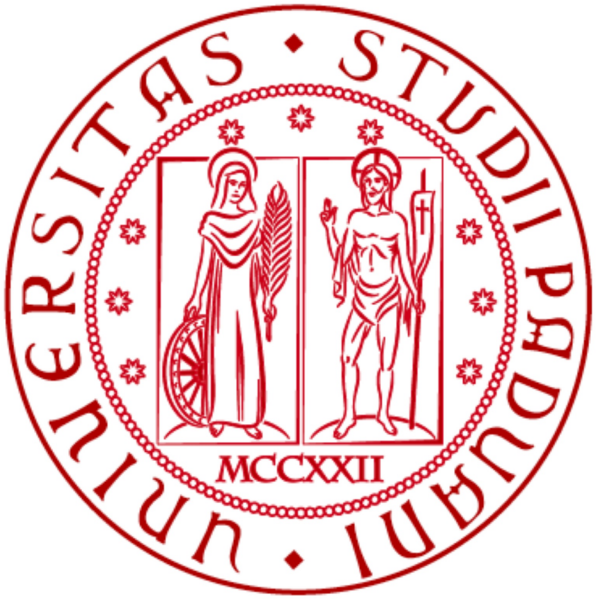

11 giugno 2015

Documento finale per la prova d'esame di Sistemi Informativi a.a. 2014/2015

Giovanni Venturelli
giovanni.venturelli@studenti.unipd.it
David Tessaro
david.tessaro@studenti.unipd.it
Aurelien Keleko Teguede
aurelien.kelekoteguede@studenti.unipd.it

Indice

Sommario

In questo documento si vuole presentare il lavoro svolto durante le lezioni di laboratorio dal gruppo 14. L'articolo si divide in due parti principali: nella prima parte vengono illustrate le metodologie di lavoro del gruppo presentando l'organizzazione del lavoro e gli approcci scelti per lo sviluppo degli algoritmi. È spiegato il concetto di peso di parola e di documento e ne è specificato l'utilizzo. L'indicizzazione sfrutta la legge di Zipf per assegnare un peso alle parole. Tale peso, tuttavia, non identificain modo sufficientemente esaustivo i documenti, il gruppo ha quindi deciso di calmierarlo introducendo due indici che fanno riferimento alla frequenza delle parole in tutti i documenti. Viene presentato un semplice algoritmo di reperimento che tuttavia si è rivelato abbastanza efficace e utilizzabile come baseline per un algoritmo di retroazione (nello specifico verranno implementati un algoritmo di Relevance Feedback e un algoritmo di Blind Feedback).
Un approccio più indirizzato al modello probabilistico viene utilizzato, invece, per lo sviluppo di algoritmi di PageRank, HITS e LSA. Nella seconda parte del documento sono, invece, presentati i risultati ottenuti applicando tali algoritmi alle query fornite a lezione.

1 Introduzione

Lo scopo di questo progetto è l'avvicinamento ai sistemi di reperimento con l'intento di capirne i comportamenti e le metodologie di funzionamento. Useremo un approccio quanto più cauto per la stesura degli algoritmi e vicino ai riferimenti dateci a lezione cercando di essere più chiari possibili.
La presenza nel gruppo di due studenti di informatica ha permesso di implementare un sistema di IR dalla base, per analizzare a un livello più approfondito lo sviluppo degli algoritmi e comprenderne più chiaramente ogni singolo comportamento, il gruppo si è quindi indirizzato verso l'utilizzo di librerie più primitive di algebra lineare a discapito di ambienti più sviluppati e collaudati ponendo l'apprendimento in posizione prioritaria rispetto alla rapidità e all'efficienza nei risultati.
Tra le possibili scelte il gruppo si è indirizzato verso il modello vettoriale che offre una più semplice lettura da parte umana e un controllo più agile nelle varie fasi di sviluppo, in seguito si è tentato di integrarlo con il modello Probabilistico per aumentarne il livello di precisione andando cosi ad aumentarne la complessità.
Si cercherà di ottenere dei risultati apprezzabili o, qualora questo non si verificasse, almeno sufficienti nell'implementazione degli algoritmi implementati dal gruppo, contenendo i tempi di elaborazione in livelli di complessità comunque accettabili.

2 Metodologia

2.1 Approccio

Composizione del gruppo
Il gruppo è composto da due studenti di informatica e uno studente di statistica. Questo ha permesso di avere punti di vista differenziati e più ricchi di quanto sarebbe stato nel caso di un gruppo più omogeneo.

Strumenti di sviluppo
Il gruppo ha scelto di utilizzare il linguaggio di programmazione c++ perché offre numerose librerie dedicate al calcolo matriciale e all'information retrieval. Inoltre la conoscenza della tecnologia ha favorito questa scelta.
Nello specifico sono state utilizzate le seguenti librerie:

- **Qt** (con relativo ambiente di sviluppo QtCreator) che offre numerose classi per la gestione dei flussi di dati e di stringhe;
- **Armadillo** e **OpenBLAS** che forniscono supporto per il calcolo matriciale.

Affrontare direttamente la costruzione completa degli algoritmi senza ricorrere al supporto di librerie dedicate come CLucene ha dato al gruppo la possibilità di approfondire in modo più dettagliato la problematica della progettazione dei sistemi IR.
L'ordinamento è stato eseguito tramite utilizzo del Quicksort motivato dal ottima efficienza di quest'ultimo, supponendo che in numerosi casi l'ordinamento non sarà mai totale ma limitato a pochi valori rispetto i totali.

Approccio incrementale Ogni laboratorio è stato affrontato in primo luogo descrivendo una bozza di algoritmo durante le ore di lezione. L'algoritmo è stato quindi rianalizzato in un secondo momento e reso più coerente con gli argomenti studiati durante le lezioni del martedì, per essere, quindi, implementato in c++.

Un approccio di tipo incrementale ha permesso che ogni settimana, a fronte dei laboratori proposti, il gruppo tornasse a rivalutare l'efficacia degli algoritmi precedenti, per correggere eventuali discrepanze alla luce delle nuove esigenze, soprattutto in riferimento alla complessità computazionale. Sfruttare i cicli già esistenti si è rivelato fondamentale per ottimizzare il lavoro.

Peso: un concetto chiave Il concetto di peso gode di una rilevanza fondamentale per gli esercizi svolti dal gruppo.
L'idea di base è che il peso di ogni singola parola influenzi il peso del documento all'interno di ogni query. Maggiore sarà il peso di una parola, maggiore sarà la rilevanza del documento nelle query legate a quella parola.

2.2 Indicizzazione

Per effettuare l'indicizzazione è stato scelto come valore di partenza l'indice di Zipf, utilizzandone i risultati come pesi delle singole parole. Questo indice, tuttavia, è stato ritenuto insufficiente per una corretta valutazione del peso delle parole all'interno di ogni documento della collezione, dando dei risultati che, a giudizio degli autori, si sono rivelati inadeguati.
Il calcolo del peso delle parole chiave nei documenti si è infatti mostrato carente quando è stato utilizzato, successivamente, per svolgere query di ricerca nella collezione, restituendo livelli di precisione del tutto scoraggianti (valutazione della precisione media con il primo algoritmo di reperimento di 1.205). Sono stati quindi pensati tre indici il cui scopo è correggere il peso delle parole.
Il primo indice (v-index) pesa i risultati dell'applicazione della legge di Zipf alla collezione in esame con il numero totale di documenti che citano tale parola in tutta la collezione.
È stato, infatti, valutato che una parola che compare in un numero maggiore di documenti possa essere una parola comune in riferimento agli argomenti trattati dai documenti della collezione (es. computer, computation, program, algorithm...). In questo caso è stato ritenuto corretto che la parola perdesse di rilevanza ai fini delle query.

Per calcolare il v-index si applichi la seguente formula

K=costante
r= frequenza della parola di cui si vuole ottenere il peso
p=K/r → Zipf's index
t = numero di documenti in cui compare la parola
V = p/t → v-index

Il secondo indice utilizzato (kt-index) pondera il v-index moltiplicandolo per il logaritmo del rapporto tra la frequenza della parola nel documento e la somma delle frequenze della parola stessa in tutti i documenti della collezione.
È stato, infatti, valutato che una parola con una frequenza elevata in molti documenti possa essere una parola di uso comune all'interno dei documenti scientifici (es. theory, theorem, work...). Anche in questo caso è stato ritenuto corretto che la parola perdesse di rilevanza ai fini delle query.
Si è scelto di applicare il logaritmo al rapporto per non sbilanciare troppo il peso totale della parola.

nfd = frequenza parola nel documento in esame
nfc = frequenza parola nella collezione
KT =V*log(nfd/nfc)

Per completare il calcolo del peso sono state prese in considerazione le citazioni del documento. È stato valutato che dovesse essere maggiorato il peso di una parola all'interno di un documento citato da documenti che contengono

la stessa parola. Questo significa, infatti, che il documento è particolarmente autorevole per le query che contengono il termine chiave. Se un documento ha un'elevata frequenza della parola A e una frequenza sempre elevata, ma comunque minore, della parola B, se viene citato da molti documenti che contengono la parola B e da nessun documento che contiene la parola A appare ragionevole ritenere che la parola B debba avere un peso maggiore rispetto alla parola A.

w= numero di documenti che contengono la parola e citano il documento in esame
p=KT*log(w) → peso finale

Algoritmo
Variabili e contenitori
parole[] = array di oggetti che rappresentano le parole chiave
parole[].key = stringa che identifica la parola chiave univocamente
parola[].frequenze[]=array di interi che identificano l'id dei documenti a cui corrisponde la frequenza di parola
Coll[]=collezione, è un array di oggetti che rappresentano i documenti.
Coll[x].pesoparola[j] = mappa che per ogni documento x associa una stringa j che identifica la parola chiave con un valore che indica il peso della parola all'interno del documento.

```
Indicizza (Coll, parole)
y=Coll.size
forall the parole as parola do
    q=0 //frequenza massima della parola nella collezione
    nfc=0 //somma delle frequenze della parola nella collezione
    parola.t=0 //numero di documenti che contengono la parola
    for i=1 to y do
        if parola.frequenze[i]>0 then
            parola.t++
            nfc+=parola.frequenze[i]
            if parola.frequenze[i]>q then
                q=parola.frequenze[i]
            end
        end
    end
    K=q+1
    for i=1 to y do
        if parola.frequenze[i]>0 then
            r=K-parola.frequenze[i]
            Coll[i].pesoparola[parola.key]=K/r //Zipf
            Coll[i].pesoparola[parola.key]*=1/t //v-index
            kt=log(parola.frequenze[i]/nfc;
            Coll[i].pesoparola[parola.key]*=1/t*kt) //kt-index
        end
    end
end
w=0;
forall the parole as parola do
    for i=1 to y do
        if parola.frequenze[i]>0 then
            forall the Coll[i].citations as cit do
                //scorre i documenti che citano il doc
                if cit.contains(parola.key) then
                    w++
                end
            end
        end
        Coll[i].pesoparola[parola.key]*=log(w)
    end
end
```

2.3 Reperimento

Il calcolo dei pesi di ogni documento relativamente alle query effettuate ci si è appoggiati a uno schema di pesatura TFIDF.
I passi principali dell'algoritmo utilizzato sono i seguenti
I
Si pongono in OR le parole chiave della query, e si cercano tutti i documenti che contengono almeno una parola della query
II
Per ognuno dei documenti ricavati si sommano i pesi delle parole della query presenti tra i suoi descrittori
ad ogni match, si aggiunge al peso del documento una costante k maggiore del prodotto del peso massimo delle keyword per il numero di parole nella query in modo da dare la precedenza ai documenti contenenti un più alto numero di parole della query.
III
Si ordina la lista di documenti sul peso così ottenuto
L'algoritmo è computazionalmente poco complesso e ha garantito una precisione abbastanza buona.
Sono state svolte più run per valutare l'opportunità di inserire il punto II.b all'interno dell'algoritmo. Tale passaggio ha notevolmente incrementato i livelli di precisione del reperimento.
Algoritmo
Date la query y e la collezione di documenti x
data y(i) la iesima keyword della query
dato x(i) l'iesimo documento della collezione

Peso(p, y.i) // p= indice documento
cerca la parola y.i e il corrispondente peso in p

```
QueryDoc(y, x)
array a[1, x.size];
array b[1, x.size];
array c[1,x.size];
riempi b di 0;
q=((peso massimo tra le parole della query)*(numero di parole nella
query))+qt //con qt>0;
for j = 1 to x.size do
    array k=parole chiave contenute in x.j
    for i = 1 to y.size do
        if k contiene y.i then
            c[j]++;
            a[j]=x.j
            b[j]=b[j]+Peso(a[j], y.i)+q
        end
    end
end
ordina b e a
for z= 0 to b.size AND not b[z] == 0 do
    c=new coppia
    c.doc=a[z]
    c.peso= b[z]
    result[z]=c
end
return result
```

2.4 Relevance Feedback

Visti i risultati positivi ottenuti attraverso l'algoritmo di reperimento non è stato considerato necessario effettuarvi interventi correttivi di natura incrementale ed è stato mantenuto come baseline per lo sviluppo dell'algoritmo di Relevance Feedback.
Ai risultati forniti dall'algoritmo di reperimento è stato applicato un algoritmo denominato Expansion. Lo scopo di Expansion è di aggiungere alla query iniziale le parole dal peso maggiore presenti nei documenti più rilevanti e richiamare nuovamente l'algoritmo di reperimento sulla nuova query così ottenuta.
Come richiesto sono state eseguite run sia partendo da un explicit feedback, sia partendo da un blind feedback.
Al fine di non aggravare ulteriormente la complessità computazionale è stato scelto di considerare solo il RF positivo.
Algoritmi
I documenti considerati rilevanti per svolgere la retroazione tramite blind feedback sono stati quelli che hanno ottenuto un peso maggiore attraverso la prima applicazione dell'algoritmo di reperimento.
Date la query y e la collezione di documenti x
data y(i) la iesima keyword della query
dato x(i) l'iesimo documento della collezione

I documenti considerati rilevanti per svolgere la retroazione tramite relevance feedback sono stati i primi 50 individuati nel documento qrels-originale.txt.
L'algoritmo RelevanceExpansion è del tutto simile a BlindExpansion tranne che, appunto per i documenti di partenza

In entrambi i casi possono essere svolte più run cambiando il numero di parole scelte per incrementare la query, il numero di documenti considerati rilevanti e il numero di iterazioni di espansione.

```
BlindExpansion(y,x)
res=QueryDoc(y, x) //array contenente tutti i documenti ritornati dalla
query ordinati
for i 1 to 50 do
 | resexp[i]=res[i]
end
ord=map //mappa destinata a contenere le parole con i pesi più grandi
newquery=y
for j 1 to resexp.size do
 | doc=resexp[i].doc
 | forall the doc.pesoparola as parola do
 | | if not ord.contains(parola.first) OR ord.[parola.first]¡parola.second
 | | then
 | | | ord[parola.first]=parola.second
 | | end
 | end
end
for j 1 to 5 and not ord.isEmpty do
 | s=chiave_con_valore_massimo(ord)
 | newquery.append((ord.pop(s)).first)
end
res=res+c*QueryDoc(newquery, x) //c≤1
ordina(res)
print res
```

```
RelevanceExpansion(y,x)
res=getResult(qrels-originale.txt) //array contenente tutti i documenti
ritornati dalla query ordinati
for i 1 to 50 do
 | resexp[i]=res[i]
end
ord=map //mappa destinata a contenere le parole con i pesi più grandi
newquery=y
for j 1 to resexp.size do
 | doc=resexp[i].doc
 | forall the doc.pesoparola as parola do
 | | if not ord.contains(parola.first) OR ord.[parola.first]
 | | <parola.second then
 | | | ord[parola.first]=parola.second
 | | end
 | end
end
for j 1 to 5 and not ord.isEmpty do
 | s=chiave_con_valore_massimo(ord)
 | newquery.append((ord.pop(s)).first)
end
res=res+c*QueryDoc(newquery, x) //c≤1
ordina(res)
print res
```

2.5 PageRank

L'algoritmo esamina la probabilità per un utente di ritrovarsi in una determinata posizione all'interno di una mappa di documenti dopo una serie N di passi. T indica il valore ottimale tempo/precisione e varia da 100 a 500, ma può essere portato fino a N=numero documenti per avere una mappatura completa. L'algoritmo è stato sviluppato sulla base di quello presentato a lezione: creata una lista d'interi S denominata CIT che contiene gli id dei documenti e la probabilità di trovarsi nel dato documento documento al dato istante.
Il valore di probabilità andrà a modificare il peso del documento, i documenti con valore di probabilità medioalto saranno considerati più autorevoli.
In ultima analisi il Pagerank si è dimostrato molto complesso a livello di progettazione, con una notevole difficoltà nel normalizzare i termini in un intervallo [0,1[soprattutto. In particolare il problema comportato da un valore di T superiore a 500, che causava problemi di underflow, è stato risolto settando a zero la probabilità oltre la soglia di trascurabilità (e^{-25}), cosi facendo è stata sacrificata precisione ma ottenuto un risultato stabile.

Algoritmo
PRE: Lista documenti X[1..m] presenti nella collezione con loro relative citazioni
POST: Lista di valori per singolo documento che ne aumenta il peso in base alla probabilità media

```
X[1..m] // lista di documenti m=n° documenti totali
for i 1 to m do
    crea in X[i] una lista di interi S chiamata CIT
    carica in CIT gli id dei documenti i documenti citati
    crea in X[i] un array P[] che indica le probabilità di trovarsi nel doc ai
    vari passi
end
for i 1 to m do
    //setta P P[0]=1/m //imposta probabilità a 1/m di partenza, dove m
    è Numero doc
end
for t 1 to T do
    for j 1 to m do
        a=0
    end
    for j 1 to m do
        for i 0 to CIT.size do
            CIT[i].a=CIT[i].a+j.P[t-1]/(1/m)
        end
    end
end
```

2.6 Latent Semantic Analysis

Per lo sviluppo di LSA è stata usata la libreria Armadillo per c++.
L'algoritmo è stato creato partendo dalla base presentata a lezione e permette di modificare il peso dei documenti in base al peso della Query.
Le parole sono state prelevate dai risultati dei precedenti algoritmi, preventivamente salvati all'interno del file best_Word1000.txt, di queste ne sono state estratte 100 da posizionare in Matrice[x,y] aventi x=50 parole rilevanti e y=100 documenti(50 rilevanti, 50 non rilevanti).
Le coordinate x:y (dove x e y sono due autovettori di Matrice), identificano la posizione di N nell'intervallo q=5, è stato scelto un intervallo ampio per avere un margine maggiore tra pesi differenti, cosi facendo sarà possibile avere per ogni documento una coppia di coordinate x:y che potranno essere moltiplicate al valore Y dell'interrogazione(peso query).
Per ogni documento si otterrà, in questo modo, un peso più preciso in relazione alla query di ricerca.
L'implementazione di LSA è stata particolarmente complicata, il programma ha subito una modifica corposa per gestire meglio l'accesso a tutti i valori prodotti dalla run. Il costo computazione è particolarmente elevato a causa del calcolo matriciale ma, se limitato ai numeri esposti in questo capitolo, rimane comunque contenuto nel minuto.

Algoritmo

PRE: Lista contenente parole pesate ordinane in ordine non decrescente per peso
POST: Lista di 50 documenti rilevanti con peso ricalcolato in pase a LSA

1. Produce un primo ranking[disattiva RF e Pagerank]

2. Seleziona i primi N=100 documenti della run in vettore

3. Crea Matrice [k,N] dove k=(100 keyword più pesanti) e N=(primi 50 doc rilevanti)
   Casella M[Ki,Nj]=1 perogni i,j— i¡=100,j¡=50 se parola presente nel documento

4. Calcola LSA: ottiene autovettori (r=rilevanti nr=non rilevanti)
   proietta il vettore della query sullo spazio da questi generato ottenendo valori (x,y) che fanno riferimento alla posizione del documento rispetto alla query, operazione ripetuta per ogni vettore rappresentante documento.

5. Il peso del documento aumenta in base alla proiezione della query, più il valore è distante dal valore di query meno probabile che sia inerente

alla query stessa. Andranno a formarsi dei punti nel posizionamento che raggruppano i documenti rilevanti

6. Ordina N in ordine non decrescente per peso

2.7 Hyper-linked Induced Topic Search

Per lo sviluppo di HITS è stata utilizzata la libreria Armadillo per c++.
L'algoritmo è stato sviluppato sulla base di quello presentato a lezione, permette una visione sotto forma di grafo delle citazioni tra documenti. Épossibile utilizzare l'algoritmo dopo un primo reperimento di documenti rilevanti oppure in contemporanea al primo ciclo di Pagerank.
A causa dell'elevato peso computazionale il numero di nodi presi in esame è stato ridotto in modo importante a un totale di 100; si tratta comunque di una sottocollezione di documenti selezionati per rilevanza e, grazie a Pagerank, autorevolezza.
Algoritmo

1. Produce primo ranking[senza RF]

2. Estrae i primi N=50 documenti rilevanti della run in vettore

3. Apre matrice di adiacenza prelevando sotto-matrice [100,100] [documento A, documento B] 1 se A linka B

4. Calcola autorevolezza, centralità e rilevanza mediante algoritmo di HITS: A= Vettore autorevolezze matrice i nodo a t passo [i,t] nei cicli non si sposta di colonna in colonna come fossero vettori

```
H= Vettore centralità
for i 1 to 100 do
    t=0
    i.a[t]=i.h[t]=1/14
    converge=false
    while not converge do
        t=t+1
        for k 1 to 100 do
            for j 1 to 100 do
                if k.cit(j) then
                    k.a[t]=k.a[t-1]+j.h[t-1]
                end
            end
        end
        for j 1 to 100 do
            for k 1 to 100 do
                if k.cit(j) then
                    j.a[t]=j.a[t-1]+k.h[t]
                end
            end
        end
        normalizza(a[a.size])
        normalizza(h[h.size])
        x=0.014
        if a[a.size]-a[a.size-1]<x and h[h.size]-h[h.size-1]¡x then
            converge=true
        end
    end
end
```

5. Ottenuti 2 valori:(centralità, autorevolezza),
   normalizza la somma risultati in un intervallo compatibile al valore del peso
   tramite log10(autorevolezza + centralità)

6. Ordina i risultati

3 Risultati sperimentali

Per valutare l'efficacia di ogni algoritmo implementato è opportuno fare un'analisi della rilevanza statistica delle differenze tra i valori di MAP; come baseline si prenderà il valore di MAP della funzione di reperimento. I valori di MAP sono stati ottenuti attraverso il software valutativo TREC_EVAL in ambiente windows. Per confrontare le medie dei diversi metodi si assume che i campioni siano indipendenti, estratti da gruppi diversi, a varianza incognita e varianze campionarie non omogenee. Per ottenere questo risultato si ricorrerà ad un test t di student a due campioni supponendo che i due campioni siano estratti da gruppi che seguano una distribuzione di tipo gaussiano. Prima di procedere con il t-test, è necessario valutare le varianze campionarie dei due gruppi, ossia effettuare un test F di Fisher per verificare l'omoschedasticità (omogeneità delle varianze). Si confronteranno, quindi, le medie di ciascun campione dei due gruppi; le due ipotesi saranno:

- H0: MAP.reperimento = MAP.i $\forall$ i $\in$ {Relevance Feedback Esplicito, Relevance Feedback Pseudo, PageRank, LSA, HITS}
- H1: le due medie (MAP) sono statisticamente diverse. Ovviamente questo confronto va fatto per tutte le altre funzioni prendendo come funzione di base la funzione di reperimento.

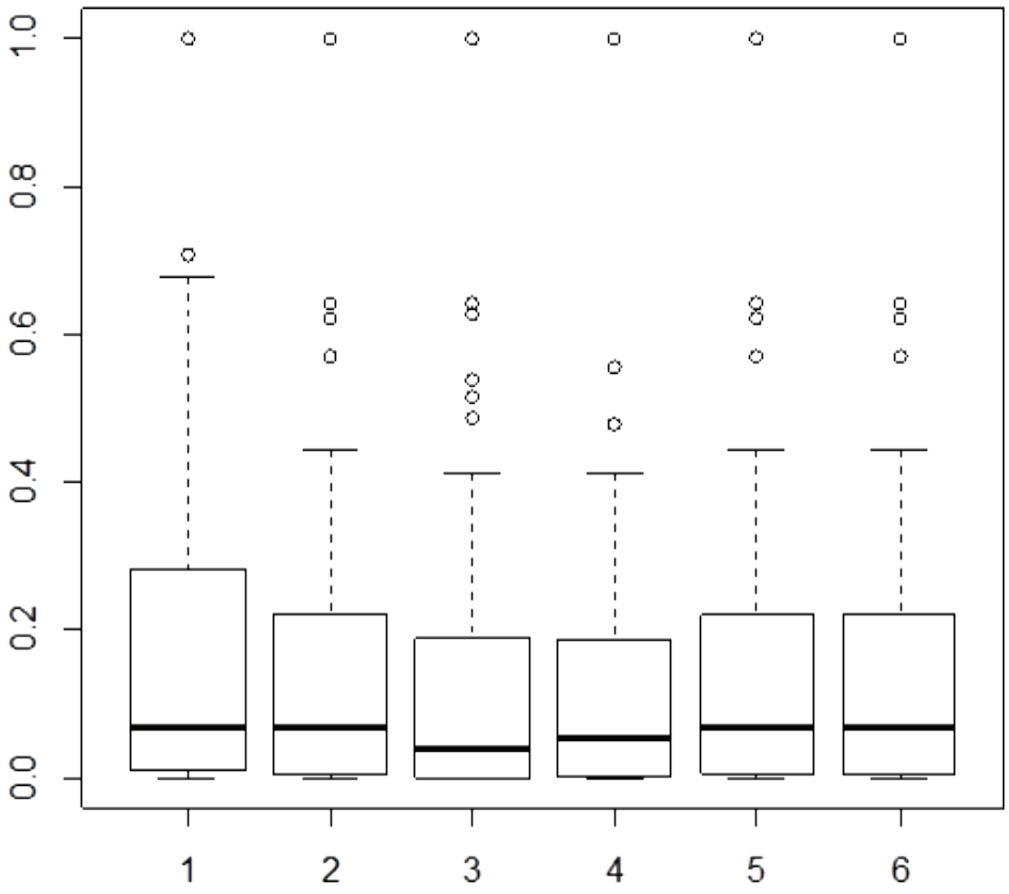

Figura 1: Boxplot relativo all'average precision di ogni algoritmo: Reperimento, Relevance Feedback Esplicito, Blind Feedback, LSA, HITS e PageRank

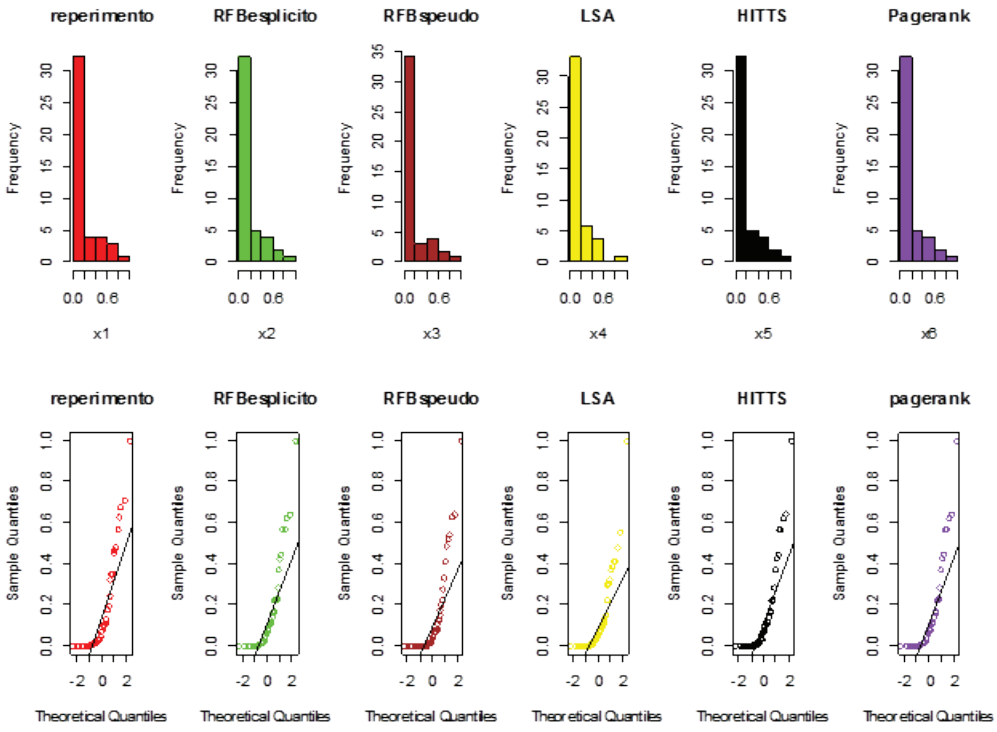

Figura 2: Distribuzione di tipo gaussiano dei risultati degli algoritmi esaminati

Attraverso questi grafici è possibile già dare una valutazione circa il comportamento delle MAP: la funzione di reperimento ha un valore MAP superiore alla MAP del metodo LSA e si può fare lo stesso confronto anche con gli altri casi. Procederemo quindi a valutare se i valori MAP hanno un significati statistici differenti.

Tabella 1: test t-student eseguiti sui risultati degli algoritmi esaminati

	test t-student
Confronto reperimento - RFBesplicito	0.7373
Confronto reperimento - RFBspeudo	0.5337
Confronto reperimento - LSA	0.4510
Confronto reperimento - HITS	0.7367
Confronto reperimento - PageRank	0.7367

È stato ottenuto un p-value maggiore di alpha = 0.05= livello di significatività osservato, quindi possiamo concludere che mediamente i metodi siano statisticamente diversi fra di loro. Questo ci conferma quanto osservato nel grafico precedente.
A ulteriore conferma di ciò si riporta una tabella di analisi della MAP per la prima query applicata ad ogni algoritmo sperimentato. Per ogni algoritmo sono riportati i primi tre documenti reperiti e la mean average precision relativa.

Tabella 2: tabella riassuntiva del valore MAP di ogni metodo per i tre primi documenti reperiti in relazione alla prima query

Algoritmo	Primi 3 documenti reperiti		MAP
Reperimento	1 Q0 1542 1	43,10837	0.2103
	1 Q0 2319 2	42,15067	
	1 Q0 1885 3	42,14629	
Relevance Feedback Esplicito	1 Q0 1542 1	80,1236	0.1511
	1 Q0 2319 2	80,087	
	1 Q0 1885 3	80,0868	
Relevance Feedback Pseudo	1 Q0 1542 1	80,1274	0.1824
	1 Q0 2319 2	80,0897	
	1 Q0 1885 3	80,0896	
PageRank	1 Q0 1542 1	78,7968	0.1705
	1 Q0 2319 2	78,7108	
	1 Q0 1885 3	78,708	
Analisi Semantica Latente	1 Q0 1542 1	37,3365	0.1887
	1 Q0 2319 2	37,2519	
	1 Q0 1885 3	37,2491	
HITS	1 Q0 1542 1	38,0332	0.1887
	1 Q0 2319 2	37,9485	
	1 Q0 1885 3	37,9457	

Questa tabella non fa che evidenziare quanto detto precedentemente circa la diversità statistica degli algoritmi.
La funzione di reperimento ha restituito un valore della MAP di 0.2103, tale valore è stato preso come termine per di confronto con gli altri valori ottenuti attraverso gli altri algoritmi. In particolare i relevance feedback esplicito e pseudo hanno un valore di MAP inferiore rispetto a quello della funzione base. Il gruppo non ha, quindi, ritenuto soddisfacenti i risultati ottenuti dagli algoritmi di Relevance Feedback.
Anche l'algoritmo di Pagerank rispetto al reperimento vede un calo nel valore di MAP, seppure ottenga risultati migliori rispetto alla funzione di retro-azione esplicita. per quanto l'analisi semantica latente et la HITS migliorano il metodo di pagerank, questi rimangono al di sotto del valore baseline (funzione di reperimento).

In sintesi, per quanto l'algoritmo originale abbia dimostrato una Average Precision migliore rispetto agli altri algoritmi è stato possibile valutare come ognuno di questi ultimi (HITS, RFB, LSA e Pagerank) possa offrire delle caratteristiche che permettono di modificare il reperimento dei documenti a seconda delle esigenze, senza che nessuno offra un risultato nettamente migliore rispetto agli altri.

4 Conclusioni

L'esperienza di laboratorio ha permesso al gruppo di sviluppare una conoscenza basilare delle metodologie di Information Retrieval. Le principali difficoltà riscontrate sono stati l'implementazione tutt'altro che immediata degli algoritmi e la gestione della complessità degli stessi, e proprio queste problematiche, mescolate alla relativa inesperienza del gruppo in questo ambito, hanno reso particolarmente difficile l'ottimizzazione del programma.
Per un eventuale miglioramento del sistema è auspicabile una maggiore modularizzazione del software con la costruzione di classi il cui unico scopo sia quello di conservare le modifiche ai pesi generate da ogni algoritmo, un perno per così dire, attorno al quale ruota l'intero sistema, così da avere la possibilità di controllare i risultati di ogni singolo sistema di reperimento. Un altro fattore chiave è la scelta delle variabili arbitrarie, chiaramente il loro valore non può essere particolarmente elevato, dal momento che al suo aumentare si possono verificare fenomeni di overflow dei risultati e può portare ad una convergenza verso valori eccessivamente bassi e, a causa dell'arrotondamento, può perdere informazioni rilevanti. Il reference feedback si è dimostrato eccessivamente inefficiente rispetto alle aspettative. Il Pagerank invece ha avuto un comportamento altalenante con un guadagno di 0.2 punti se combinato con HITS. Un altro aspetto interessante del progetto è il reperimento: la sua valutazione, infatti, restituisce una precisione molto buona, ma quando i suoi risultati vengono utilizzati come baseline per gli altri algoritmi, si verificano crolli di precisione particolarmente interessanti.
In definitiva l'esperienza a messo in evidenza le problematiche dello sviluppo di sistemi IR, partendo da complessità di linguaggio fino ad arrivare alla scelta delle variabili arbitrarie ma anche l'enorme potenziale che questi algoritmi possono racchiudere.