relativamente piccolo, contiene molte pagine rilevanti e contiene la maggior parte delle più forti authority. L'approccio computazionale di HITS sul grafo $G[S_\sigma]$ per l'estrazione delle pagine più autorevoli parte dalla seguente euristica: intuitivamente non è sufficiente estrarre pagine con elevato Indegree, perché tra queste includeremmo anche pagine ad elevata popolarità, che non necessariamente comporta alta autorevolezza. Per questa ragione, ipotizzando ragionevolmente che le authority su un tema comune siano nel complesso puntate da un insieme di hub comuni, si individuano nel grafo sia i nodi ad elevato punteggio di authority, che quelli ad elevato punteggio di hub. L'algoritmo, più approfonditamente descritto nella sezione 2.3.3, concretizza un approccio che tiene conto di quanto segue [4]:

> Hub e authority mostrano quella che può essere chiamata *relazione di mutuo rinforzo*: un buon hub è una pagina che punta a molte buone authoriry; una buona authority è una pagina che è puntata da molti buoni hub.

Caratteristiche implementative

L'implementazione di HITS ha, nel complesso, richiesto la realizzazione di tre classi:

- GrafoHITS;
- IdHITS;
- HITS.

2.3.1 Caratteristiche implementative della classe GrafoHITS

Input:

- file contenente gli id, a 32 bit, delle t pagine selezionate dal motore di ricerca in risposta all'interrogazione σ;
- file compatto del grafo del web.

Output:

- file contenente gli id, a 32 bit, dei nodi costituenti il sottografo $G[S_\sigma]$, la cui struttura ricalca quella del file compatto del grafo del web.

Questa classe si occupa di scrivere il sottografo $G[S_\sigma]$ su cui verrà applicato l'algoritmo di HITS. Ricordiamo che il grafo $G[S_\sigma]$ è costituito dai t nodi restituiti dall'interrogazione σ, cui vengono aggiunti tutti i relativi nodi figli e un massimo di d genitori per nodo.

Innanzitutto devono essere salvati i t nodi restituiti dall'interrogazione σ in due array, nella 'versione' con bit più significativo pari a 0 e in quella con bit più significativo pari a 1. Questi due array verranno utilizzati per facilitare le ricerche dei nodi, sia sotto forma di figli che di padri. Viene quindi effettuato un ciclo di lettura del file compatto del grafo del web:

- ogniqualvolta il nodo scandito nel file compatto del grafo del web viene trovato tra i t nodi, esso viene scritto nel file di output assieme a tutti i propri figli;
- se, al contrario, il nodo scandito non è presente tra i t nodi, si verifica se sono presenti i figli di tale nodo, letti consequenzialmente allo stesso nel file compatto del grafo del web
  - ogniqualvolta un nodo figlio nel file compatto del grafo del web viene trovato tra i t nodi, vengono scritti nel file di output, dopo aver controllato che ciò non sia già stato fatto per nodi precedentemente scanditi, il padre e il nodo stesso.

Si mantiene la convenzione, anche per il file prodotto in output, di identificare i genitori con il bit più significativo impostato a 1 e i figli con bit più significativo impostato a 0.

2.3.2 Caratteristiche implementative della classe IdHITS

Input:

- file contenente gli id, a 32 bit, dei nodi costituenti il sottografo $G[S_\sigma]$, la cui struttura ricalca quella del file compatto del grafo del web.

Output:

- file rappresentante una nuova versione $G'[S_\sigma]$ del suddetto sottografo, in cui gli id dei nodi non sono più a 32 bit, ma sono a 14 bit; in modo che, se M sono in tutto i nodi di $G'[S_\sigma]$, essi vengano rappresentati da id inclusi tra 0 e il binario corrispondente al decimale M-1.

Questa classe si occupa semplicemente di associare univocamente a ciascun nodo del sottografo $G[S_\sigma]$, avente id a 32 bit, un id a 14 bit. Approssimativamente, la cardinalità dell'insieme dei nodi di $G[S_\sigma]$ si colloca attorno alle 1000-5000 unità [4]. Sono quindi largamente sufficienti 13 bit, cui si aggiunge un bit, il più significativo, per indicare la 'genitorialità'. Dopo aver effettuato l'associazione univoca degli id, l'algoritmo effettua la ricostruzione della struttura del sottografo $G[S_\sigma]$, con la produzione in output di un nuovo sottografo che chiameremo $G'[S_\sigma]$, i cui nodi hanno gli id sopra citati, a 14 bit.

2.3.3 Caratteristiche implementative della classe HITS

Input:

- file del grafo $G'[S_\sigma]$.

Output:

- file con i punteggi di authority e di hub per tutti i nodi del grafo in input.

È la classe che costituisce l'effettivo corpo dell'algoritmo HITS.

Quest'ultimo, come visto, è di tipo query-dependent, quindi il calcolo dei punteggi dovrà essere effettuato per un numero abbastanza limitato di nodi, al massimo 5000. I punteggi verranno memorizzati in due vettori, rispettivamente per i valori di hub e di authority. Gli id consecutivi dei nodi del grafo del web vengono utilizzati, una volta tradotti da stringhe binarie nei corrispondenti interi decimali, come indici per accedere rapidamente alle celle contenenti i punteggi di HITS di ciascuno di essi.

La struttura dell'algoritmo è di tipo iterativo. Denotando con x il vettore dei punteggi di authority e con y il vettore di punteggi di hub, a partire da un vettore iniziale y che attribuisce a ciascuno dei nodi del grafo $G'[S_\sigma]$ un punteggio unitario, ad ogni iterazione il codice implementato effettua le quattro operazioni descritte in seguito. Innanzitutto inizializza a 0 il vettore di authority x. Prosegue realizzando un'operazione di aggiornamento del vettore x in modo da attribuire a ciascun nodo un punteggio di authority pari alla somma dei punteggi di hub dei nodi che lo puntano:

$$x^{<p>} \longleftarrow \sum_{q:(q,p)\in E} y^{<q>} \quad \forall p \in V$$

dove p e q sono nodi, E è l'insieme degli archi e V è l'insieme dei nodi. Realizza un'operazione di aggiornamento del vettore y in modo da attribuire a ciascun nodo un punteggio di hub pari alla somma dei punteggi di authority dei nodi da esso puntati:

$$y^{<p>} \longleftarrow \sum_{q:(p,q)\in E} x^{<q>} \quad \forall p \in V$$

Infine effettua la normalizzazione dei vettori x e y in modo che la somma dei quadrati dei loro elementi sia unitaria: ciò avviene dividendo i vettori, componente per componente, per la radice della sommatoria dei quadrati di tutti i loro elementi:

$$x^{<p>} \longleftarrow \frac{x^{<p>}}{\sqrt{\sum_{t\in V}(x^{<t>})^2}} \quad \forall p \in V$$

$$y^{<p>} \longleftarrow \frac{y^{<p>}}{\sqrt{\sum_{t\in V}(y^{<t>})^2}} \quad \forall p \in V$$

Empiricamente, secondo Kleinberg [4], si raggiunge la convergenza dopo circa 20 iterazioni.

L'occupazione di memoria RAM per i punteggi di HITS consiste nello spazio richiesto da due array di elementi di tipo double (64 bit) di un massimo di circa 5000 celle: lo spazio necessario è esiguo, stimato attorno ai 40 KB.

2.4 SALSA

SALSA, acronimo per Stochastic Approch for Link-Structure Analysis, è un algoritmo sviluppato da R. Lempel e S. Moran e presentato nell'arti-

colo "SALSA: The stochastic approach for link-structure analysis" (ACM Transactions on Information Systems, 19(2):131-160, Apr. 2001).

L'idea a fondamento di SALSA è sostanziamente comune a quella su cui si basa HITS. Come nel lavoro di Kleinberg [4], anche in quello di Lempel e Moran si parte dal presupposto di considerare la distinzione tra due tipologie di nodi: hub e authority. La visione dicotomica dei nodi è motivata, anche in questo caso, dai seguenti principi:

- l'entità del punteggio di hub di un nodo è determinata dall'entità dei punteggi di authority dei nodi cui esso punta;
- l'entità del punteggio di authority di un nodo è determinata dall'entità dei punteggi di hub dei nodi da cui esso è puntato.

In aggiunta rispetto all'approccio di HITS, SALSA combina la suddetta distinzione tra i ruoli di hub e authority con la teoria dei cammini randomici su un grafo (su cui si fonda PageRank). Consideriamo un grafo bipartito G in cui i nodi compaiono duplicati, sotto forma di componente di hub e di componente di authority, e in cui la presenza di un arco tra un hub r e una authority s indica l'esistenza di un link informativo da r a s. Sicuramente i nodi con più elevato punteggio di hub e di authority relativamente ad un determinato argomento saranno quelli più facilmente raggiungibili da un'elevata quantità di pagine del grafo G. SALSA si propone di identificare le pagine di G che risultano maggiormente visitate da cammini randomici sul grafo, dalle quali poi potrà estrarre quelle con maggiore punteggio di authority.

L'approccio rigorosamente formalizzato di SALSA è di tipo iterativo. Esso prende avvio da un'analogia simile a quella a fondamento di PageRank: sul grafo bipartito G vengono modellati due distinti cammini, uno che attraversa solo le componenti hub, uno che attraversa solo le componenti authority; ciascuno dei due cammini percorre in ogni istante discreto due archi, in modo da rimanere confinato solo tra le componenti hub o solo tra quelle authority. I punteggi attribuiti ai nodi sono direttamente proporzionali alle probabilità con cui la navigazione randomica interessa gli stessi in ogni istante discreto e il loro calcolo viene effettuato attraverso operazioni iterative di prodotto matrice per vettore, fino alla convergenza di tali probabilità.

Dopo aver descritto l'approccio iterativo, gli stessi Lempel e Moran, dimostrano che, complessivamente, l'intero algoritmo può essere ricondotto ad una semplice versione di Indegree pesato, nel caso in cui gli hyperlink del grafo vengano considerati tutti di uguale valenza informativa. In questo

caso, infatti, SALSA attribuisce ai nodi punteggi di hub e di authority semplicemente proporzionali rispettivamente al numero di link in uscita dagli stessi e in ingresso agli stessi. I punteggi che interessano effettivamente ai fini di ricavare le pagine più autorevoli e di qualità sono solo quelli di authority; per questa ragione diciamo che SALSA si riconduce semplicemente ad una forma di Indegree pesato.

Tratteremo separatamente il caso particolare in cui i link abbiano pesi in generale differenti tra loro.

2.4.1 Caratteristiche implementative della classe SALSA

Input:

- file del grafo $G'[S_\sigma]$.

Output:

- file con i punteggi di authority e di hub per tutti i nodi del grafo in input.

SALSA è un algoritmo di tipo query-dependent e il grafo su cui opera è lo stesso grafo utilizzato da HITS e generato, a partire dai nodi restituiti da un'interrogazione, dalla classe GrafoHITS precedentemente descritta. La computazione dei punteggi di hub e authority deve essere effettuata per tutti i nodi appartenenti a tale grafo; ancora una volta la struttura dati utilizzata è stata l'array. Gli id consecutivi dei nodi del grafo del web vengono utilizzati, una volta tradotti da stringhe binarie nei corrispondenti interi decimali, come indici per accedere rapidamente alle celle contenenti i punteggi di SALSA di ciascuno di essi.

Sulla base delle precedenti considerazioni, dal punto di vista computazionale l'algoritmo si riduce semplicemente al calcolo, per tutte le pagine dell'insieme S_σ, della somma dei link in esse entranti e da esse uscenti (rispettivamente per il computo dei punteggi di authority e di hub non normalizzati):

$$x^{<p>} \longleftarrow \sum_{q:(q,p)\in E} 1 \quad \forall p \in V$$

cui viene fatta seguire una normalizzazione dei risultati ottenuti:

$$x^{<p>} \longleftarrow \frac{x^{<p>}}{\sum_{t\in V} x^{<t>}} \quad \forall p \in V$$

Si è pensato di realizzare anche il calcolo dei punteggi di hub, nonostante non siano necessari, per tre ragioni: la loro computazione non incide sulla complessità computazionale dell'algoritmo, lo spazio di memoria aggiuntivo richiesto è esiguo e può essere interessante confrontare i punteggi di hub attribuiti da HITS con quelli attribuiti da SALSA.

La struttura dell'algoritmo si adatta alle caratteristiche del file in input e una sola lettura di quest'ultimo consente il calcolo dei vettori dei punteggi di hub e authority per tutte le pagine. La struttura del file, infatti, che vede l'id di un genitore seguito dagli id di tutti i relativi figli, è descrittiva di un insieme di link uscenti da ciascun genitore verso i figli. Per calcolare il punteggio di authority non normalizzato di una pagina, un nodo del grafo, è semplicemente sufficiente incrementare un contatore per ciascuno dei figli scanditi del nodo stesso, letti di seguito ad esso nel file in input. Per calcolare il punteggio di hub non normalizzato di una pagina, invece, è semplicemente sufficiente incrementare un contatore associato a tale nodo ogniqualvolta esso compare come figlio. Terminata la lettura del file in input e, dunque, la descritta attribuzione dei punteggi, viene effettuata la normalizzazione degli stessi.

L'occupazione di memoria RAM per i punteggi di SALSA consiste nello spazio richiesto da due array di elementi di tipo double (64 bit) di un massimo di circa 5000 celle: lo spazio necessario è esiguo, stimato attorno ai 40 KB.

Caso particolare: link pesati

L'implementazione di SALSA appena descritta, nel caso in cui i nodi del sottografo su cui l'algoritmo viene applicato siano tutti debolmente connessi tra loro, produce un ordinamento delle pagine per punteggio computato esattamente uguale a quello prodotto utilizzando Indegree.

Possiamo descrivere una variante della precedente implementazione: il caso particolare in cui i link del sottografo su cui viene applicato l'algoritmo abbiano pesi, in generale, differenti.

Per i punteggi di authority, consideriamo che ogni link in uscita da un nodo s abbia un peso $w_a(s)$ pari al reciproco della somma dei link uscenti da quel nodo:

$$w_a(s) \longleftarrow \frac{1}{\sum_{r:(s,r)\in E} 1} \quad \forall s \in V$$

Il calcolo dei punteggi di authority, vettore x, viene effettuato attraverso la somma, per ciascuno dei nodi del sottografo, dei pesi w_a dei link in essi

entranti:

$$x^{<p>} \longleftarrow \sum_{q:(q,p)\in E} w_a(q) \quad \forall p \in V$$

Terminato il calcolo di tutti i punteggi, ne viene effettuata una normalizzazione:

$$x^{<p>} \longleftarrow \frac{x^{<p>}}{\sum_{t\in V} x^{<t>}} \quad \forall p \in V$$

In modo analogo si possono ottenere i punteggi di hub; in questo caso ogni link in ingresso ad un nodo p ha un peso $w_h(p)$ pari al reciproco della somma dei link entranti in tale nodo. I punteggi di hub si ottengono sommando, per ciascuno dei nodi del sottografo, i pesi w_h dei link da essi uscenti, e normalizzando il vettore risultante.

La struttura dell'algoritmo è quella descritta precedentemente per la versione originale di SALSA, cui si deve aggiungere un ulteriore ciclo di lettura del file in input se si vuole effettuare anche il conteggio dei punteggi di hub.

L'occupazione di memoria RAM è la stessa calcolata per SALSA, circa 40 KB.

2.5 Indegree

Indegree è l'algoritmo precursore e fondatore di tutti gli algoritmi di link analysis sia per semplicità di funzionamento che per agevolezza computazionale. La sua euristica non è mai stata formalizzata, ma di fatto consiste nel presupposto comune a tutti gli algoritmi di link analysis.

L'idea fondante Indegree è quella di inferire l'importanza di una pagina web direttamente sulla base alla quantità di link entranti nella stessa. È ragionevole pensare, infatti, che la presenza di un collegamento da un documento digitale ad un altro contribuisca a conferire a quest'ultimo una forma di importanza, mediante una raccomandazione di qualità da parte dell'autore della prima pagina verso i contenuti della seconda.

2.5.1 Caratteristiche implementative della classe Indegree

Input:

- file compatto del grafo del web.

Output:

- file dei punteggi di Indegree di tutti i nodi del grafo in input.

L'algoritmo Indegree è di tipo query-independent, quindi il calcolo dei punteggi dovrà essere effettuato per tutti i nodi del grafo del web. La scelta implementativa più semplice e conveniente è quella di utilizzare un array di interi per il salvataggio dei punteggi. Gli id consecutivi dei nodi del grafo del web vengono utilizzati, una volta tradotti da stringhe binarie nei corrispondenti interi decimali, come indici per accedere rapidamente alle celle contenenti i punteggi di Indegree di ciascuno di essi.

La struttura dell'algoritmo si adatta alle caratteristiche del file di input, il file compatto del grafo del web. Dopo l'inizializzazione del vettore di punteggi di Indegree, attraverso un solo ciclo di lettura del suddetto file, è possibile terminare il computo totale dei punteggi di tutti i nodi. La struttura del file, infatti, che vede l'id di un genitore seguito dagli id di tutti i relativi figli, è descrittiva di un insieme di link uscenti da ciascun genitore verso i figli. Rovesciando la prospettiva sulla base delle nostre necessità, per ogni nodo padre scandito nel file di input, è possibile attribuire a ciascuno dei suoi figli un incremento unitario del punteggio di Indegree per effetto della presenza di un link in esso entrante e proveniente, per l'appunto, dal genitore scandito. Sia x il vettore di punteggi di Indegree:

$$x^{<p>} \longleftarrow \sum_{q:(q,p)\in E} 1 \quad \forall p \in V$$

L'occupazione di spazio di memoria in RAM per i punteggi di Indegree consiste nello spazio richiesto da un array di elementi di tipo int (32 bit) di circa 300 milioni di celle: indicativamente nel complesso 1.2 GB.

Capitolo 3

Campionamento

3.1 Motivazioni del campionamento

La nostra ricerca mira a valutare l'efficacia di funzionamento dei più importanti algoritmi di link analysis: PageRank, HITS, SALSA e Indegree. Vogliamo verificare se i punteggi che i suddetti algoritmi attribuiscono alle pagine web a partire da un'analisi della struttura topologica del grafo si possono considerare un valido indicatore di qualità: elevato punteggio è, statisticamente parlando, assicurazione di elevata qualità delle pagine web che lo possiedono (e viceversa)?

A tale scopo la nostra indagine prevede di sottoporre al giudizio empirico di una platea di intervistati un insieme di pagine web. Dal momento che la popolazione di pagine restituite da un motore di ricerca ad una generica interrogazione ne conta circa 1000, non è pensabile che la valutazione le consideri tutte. Ciò comporta l'esigenza di introdurre un processo di campionamento sulla suddetta popolazione.

Interverrà, in seguito, la statistica inferenziale, quella branca della statistica che sostituisce all'analisi di un dato universo quella di un campione estratto da esso, precisando al contempo il grado di attendibilità con cui le conclusioni tratte per il campione possono essere estese all'intera popolazione.

3.2 Presupposti per la scelta del campionamento

Il campionamento dovrà essere effettuato in modo tale che il campione estratto abbia carattere e numerosità tali da concludere la validità dell'inferenza statistica: si dovrà poter asserire che ciò che vale, statisticamente,

per il campione, con una certa confidenza impostata a priori, vale anche per l'intera popolazione.

Una volta conosciute le caratteristiche di distribuzione dei punteggi per l'universo in esame, auspicabilmente un insieme di elementi prelevati da esso, un campione, dovrebbe andare a costituire una sottopopolazione per cui la distribuzione dei punteggi segue ancora tale andamento.

3.2.1 Tipologie di campionamento probabilistico

La teoria campionaria distingue due macro-categorie di campionamento, quello probabilistico e quello non probabilistico. Quest'ultimo non consente l'applicazione del principio di inferenza statistica, in quanto preleva un campione secondo euristiche di praticità e senza rispettare le leggi del caso per la sua selezione. Volendo giungere a conclusioni la cui validità possa essere estesa al collettivo, escludiamo il campionamento non probabilistico e focalizziamo l'attenzione su quello probabilistico.

Esistono diverse tipologie di campionamento probabilistico: campionamento casuale semplice, sistematico, stratificato, a stadi, per aree e a grappoli. Pur precisando che il lavoro di documentazione ha riguardato tutte le tipologie citate, allo scopo di valutare il tipo di campionamento più adeguato al caso in esame, non ci è parso opportuno dilungarci in questa sede sulle relative descrizioni, che esulano dallo scopo di questa relazione.

3.2.2 Caratteristiche di un campione valido

Affinché i risultati di una indagine campionaria siano generalizzabili è necessario valutare la validità del campione.
Un campione è valido se è:

- Eterogeneo: include tutte le caratteristiche e qualità diverse. Il campione deve essere diversificato al suo interno in modo da rappresentare una variabilità di caratteristiche collegate alle informazioni da rilevare.

- Rappresentativo: presenta, senza distorsioni, tutte le caratteristiche della popolazione di riferimento. La rappresentatività è garantita se la procedura di campionamento è casuale, cioè se è regolata dalla legge del caso/probabilità.

- Accurato: il grado di minimizzazione degli errori di copertura è elevato e tutti gli elementi della popolazione sono potenzialmente prelevabili.

- Ampio: include un elevato numero di casi. L'ampiezza è inversamente proporzionale agli errori di rilevazione.

3.2.3 Caratteristiche della popolazione in esame

In un campionamento è fondamentale la conoscenza dell'universo: per poter estrarre in modo probabilistico i casi dalla popolazione è necessario disporre di informazioni circa le caratteristiche della popolazione stessa.

Nel nostro caso, l'universo in esame è costituito dal migliaio di pagine web restituite da un'interrogazione e la caratteristica da tenere in osservazione è la distribuzione, per tali pagine, dei relativi punteggi (di PageRank, HITS, SALSA e Indegree). Una volta che si hanno a disposizione i punteggi delle pagine di interesse, è possibile rappresentarli in un piano cartesiano avente in ascissa la quantità di pagine e in ordinata i relativi punteggi. Si troverà [8] che le distribuzioni seguiranno un andamento matematicamente interpolabile con una power-law.

La distribuzione power-law [9], o distribuzione a legge di potenza, in termini generali ha la forma:

$$p(x) \propto \mathcal{L}(x) \cdot x^{-\alpha} \tag{3.1}$$

Dove $\alpha > 1$ è detto esponente e $\mathcal{L}(x)$ è una funzione che "cresce lentamente", ossia tale per cui $\lim_{x\to\infty} \frac{\mathcal{L}(tx)}{\mathcal{L}(x)} \to 1$, per $t = costante$. Qualora $\mathcal{L}(x)$ fosse costante, la (3.1) diventerebbe:

$$p(x) = C \cdot x^{-\alpha}$$

3.3 La scelta: il campionamento sistematico

Il campionamento sistematico è un particolare tipo di campionamento casuale semplice in cui si utilizzano una o più caratteristiche quantitative precedentemente acquisite sulla popolazione in esame per ordinarne le singole unità. Sulla popolazione opportunamente riordinata si esegue poi una procedura che porta ad estrarre gli n elementi del campione, seguendo la seguente euristica:

- si calcola il coefficiente intero k, pari al reciproco della frazione di campionamento: se N è la numerosità della popolazione e n quella del campione, si ha che $k = \frac{N}{n}$ (se k non è intero si sceglierà $\lceil k \rceil$ o $\lfloor k \rfloor$, adattandosi rispettivamente ad un leggero incremento o ad una leggera riduzione della dimensione del campione);

- si estrae casualmente un intero j appartenente all'intervallo chiuso $[1, k]$;

- si estraggono dalla popolazione gli n elementi che, nell'ordinamento effettuato, occupano la posizione $(k \cdot i + j)$-esima per ogni $i \in [0, 1, \ldots, n-1]$.

Il campionamento sistematico si utilizza quando, essendo in possesso di una lista completa della popolazione, se ne intende studiare un carattere specifico e determinante, in relazione al quale sia possibile effettuare il suddetto ordinamento dei suoi elementi. Prelevando poi, a partire dalla prima unità, che viene estratta in modo casuale, i vari elementi secondo un passo costante di campionamento, si assicura che le singole unità del campione siano distribuite uniformemente all'interno della popolazione.

3.4 Giustificazioni del campionamento scelto

A fronte di un'attenta valutazione delle diverse tipologie di campionamento, delle proprietà richieste per la validità del campione e delle caratteristiche di distribuzione disomogenea dei punteggi computati, il campionamento che risulta più adatto al caso in esame è il campionamento sistematico. Esso ci consente di recuperare dalla popolazione un campione con caratteristiche analoghe, in termini di percentuali statistiche, e notevolmente inferiori, in termini di dimensioni, a quelle della popolazione.

Nel nostro specifico caso, l'ordinamento della popolazione avverrà in funzione dei punteggi computati da parte degli algoritmi di link analysis; potremo scegliere un ordinamento per punteggio crescente o decrescente. Il prelievo del campione secondo la disciplina di campionamento sistematico ci permetterà di raggiungere l'obiettivo precedentemente citato, ossia quello di ricostruire un campione che rispetti i rapporti numerici presenti nella realtà. In termini empirici, grazie al campionamento sistematico, rappresentando in un piano cartesiano (avente in ascissa il numero di pagine e in ordinata i relativi punteggi) anche per il campione i punteggi calcolati dagli algoritmi, otterremo che la distribuzione degli stessi seguirà un andamento analogo a quello registrato per l'intera popolazione, ossia un andamento matematicamente interpolabile con una power-law.

3.5 Inferenza statistica per variabili qualitative

Una volta determinato il campione, si avranno a disposizione le pagine su cui verranno condotte le interviste qualitative presso un'opportuna platea di intervistati. Dopo aver terminato la raccolta delle interviste, campione e relativi giudizi alla mano, si potrà procedere all'attività di induzione di un risultato generale a partire da quello particolare ottenuto. Si tratta di affrontare un problema di inferenza statistica per variabili qualitative [11, 12]; nel nostro caso la variabile statistica qualitativa è rappresentata dalla *coerenza* tra giudizi umani e punteggi numerici computati.

3.5.1 Frequenze e proporzioni

Prima di tutto è necessario determinare la *frequenza* a della caratteristica che interessa. Nel nostro caso, una volta analizzata la coerenza tra i punteggi computati dagli algoritmi sulle pagine del campione e i giudizi umani raccolti, definire la frequenza significherà fornire il numero a ($\leq n$, dove n è la numerosità del campione) di pagine web per cui si è verificato empiricamente il successo dell'ipotesi che vogliamo dimostrare, cioè la correttezza di funzionamento degli algoritmi di link analysis.

Spesso la frequenza viene espressa in termini statistici, dopo averla rapportata al totale degli elementi del campione, ottenendo in questo modo la cosiddetta *frequenza relativa* o *proporzione* $p = \frac{a}{n}$. Nel nostro caso la proporzione è rappresentata dalla percentuale per cui risulta valida la coerenza tra giudizi umani e punteggi computati dagli algoritmi.

3.5.2 Intervalli di confidenza di una proporzione

Una volta valutato per quale proporzione p di pagine web del campione esiste una coerenza tra punteggio qualitativo umano e punteggio meccanicamente computato dagli algoritmi di link analysis, non possiamo concludere direttamente che tali algoritmi hanno successo con una probabilità pari a p. Vogliamo vedere quindi in che termini possiamo parlare di validità degli stessi presso la popolazione [12].

Quando si vuole inferire dalla proporzione p, ottenuta sul campione, una proporzione P, relativa alla totalità della popolazione, è necessario stabilire i *limiti fiduciari* della proporzione osservata nel campione studiato. I limiti fiduciari sono quei due valori entro i quali si può ritenere che stia, con il 95% (o 99%) di confidenza, il valore "vero" della proporzione P nella popolazione. Essi individuano quello che viene chiamato *intervallo di con-*

fidenza. Dire che la confidenza sta al 95 o 99% significa imporre che la validità di quanto inferito dal campione alla popolazione è certa per tali valori percentuali. Ricavato, cioè, l'intervallo di validità della proporzione p all'interno del campione, sarà lecito affermare che, con una probabilità del 95 o 99%, la proporzione in realtà valida (il valore "vero" di P), che per motivi di dimensioni non potremo mai conoscere, appartiene al suddetto intervallo. La stima che otterremo si chiamerà *stima per intervalli*.

3.5.3 Risultati numerici dell'inferenza

Una volta imposto il grado di confidenza α che riteniamo che il nostro risultato debba garantire, dovremo procedere a concretizzare i risultati numerici dell'inferenza statistica.

Richiami

Richiamiamo alcuni concetti di statistica per introdurre i ragionamenti e i calcoli riportati in seguito.

Nel calcolo delle probabilità, la *funzione di distribuzione* di una variabile aleatoria X è la funzione che associa a ciascun valore x assunto da X la probabilità dell'evento "la variabile aleatoria X assume valori minori o uguali ad x". Essa è descritta dalla relazione:

$$F_X(x) = \mathcal{P}[X \leq x]$$

La *funzione di densità di probabilità* di una variabile aleatoria X è definita come la derivata della funzione di distribuzione:

$$f_X(x) = \frac{dF_X(x)}{dx}$$

Pertanto, integrando $f_X(x)$ tra due punti, x_1 ed x_2, si ottiene la probabilità che la variabile aleatoria X sia compresa nell'intervallo $[x_1, x_2]$; da ciò segue che l'integrale esteso a tutto l'asse reale deve valere 1. Quando si impone di ottenere un risultato statistico con una confidenza pari a α (valore normalizzato $\in [0, 1]$), si intende ricavare il valore assunto dalla generica occorrenza $x*$ della variabile aleatoria X in corrispondenza a cui l'integrale $\int_{-\infty}^{x*} f_X(x)$ vale α.

Calcolo degli intervalli di confidenza

È noto [7] che la variabile aleatoria $X = \bar{p}$ che rappresenta la proporzione di successi ottenuti in un campione di n elementi, è modellata dalla legge

della distribuzione binomiale a media $\mu = p$ e varianza $\sigma^2 = p(1-p)$ (dove p è un generico valore assunto dalla variabile aleatoria $\bar{p}$). Troviamo che [7], per n sufficientemente grande (tale per cui $n \cdot p \geq 5$), la distribuzione si riconduce con ottima approssimazione a quella normale a media $\mu = p$ e varianza $\sigma^2 = \frac{p(1-p)}{n}$.

La funzione statistica di densità di probabilità della distribuzione normale $\mathcal{N}(\mu, \sigma^2)$ è:

$$f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \tag{3.2}$$

Per trovare il valore della variabile $\bar{p}$ in corrispondenza a cui l'integrale della distribuzione normale raggiunge il valore di confidenza impostato a priori sarebbe necessario innanzitutto integrare la funzione (3.2). In realtà, per calcolare le aree di probabilità di una distribuzione normale generale $\mathcal{N}(\mu, \sigma^2)$ si può introdurre, per ragioni di comodità, la variabile aleatoria Z, che rappresenta la trasformazione della variabile normale X in una variabile normale standardizzata. La trasformazione è data dalla relazione:

$$Z = \frac{(X - \mu)}{\sigma} \tag{3.3}$$

Ricorrere alla distribuzione standardizzata risulta estremamente più conveniente in termini di calcolo. Essa, infatti, permette di evitare il calcolo di un integrale non elementare e consente di ottenere il valore cercato semplicemente ricorrendo alla consultazione della tavola della distribuzione normale standardizzata.

Calcolare l'area entro cui l'integrale della densità di probabilità vale α all'interno della campana gaussiana, per ragioni di simmetria della curva standardizzata rispetto all'origine, equivale a calcolare l'area entro cui l'integrale della densità di probabilità vale $\frac{\alpha}{2}$ ristrettamente alla parte di gaussiana collocata nel primo quadrante (o nel secondo). Consultando la tabella riportata in appendice C, se desideriamo ottenere un risultato con una confidenza pari a α, dovremo individuare l'ascissa z in corrispondenza a cui l'integrale di $f_Z(z)$ vale $\frac{\alpha}{2}$. Usiamo la notazione $z_{\frac{\alpha}{2}}$ per indicare la suddetta ascissa. Sostituendo nella relazione (3.3), abbiamo:

$$z_{\frac{\alpha}{2}} = \frac{(\bar{p} - \mu)}{\sigma}$$

dove $\sigma = \pm\sqrt{\sigma^2}$. Ricordando che $\mu = p$ e $\sigma^2 = \frac{p(1-p)}{n}$, otteniamo, risovendo rispetto a $\bar{p}$:

$$\bar{p} = \mu + z_{\frac{\alpha}{2}} \cdot \sigma = \mu \pm z_{\frac{\alpha}{2}} \cdot \sqrt{\sigma^2} = p \pm z_{\frac{\alpha}{2}} \cdot \sqrt{\frac{p(1-p)}{n}} \tag{3.4}$$

che rappresenta l'intervallo di confidenza cercato.

Esempio applicativo

Ammettiamo di aver imposto un valore di confidenza pari al 95%, che indichiamo con $\alpha = 0.95$. Ammettiamo di aver estratto dalla popolazione in esame, costituita da 1000 pagine web restituite da una query ed ordinate per punteggio (di PageRank, HITS, SALSA o Indegree) un campione di 50 pagine con una procedura di campionamento sistematico. Supponiamo di aver rilevato, a seguito dell'analisi delle interviste di qualità, che le pagine per cui si registra una forma di coerenza tra giudizio umano e punteggio algoritmico siano $a = 40$. La proporzione empiricamente ottenuta $p = \frac{a}{n}$ è dell'80%, in termini normalizzati è $p = 0.8$.

Vogliamo ora definire l'intorno di 0.8 cui statisticamente possiamo ammettere la nostra proporzione appartenga, con una confidenza del 95%. Innanzitutto, dalla consultazione delle tavole della distribuzione normale standardizzata, ricaviamo il valore dell'ascissa $z_{\frac{\alpha}{2}}$, ossia troviamo che il valore che risolve l'equazione $f_Z(z*) = \frac{\alpha}{2} = 0.475$ è $z* = 1.96$. Dopodiché, utilizzando la formula (3.4), otteniamo che l'intervallo di confidenza di interesse è dato da:

$$\mu \pm z_{\frac{\alpha}{2}} \cdot \sigma = 0.8 \pm 1.96 \cdot \sqrt{\frac{0.8 \cdot (1 - 0.8)}{50}} = 0.8 \pm 0,11087$$

Cioè: $[0,68913; 0.91087]$. Ossia la proporzione pari all'80% nel campione si colloca nell'intervallo incluso tra il 68.9% e il 91.1% nella popolazione.

3.6 Estensione del risultato ottenuto sul campione all'intera popolazione

Riassumendo quanto fin qui trattato, possiamo schematizzare i passi che dovranno essere compiuti in concreto per trarre delle conclusioni dall'attività di analisi del campione svolta nella nostra ricerca.

Prima di tutto dovremo definire la dimensione del campione che intendiamo prelevare per effettuare le interviste di qualità. Dovremo, in seguito, impostare il grado di confidenza che riteniamo accettabile. Una volta che avremo terminato la raccolta delle interviste, dovremo provvedere a determinare la proporzione p con cui si registra empiricamente una coerenza tra giudizio umano raccolto e giudizio numerico calcolato dagli algoritmi di link analysis. Infine procederemo a ricavare, grazie alle formule sopra riportate

e grazie alla tavola della distribuzione normale standardizzata, l'intorno di p cui appartiene la proporzione P valida presso l'intera popolazione.

Esiste la possibilità di un approccio alternativo, che procede in modo invertito rispetto a quello appena descritto. Esso consiste nel definire a priori l'ampiezza che riteniamo di poter considerare accettabile per l'intervallo di appartenenza della proporzione P che risulterà valida all'interno della popolazione e, sulla base di tale valore, ricostruire la dimensione necessaria per il campione da estrarre e sul quale procedere poi a raccogliere le interviste qualitative.

Capitolo 4

Conclusioni

L'obiettivo del nostro progetto è quello di valutare l'effettiva efficacia di funzionamento degli algoritmi di link analysis. La letteratura scientifica che ha trattato l'argomento, fino a questo momento, non ha ancora fornito risultati sperimentali convincenti, che invece il progetto *Qualità* mira ad ottenere.

La presente relazione documenta parte del lavoro svolto all'interno del progetto. Innanzitutto l'implementazione del codice sorgente degli algoritmi di link analysis che si sono voluti testare (PageRank, HITS, SALSA e In-degree). Poi la formulazione delle euristiche per mezzo delle quali ottenere un campione rappresentativo di pagine web su cui poter raccogliere giudizi umani qualitativi. Infine le modalità con cui accertare statisticamente la validità globale dei risultati ottenuti sul campione. Quanto sopra è stato possibile soltanto grazie ad un precedente impegno nel progetto da parte di altri studenti dell'Università di Padova, consistito in un'attività di crawling del grafo del web e di implementazione di un'applicazione per la raccolta dellle interviste di qualità.

Il futuro del progetto consisterà nell'utilizzo del materiale realizzato: l'esecuzione del codice sorgente degli algoritmi permetterà di computare i punteggi algoritmici delle pagine prelevate secondo l'euristica di campionamento formalizzata; dopo aver ottenuto i risultati delle interviste di qualità e aver determinato il grado di coerenza tra punteggi numerici e giudizi umani, seguendo le indicazioni tecniche di inferenza statistica si potrà indurre la validità del risultato generale.

Appendice A

Alcune scelte di programmazione

Accanto alle specifiche scelte implementative effettuate per i vari algoritmi, ve ne sono alcune ricorrenti in tutti, in seguito descritte in modo particolareggiato.

- Utilizzo della classe `FileInputStream` per leggere, bit a bit, i file di stringhe binarie che memorizzano i grafi e sottografi su cui gli algoritmi operano.
- Utilizzo del metodo `read()` per la lettura da file su disco dei singoli caratteri costitutivi dei file stessi:
  - `read()` restituisce il valore decimale corrispondente alla codifica ASCII del carattere letto (la codifica ASCII di 0 è il decimale 48, la codifica ASCII di 1 è il decimale 49);
  - `read()` restituisce -1 se non ci sono più caratteri da leggere.
- Lettura degli id dei nodi sotto forma di stringhe s di 32 caratteri ASCII, poi convertite in interi con il metodo `(int) Long.parseLong(String s)`:

```
int t = inX.read();
char buffer[] = new char[32];
buffer[0] = (char) '0';
for (int i=1; i<32; i++){
        t = inX.read();
        buffer[i] = (char) t;}
String value = new String(buffer);
int numero = (int)Long.parseLong(value,2);
```

- Controllo del bit di 'genitorialità' per mezzo della valutazione del valore del decimale corrispondente alla codifica ASCII del carattere letto:

```
t = inX.read();
int m = (int) t;
```

  - se `m = 49` allora il nodo è genitore;
  - se `m = 48` allora il nodo è figlio.

Appendice B

Dettagli e istruzioni di utilizzo dei codici sorgenti

B.1 PageRank

- **Inizializzazioni**
  Sono necessari:
  - l'array di double `pageRank_prec` per il salvataggio dei punteggi di una generica iterazione, inizializzato a 0;
  - l'array di double `pageRank_succ` per il salvataggio dei punteggi dell'iterazione successiva, inizializzato a 0;
  - il valore del `damping factor,` impostabile a piacere;
  - il numero di nodi del grafo del web;
  - il valore di residuo che governa la convergenza;
  - l'array sovradimensionato, `id_figli`, di interi per il salvataggio, per ogni nodo genitore letto, di tutti i relativi figli.
- **Corpo dell'algoritmo, da iterare fino alla convergenza**

```
/**unico ciclo di lettura, fino alla fine del file*/
    while (t != -1){
    /*Ad ogni iterazione di questo ciclo si sta leggendo un genitore*/
                char buffer[] = new char[32];
                buffer[0] = (char) '0';
                for (int i=1; i<32; i++){
                        t = in.read();
                        buffer[i] =(char) t;
                }
                String value = new String(buffer);
                int numero = (int)Long.parseLong(value,2);
```

```
                /**il nodo padre scandito viene fissato in corr*/
                int corr = numero;

                int m = 0;
                t = in.read();
                int w = (int) t;

                /**finché sto leggendo figli*/
                while (t != -1 && w == 48){
                        buffer[0] = (char) t;
                        for (int i=1; i<32; i++){
                                t = in.read();
                                buffer[i] =(char) t;
                        }
                        value = new String(buffer);
                        numero = (int)Long.parseLong(value,2);

                        /**metto tutti i figli nell'array id_figli*/
                        id_figli[m] = numero;
                        m++;

                        t = in.read();
                        w = (int) t;
                }

                /**se il nodo scandito non è un nodo sink,
              * cioè se ha figli*/
                if(m>0) {
                        for (int j=0; j<m; j++){
                        /**do il contributo di punteggio di PageRank
                     * del nodo corrente ai nodi puntati dal
                     * nodo corrente*/
                                pageRank_corr[id_figli[j]] =
                        pageRank_corr[id_figli[j]] +
                        pageRank_prec[corr]/m;
                        }
                }

                /**altrimenti il nodo scandito è un nodo sink,
              * non ha figli*/
                else {
                        for (int j=0; j<CONT; j++){
                        /**do a tutti i nodi una frazione di punteggio
                     * di PageRank del nodo corrente*/
                                pageRank_corr[j] =
                        pageRank_corr[j] +
                        pageRank_prec[corr]/CONT;
                        }
                }
        /**si attribuisce a tutti i nodi il punteggio
        * di PageRank pesato secondo il damping factor*/
                for (int i=0; i<CONT; i++){
                        pageRank_corr[i] =
           pageRank_corr[i]*DAMPING_FACTOR +
                                (1 - DAMPING_FACTOR)/CONT;}
}
```

- **Controllo sulla convergenza**
  - calcolo della somma delle differenze al quadrato tra i punteggi di PageRank all'iterazione corrente e quelli all'iterazione precedente;
  - verifica se il residuo ottenuto è minore della costante di tolleranza, in caso affermativo le iterazioni si arrestano, altrimenti segue un nuovo ciclo del corpo dell'algoritmo.

Istruzioni per l'utilizzo del codice

Per la compilazione e l'esecuzione della classe `PageRank` è sufficiente:

- inserire il path del file di output (riga 22);
- inserire il numero totale di nodi del grafo del web, costante nota dopo il crawling (riga 31);
- impostare il valore di tolleranza per il calcolo della convergenza (riga 35);
- inserire il path del file di input (riga 64).

B.2 HITS

B.2.1 Classe GrafoHITS

- **Inizializzazioni**
  Sono necessari tre array di elementi di tipo int:
  - `array1`: per mantenere gli id dei nodi ottenuti in risposta alla query, con bit più significativo settato a 1;
  - `array0`: per mantenere gli id dei nodi ottenuti in risposta alla query, con bit più significativo settato a 0;
  - `cont`: per conteggiare, nella cella di indice `i`, il numero di genitori già inseriti del nodo con id `i`.
- **Ciclo di lettura del file di id dei nodi restituiti dalla query**
  Ogni nodo letto viene inserito nell' `array1` e nell' `array0`, rispettivamente con bit più significativo impostato a 1 e a 0 (come genitore e come figlio).

- **Ciclo di lettura del file di id dei nodi del grafo del web**
  Attraverso una sola lettura di tutti gli id del grafo del web avviene la ricostruzione del sottografo utilizzato da HITS.
  Il codice è particolarmente articolato, per facilitarne la comprensione segue una schematica ricostruzione:

  1. lettura di un id x.
  2. controllo: il nodo scandito x è presente tra i 200 nodi restituiti dalla query?
     Sì, punto 3. No, punto 9.
  3. scrittura dell'id letto x in output con bit più significativo a 1.
  4. lettura id successivo y.
  5. controllo: il nodo scandito y è un figlio?
     Sì, punto 6. No, punto 2.
  6. scrittura dell'id letto y in output con bit più significativo a 0.
  7. controllo: il nodo scandito y è presente tra i 200 nodi restituiti dalla query?
     Sì, punto . No, punto 8.
  8. salvataggio del nodo y in array1, dopo i 200 nodi iniziali.
  9. lettura id successivo z.
  10. controllo: il nodo scandito z è un figlio?
      Sì, punto 11. No, punto 2.
  11. controllo: il nodo scandito z è presente tra i 200 nodi restituiti dalla query e ha meno di 50 padri?
      Sì, punto 12. No, punto 16.
  12. incremento del numero padri di z.
  13. controllo: z è già stato scritto in precedenza in output?
      Sì, punto 14. No, punto 15.
  14. scrittura dell'id x con bit più significativo a 1.
  15. scrittura dell'id z con bit più significativo a 0.
  16. controllo: ancora nodi da leggere nel file del grafo del web?
      Sì, punto 1. No, FINE.

- **Inserzione dei nodi figli aggiuntivi anche come genitori**
  Dal momento che la convenzione adottata relativamente al grafo del web prevede che tutti i nodi inseriti compaiano come genitori, si deve provvedere a inserire con bit più significativo impostato a 1 anche i nodi che finora sono stati inseriti solo come figli.

Istruzioni per l'utilizzo del codice

Per la compilazione e l'esecuzione della classe `GrafoHITS` è sufficiente:

- inserire il numero di pagine considerate tra quelle restituite dalla query (riga 20);
- inserire il path del file di output (riga 26);
- inserire il path del file di input, file degli id dei nodi restituiti dalla query (riga 44);
- inserire il path del file di input, file degli id dei nodi del grafo del web (riga 79).

B.2.2 Classe IdHITS

- **Inizializzazioni**
  È necessario disporre di un array di interi, `arr,` in cui inserire i vecchi id; l'indice `i` della cella dell'array contenente il vecchio id di un nodo costituisce il nuovo id dello stesso.
- **Primo ciclo di lettura del file del sottografo con i vecchi id**
  Ogni nodo padre letto viene salvato in una cella dell'array `arr.`
- **Secondo ciclo di lettura del file del sottografo con i vecchi id**
  Questa seconda lettura permette di ricostruire il file del sottografo sostituendo ai vecchi id, a 32 bit, i nuovi id, a 14 bit, associati loro univocamente.

Istruzioni per l'utilizzo del codice

Per la compilazione e l'esecuzione della classe `IdHITS` è sufficiente:

- inserire il numero totale di nodi del sottografo prodotto dalla classe `GrafoHITS` (riga 18);
- inserire il path del file di input (righe 23 e 72);
- inserire il path del file di output (riga 64).

B.2.3 Classe HITS

- **Inizializzazioni**
  È necessario disporre di due array di tipo double:
  - `authority`: per memorizzare i punteggi di authority di tutti i nodi, inizializzato a 0;
  - `hub`: per memorizzare i punteggi di hub di tutti i nodi, inizializzato a 1.
- **Corpo dell'algoritmo**

```
while (n_i < num_iterazioni){
        n_i++;

        /**inizializzazione ad ogni ciclo dei punteggi di authority
      * a 0*/
        for (int i=0; i<CONT; i++)
                authority[i] = 0;

        /**prima apertura del file di input: sottografo completo
      * con id a 14 bit*/
        FileInputStream inX=new FileInputStream("C:\\Documents "+
          "and Settings\\SuperUser\\Desktop\\Java\\" +
          "ProgrammiVariJava\\src\\nomefile1.txt");

        int t = inX.read();

        /**lettura del file di input per il calcolo dei punteggi
      * di authority*/
        while (t != -1){
                /**Ad ogni iterazione di questo ciclo si sta
          * leggendo un genitore*/
                char buffer[] = new char[14];
                buffer[0] = (char) '0';
                for (int i=1; i<14; i++){
                        t = inX.read();
                        buffer[i] = (char) t;
                }
                String value = new String(buffer);
                int numero = (int)Long.parseLong(value,2);
                /**corr mantiene il padre scandito, con bit
          * più significativo già impostato a 0*/
                int corr = numero;

                t = inX.read();
                int m = (int) t;

                /**finché si stanno scandendo figli*/
                while (t != -1 && m == 48){
                        buffer[0] = (char) t;
                        for (int i=1; i<14; i++){
                                t = inX.read();
                                buffer[i] = (char) t;
```

```
                        }
                        value = new String(buffer);
                        numero = (int)Long.parseLong(value,2);

                        /**aggiorna il punteggio di authority dei
                * figli del nodo scandito*/
                  authority[numero] = authority[numero] + hub[corr];

                        t = inX.read();
                        m = (int) t;

                  }
    }
}
```

- **Normalizzazione dei punteggi di hub e authority**

```
/**normalizza il vettore di authority*/
double totale = 0;
for (int i=0; i<CONT; i++){
        totale = totale + authority[i]*authority[i];
}
totale = Math.sqrt(totale);
for (int i=0; i<CONT; i++){
        authority[i] = authority[i]/totale;
}

/**normalizza il vettore di hub*/
totale = 0;
for (int i=0; i<CONT; i++){
        totale = totale + hub[i]*hub[i];
}
totale = Math.sqrt(totale);
        for (int i=0; i<CONT; i++){
                hub[i] = hub[i]/totale;
}
```

Istruzioni per l'utilizzo del codice

Per la compilazione e l'esecuzione della classe `HITS` è sufficiente:

- inserire il numero totale di nodi del sottografo prodotto dalla classe `GrafoHITS` (riga 18);
- inserire il numero di iterazioni che si vuole che l'algoritmo effettui (riga 20);
- inserire il path del file di output (riga 26);
- inserire il path del file di input (righe 55 e 106).

B.3 SALSA

- **Inizializzazioni**
  Sono necessari:
  - l'array di double `auth_s` per il salvataggio dei punteggi di authority, inizializzato a 0;
  - l'array di double `hub_s` per il salvataggio dei punteggi di hub, inizializzato a 0;
  - il numero totale di nodi del sottografo prodotto dalla classe `GrafoHITS`.
- **Corpo dell'algoritmo**

```
int t = in.read();
int m = (int) t;

while (t != -1){
/* Ad ogni iterazione di ciclo si sta leggendo un genitore*/
        char buffer[] = new char[14];
        buffer[0] = (char) '0';
        for (int i=1; i<14; i++){
                t = in.read();
                buffer[i] = (char) t;
        }
        String value = new String(buffer);
        int numero = (int)Long.parseLong(value,2);
        /**si fissa il nodo scandito, che è un nodo padre,
      * in corr*/
        int corr = numero;

        t = in.read();
        m = (int) t;

        /**finché sto leggendo figli*/
        while (t != -1 && m == 48){
                buffer[0] = (char) t;
                for (int i=1; i<14; i++){
                        t = in.read();
                        buffer[i] = (char) t;
                }
                value = new String(buffer);
                numero = (int)Long.parseLong(value,2);
                /**per ogni figlio che incontro incremento di
        *un'unità il punteggio di authority del padre*/
                auth_s[corr]++;
                /**incremento di un'unità il punteggio di hub
        *di ciascun figlio*/
                hub_s[numero]++;

                t = in.read();
                m = (int) t;
        }

}
```

- **Normalizzazione dei punteggi di hub e authority**

```
/**normalizzazione del vettore di authority*/
double totale = 0;
for (int i=0; i<CONT; i++){
        totale = totale + auth_s[i];
}
for (int i=0; i<CONT; i++){
        auth_s[i] = auth_s[i]/totale;
}

/**normalizzazione del vettore di hub*/
totale = 0;
for (int i=0; i<CONT; i++){
        totale = totale + hub_s[i];
}
for (int i=0; i<CONT; i++){
        hub_s[i] = hub_s[i]/totale;
}
```

Istruzioni per l'utilizzo del codice

Per la compilazione e l'esecuzione della classe `SALSA` è sufficiente:

- inserire il numero totale di nodi del sottografo prodotto dalla classe `GrafoHITS` (riga 21);
- inserire il path del file di output (riga 37);
- inserire il path del file di input (riga 45).

B.3.1 SALSAbis

- **Inizializzazioni**
  Sono necessari:
  - l'array di double `auth_s` per il salvataggio dei punteggi di authority, inizializzato a 0;
  - l'array di double `hub_s` per il salvataggio dei punteggi di hub, inizializzato a 0;
  - l'array di int `id_figli` per il salvataggio, per ogni nodo genitore letto, di tutti i relativi figli;
  - il numero totale di nodi del sottografo prodotto dalla classe `GrafoHITS`.
- **Primo ciclo di lettura, calcolo dei punteggi di authority**

```
int t = in.read();
int m = (int) t;

while (t != -1){
        /* Ad ogni iterazione di ciclo si sta leggendo un genitore*/
    char buffer[] = new char[14];
        buffer[0] = (char) '0';
        for (int i=1; i<14; i++){
                t = in.read();
                buffer[i] = (char) t;
        }
        String value = new String(buffer);
    int numero = (int)Long.parseLong(value,2);
        /**si fissa il nodo scandito, che è un nodo padre, in corr*/

        t = in.read();
        m = (int) t;
    /**contatore del numero di figli del nodo scandito*/
        int f=0;

        /**finché sto leggendo figli*/
        while (t != -1 && m == 48){
                buffer[0] = (char) t;
                for (int i=1; i<14; i++){
                        t = in.read();
                        buffer[i] =(char) t;
                }
                value = new String(buffer);
                numero = (int)Long.parseLong(value,2);

        /**contatore del numero di link in ingresso
          *al nodo figlio scandito*/
                peso_per_Outdegree[numero]++;

                /**salvo tutti i figli del nodo scandito in un array
          *di supporto*/
                id_figli[f] = numero;
                f++;

                t = in.read();
                m = (int) t;
        }

    /**calcolo il peso dei link uscenti dal nodo padre scandito*/
        double weighted_outdegree = 1./f;

        if (f > 0){     /**attribuisco a tutti i figli del nodo
                *padre scandito il peso calcolato*/
                for(int j=0; j<f; j++)
                auth_s[id_figli[j]]=auth_s[id_figli[j]]+
                                weighted_outdegree;
        }
}
```

Analogamente viene fatto per i punteggi di hub.

- **Normalizzazione dei punteggi di authority (e di hub)**

```
/**normalizzazione del vettore di authority*/
double totale = 0;
for (int i=0; i<CONT; i++){
        totale = totale + auth_s[i];
}
for (int i=0; i<CONT; i++){
        auth_s[i] = auth_s[i]/totale;
}
```

Istruzioni per l'utilizzo del codice

Per la compilazione e l'esecuzione della classe `SALSAbis` è sufficiente:

- inserire il numero totale di nodi del sottografo prodotto dalla classe `GrafoHITS` (riga 21);
- inserire il path del file di output (riga 37);
- inserire il path del file di input (righe 47 e 101).

B.4 Indegree

- **Inizializzazioni**
  È necessario disporre di un array di interi, `indegree,` inizializzato a 0, in cui salvare i punteggi di Indegree dei nodi: nella cella di indice `i` di `indegree` è presente il numero di link entranti nel nodo di id `i` del grafo del web.

- **Corpo dell'algoritmo**

```
int t = in.read();
int m = (int) t;

while (t != -1){

/*Ad ogni iterazione di questo ciclo si sta leggendo un genitore*/
        char buffer[] = new char[32];
        buffer[0] = (char) '0';
        for (int i=1; i<32; i++){
                t = in.read();
                buffer[i] = (char) t;
        }
        String value = new String(buffer);
        int numero = Integer.parseInt(value,2);
        t = in.read();
        m = (int) t;

        /**finché si stanno scandendo figli*/
```

```
            while (t != -1 && m == 48){
                    buffer[0] = (char) t;
                    for (int i=1; i<32; i++){
                            t = in.read();
                            buffer[i] = (char) t;
                    }
                    value = new String(buffer);
                /**id del figlio scandito*/
                    numero = Integer.parseInt(value,2);
                    /**incremento di una unità del punteggio
              * di indegree del figlio scandito*/
                    indegree[numero]++;
                    t = in.read();
                    m = (int) t;
                    }

}
```

Istruzioni per l'utilizzo del codice

Per la compilazione e l'esecuzione della classe `Indegree` è sufficiente:

- inserire il numero totale di nodi del grafo del web (riga 18);
- inserire il path del file di input (riga 27);
- inserire il path del file di output (riga 34).

Appendice C

Tavola della distribuzione normale standardizzata

Viene di seguito riportata la tavola della distribuzione normale standardizzata, che sarà necessario consultare una volta terminata la raccolta delle interviste qualitative allo scopo di definire l'intervallo di confidenza per la validità della nostra ipotesi di ricerca all'interno della popolazione.

z	0.00	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
0.0	.00000	.00399	.00792	.01197	.01595	.01994	.02392	.02790	.03188	.03586
0.1	.03983	.04380	.04776	.05172	.05567	.05962	.06356	.06749	.07142	.07535
0.2	.07926	.08317	.08706	.09095	.09483	.09871	.10257	.10642	.11026	.11409
0.3	.11791	.12172	.12552	.12930	.13307	.13683	.14058	.14431	.14803	.15173
0.4	.15542	.15910	.16276	.16640	.17003	.17364	.17724	.18082	.18439	.18793
0.5	.19146	.19497	.19847	.20194	.20540	.20884	.21226	.21566	.21904	.22240
0.6	.22575	.22907	.23237	.23565	.23891	.24215	.24537	.24857	.25175	.25490
0.7	.25804	.26115	.26424	.26730	.27035	.27337	.27637	.27935	.28230	.28524
0.8	.28814	.29103	.29389	.29673	.29955	.30234	.30511	.30785	.31057	.31327
0.9	.31594	.31859	.32121	.32381	.32639	.32794	.33147	.33398	.33646	.33891
1.0	.34134	.34375	.34614	.34849	.35083	.35314	.35543	.35769	.35993	.36214
1.1	.36433	.36650	.36864	.37076	.37286	.37493	.37698	.37900	.38100	.38298
1.2	.38493	.38686	.38877	.39065	.39251	.39435	.39617	.39796	.39973	.40147
1.3	.40320	.40490	.40658	.40824	.40988	.41149	.41309	.41466	.41621	.41774
1.4	.41924	.42073	.42220	.42364	.42507	.42647	.42786	.42922	.43056	.43189
1.5	.43319	.43448	.43574	.43699	.43822	.43943	.44062	.44179	.44295	.44408
1.6	.44520	.44630	.44738	.44845	.44950	.45053	.45154	.45254	.45352	.45449
1.7	.45543	.45637	.45728	.45818	.45907	.45994	.46080	.46164	.46246	.46327
1.8	.46407	.46485	.46562	.46637	.46712	.46784	.46856	.46926	.46995	.47062
1.9	.47128	.47193	.47257	.47320	.47381	.47441	.47500	.47558	.47615	.47670
2.0	.47725	.47778	.47831	.47882	.47932	.47982	.48030	.48077	.48124	.48169
2.1	.48214	.48257	.48300	.48341	.48382	.48422	.48461	.48500	.48537	.48574
2.2	.48610	.48645	.48679	.48713	.48745	.48778	.48809	.48840	.48870	.48899
2.3	.48928	.48956	.48983	.49010	.49036	.49061	.49086	.49111	.49134	.49158
2.4	.49180	.49202	.49224	.49245	.49266	.49286	.49305	.49324	.49343	.49361
2.5	.49379	.49396	.49413	.49430	.49446	.49461	.49477	.49492	.49506	.49520
2.6	.49534	.49547	.49560	.49573	.49585	.49598	.49609	.49621	.49632	.49643
2.7	.49653	.49664	.49674	.49683	.49693	.49702	.49711	.49720	.49728	.49736
2.8	.49745	.49752	.49760	.49767	.49774	.49781	.49788	.49795	.49801	.49807
2.9	.49813	.49819	.49825	.49831	.49836	.49841	.49846	.49851	.49856	.49861
3.0	.49865	.49869	.49874	.49878	.49882	.49886	.49889	.49893	.49897	.49900
3.1	.49903	.49906	.49910	.49913	.49916	.49918	.49921	.49924	.49926	.49929
3.2	.49931	.49934	.49936	.49938	.49940	.49942	.49944	.49946	.49948	.49950
3.3	.49952	.49953	.49955	.49957	.49958	.49960	.49961	.49962	.49964	.49965
3.4	.49966	.49968	.49969	.49970	.49971	.49972	.49973	.49974	.49975	.49976
3.5	.49977	.49978	.49978	.49979	.49980	.49981	.49981	.49982	.49983	.49983
3.6	.49984	.49985	.49985	.49986	.49986	.49987	.49987	.49988	.49988	.49989
3.7	.49989	.49990	.49990	.49990	.49991	.49991	.49991	.49992	.49992	.49992
3.8	.49993	.49993	.49993	.49994	.49994	.49994	.49994	.49995	.49995	.49995
3.9	.49995	.49995	.49995	.49996	.49996	.49996	.49996	.49996	.49997	.49997

Nota per la lettura: per trovare l'ascissa $x*$ in corrispondenza a cui l'integrale della curva tra 0 e $x*$ vale (vedi esempio applicativo proposto nel paragrafo 3.5.3) $\frac{\alpha}{2} = 0.475$, si cerca tale valore nella tavola e si legge nella corrispondente riga la parte intera e il primo valore decimale, e nella corrispondente colonna il secondo valore decimale dell'ascissa $x*$. In questo caso rispettivamente 1.9 e 0.06, quindi $x* = 1.96$.

Tavola della distribuzione normale standardizzata (tratta da [6]).

Bibliografia

[1] L. Terveen B. Amento and W. Hill. Does "authority" mean quality? Predicting expert quality ratings of web documents. In *Proc. of WWW Conference*, pages 296–303, 1998.

[2] S. Brin and L. Page. The anatomy of a large scale hypertextual web search engine. In *Proc. of WWW Conference*, 1998.

[3] T. Haveliwala. Efficient computation of PageRank. Technical report, Stanford, 1999.

[4] J. M. Kleinberg. Authoritative sources in a hyperlinked enviroment. *Journal of the ACM*, 46(5), 1999.

[5] R. Lempel and S. Moran. SALSA: The stochastic approach for link-structure analysis. *ACM Transactions on Information Systems*, 19(2):131–160, 2001.

[6] G. Manzone and S. Longo. *Probabilità e statistica*. Tramontana, 1992.

[7] A. Onofri. Dal campione alla popolazione: introduzione al test d'ipotesi. `http://www.unipg.it/onofri/RTutorial/index.html`.

[8] C. Castillo R. Baeza-Yates and E.N. Efthimiadis. Characterization of national web domains. *ACM Transactions on Internet Technology*, 7(9), 2007.

[9] H. A. Simon. On a class of skew distribution functions. *Biometrika*, (42):425–440, 1955.

[10] N. Craswell T. Upstill and D. Hawking. Predicting fame and fortune: PageRank or Indegree? In *Proc. of ACM SIGIR*, 2003.

[11] M. Trovato and R. Manfredi. *Calcolo delle probabilità e statistica inferenziale*. Ghisetti-Corvi, 1997.

[12] A. Zanella. *Elementi di teoria del campionamento da popolazioni finite.* CLEUP, 1974.