Relazione

Information Retrival

2012/2013

Implementazione e test

dell'Algoritmo di Clustering:

K-Medoids

Antonio Ercole De Luca

413101

Corso di Laurea In Informatica per l'Economia e l'Azienda

Business Intelligence

Introduzione

Per effettuare il clustering delle news, ho deciso di utilizzare parte di una libreria sviluppata durante il mio tirocinio presso il Dipartimento di Informatica sotto la supervisione del dott.Vincenzo Gervasi.

La parte che ho utilizzato riguarda l'individuazione di un insieme di pagine di Wikipedia in italiano a partire da un testo generico.

Viene inclusa la relazione del tirocinio per maggiori dettagli tecnici.

La libreria estrae da un testo una serie di termini; questi non sono necessariamente contenuti nel testo, ma sono titoli di pagine di Wikipedia in lingua italiana.

Ho effettuato il clustering rappresentando ogni news come l'insieme delle pagine estratte dal suo corpo e dal suo titolo.

Tipicamente in IR per rappresentare un singolo documento come insieme di termini viene utilizzata la rappresentazione vettoriale delle occorrenze (bag-of-words); questo è un tipo di rappresentazione sparso; per ragioni di efficienza ho deciso di rappresentare ogni singolo documento come un *insieme (java set)* di termini.

Per calcolare la similarità (o distanza) tra due insiemi di termini (documenti) ho utilizzato Jaccard, che sfrutta la rappresentazione insiemistica per effettuare semplici calcoli di intersezione tra termini.

Nell'algoritmo di clustering implementato, il calcolo della similarità (o distanza) tra due elementi può avvenire diverse volte, quindi per ragioni di efficienza ho implementato una struttura dati di memorizzazione dei risultati.

Nella rappresentazione vettoriale, gli insiemi di termini, sono rappresentati come vettori delle occorrenze, cioè vettori in cui ogni componente appartiene all'insieme {0,1}. L'algoritmo k-means calcola i nuovi centroidi come la media dei vettori che compongono il cluster, quindi non necessariamente un nuovo centroide appartiene all'insieme dei vettori in cui ogni componente appartiene all'insieme {0,1}.

Quindi, per usare come funzione di similarità (o distanza) l'implementazione di Jaccard, che fa uso delle operazioni insiemistiche, ho deciso di implementare e utilizzare una versione modificata dell'algoritmo k-means: k-Medoids.

La differenza fondamentale tra k-Medoids e k-means sta nella fase di scelta del centroide di ogni cluster; in k-Medoids il centroide (medoide) è l'elemento del cluster con distanza media verso tutti gli altri elementi del cluster minore .

Descrizione della fase di Estrazione

Il testo viene diviso in frasi, poi ogni frase viene divisa in token, come carattere di divisione dei token vengono usati lo spazio, la tabulazione e l'accapo;

Attraverso l'uso di un algoritmo a finestra mobile, per ogni frase viene generato un flusso di n-gram di token contigui nel testo, con n che varia da 4 a 1.

Ogni n-gram del flusso, se non appartiene ad un insieme di stop-word italiane, viene cercato in Wikipedia in lingua italiana.

Un' intera copia di Wikipedia in italiano è stata salvata in un database in locale per far si che, le interrogazioni, siano effettuate in maniera efficiente.

Attraverso l'uso della libreria open source Jwpl è stato possibile:

- trasformare il file di dump di wikipedia in un database sql;
- accedere al suo contenuto attraverso un mapping tra il modello a oggetti Java e quello relazionale SQL, che ha permesso di cercare ogni pagina nel database attraverso l'uso di una semplice api Java.

Struttura e composizione del database

Nella fase di costruzione del database, Jwpl immagazzina una serie di informazioni:

- Le pagine di Wikipedia;
- I link ipertestuali tra quest'ultime;
- le categorie a cui appartengono le pagine;
- le gerarchie e i collegamenti ipertestuali tra le categorie;
- altro.

Nella fase di costruzione del database, Jwpl crea due tipi di pagine:

- con contenuti, corpus e collegamenti ipertestuali in uscita;
- senza contenuti, che fanno riferimento a una pagina con contenuti.

Le pagine senza contenuti sono composte da:

- Le pagine di redirect, cioè speciali pagine di Wikipedia, create dagli utenti, per gestire il fenomeno della sinonimia; esse hanno un riferimento alla pagina concreta della quale sono sinonimi;
- Le anchor text, cioè le stringhe di testo utilizzate dagli utenti, nell'intero corpus di Wikipedia, per creare un collegamento ipertestuale a un'altra pagina; queste hanno come riferimento la pagina di cui quel anchor text è collegamento.

Quindi per ottenere una pagina tramite la libreria Jwpl basta utilizzare, o il nome della pagina, o il nome di uno delle sue pagine di redirect, oppure uno dei suoi anchor text.

Problemi riscontrati: Notizie in formato digest

Le intenzioni iniziali erano quelle di utilizzare l'intera libreria sviluppata per il mio tirocinio; essa, sfruttando i collegamenti ipertestuali tra le pagine di Wikipedia, effettua le seguenti fasi:

- Estrazione di una serie di pagine di Wikipedia da un testo;
- Word Sense Disambiguation;
- Calcolo dell'importanza (Ranking) delle pagine.

Queste tecniche funzionano molto bene con testi medio-lunghi, ad esempio 200-300 parole, in quanto sia la fase di Word Sense Disambiguation, sia la fase di Ranking, costruiscono una parte del grafo dei collegamenti ipertestuali tra pagine di Wikipedia, relativo solo alle pagine estratte dal testo; quindi con testi troppo brevi, e di conseguenza con insiemi di pagine troppo piccoli, queste due fasi perdono di efficacia.

L'alternativa sarebbe stata creare uno spider che prelevasse l'intera notizia dall'url.

Valutando questa ipotesi, dopo una serie di tentativi, ho notato che le pagine web contenenti le news in questione, possedevano strutture troppo diverse.

Quindi si è scelto di applicare solo la fase di estrazione, che però restituisce per ogni termine trovato in Wikipedia un'insieme di pagine sinonime, che cioè necessitavano di disambiguazione.

Per semplificare, ho rappresento ogni news come l'unione degli insiemi di pagine sinonime che la fase di estrazione mi restituiva. Intuendo casualmente che un n-gram che restituisce più sinonimi potesse essere più importante di un n-gram che ne restituisce uno solo, si è potuto sfruttare la funzione di similarità di Jaccard, che da maggior importanza, perché in maggior numero, a insiemi di termini che sono stati generati da n-gram che generano più sinonimi.

Descrizione della fase di clusterizzazione

Funzione di Distanza

Come funzione distanza ho usato la funzione:

$$Distanza(A,B)=1-Jaccard(A,B)$$

Sfruttando la funzione si similarità di Jaccard; con A e B insiemi di termini.

Ricordo che ogni ogni notizia è rappresentata da un insieme di termini; utilizzando Jaccard il calcolo della distanza si riduce al calcolo dell'intersezione tra insiemi.

Ho ritenuto più efficiente questo approccio piuttosto che maneggiare l'intera rappresentazione vettoriale, la cui complessità sarebbe stata in funzione dell'intero insieme di termini che appaiono in tutte le notizie.

Siano A e B due insiemi di termini:

$$Jaccard(A,B)=\frac{|A\cap B|}{|A\cup B|}$$

Dalla teoria degli insiemi:

$$|A \cup B| = |A| + |B| - |A \cap B|$$

Semplificando, abbiamo così solo bisogno di calcolare l'intersezione:

$$Jaccard(A, B) = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Distance Matrix

Per migliorare le prestazioni, è stata aggiunta una struttura dati che contiene il risultato del calcolo delle distanze tra coppie di news.

Questa struttura dati può essere vista come una matrice delle distanze, contenente valori nulli, nel senso che, durante l'esecuzione, i valori che non sono stati calcolati nei passi precedenti vengono all'occorrenza generati e memorizzati per utilizzi futuri.

Le news da clusterizzare sono 5000, quindi una soluzione di tipo matriciale classica occuperebbe 5000^2 double da 64 bit, quindi in totale 1.6 Gb di memoria virtuale occupata.

Essendo una matrice simmetrica, per ogni coppia di news, verrebbero salvati due valori di distanza identici. Per evitare questo problema viene utilizzato uno stratagemma: per ottenere la distanza tra due elementi, questi vengono confrontati e vengono individuati un elemento minore ed uno maggiore. Nella matrice si accede alla riga tramite l'elemento minore e, alla colonna tramite quello maggiore, in modo da accedere solo agli elementi della matrice che sono al di sopra della diagonale.

Alla luce di ciò, cerchiamo una soluzione che ci permetta di risparmiare spazio evitando così l'occupazione di memoria con elementi della matrice che non vengono mai utilizzati.

A questo scopo ho implementato una struttura dati che ho denominato HashTreeMatrix:

L'elemento minore m viene utilizzato come chiave in una HashMap che restituisce un TreeMap che mappa, ad ogni elemento verso cui è stata calcolata la distanza con m, il valore della distanza stessa.

Con un tempo di accesso di $O(\log(n))$

Dopo una serie di risultati sperimentali sulla clusterizzazione delle news, la memoria virtuale del processo si aggirava mediamente intorno ai 1.3 Gb.

Questa soluzione non è la migliore in assoluto, ma ritengo sia valida agli scopi di questo progetto; essa dovrebbe essere rivalutata, dovendo affrontare quantità di notizie superiori. Per esempio, potrebbe non essere necessario mantenere in memoria tutta la matrice delle distanze, ma semplicemente solo quelle utilizzate recentemente; per esempio utilizzando una struttura dati che attui da memoria cache, oppure utilizzandone k, una per ogni cluster, sfruttando la località temporale del calcolo delle distanze dell'algoritmo k-Medoids.

Algoritmo K-Medoids

L'algoritmo implementato è una variante dell'algoritmo k-means, la differenza tra i due risiede nella scelta del centroide; in k-means il centroide è la media degli elementi che appartengono al cluster, in k-Medoids viene scelto l'elemento con distanza media minore verso tutti gli altri elementi del cluster.

Sia c_k il cluster k-esimo, e x_n l'elemento con distanza media minore verso tutti gli altri elementi del cluster:

$$\forall x_i \in c_k \quad \sum_{x_j \in c_k} distanza(x_n, x_j) \leq \sum_{x_j \in c_k} distanza(x_i, x_j)$$

Con questa scelta del medoide viene minimizzato anche il valore dell'SSE.

In quanto per ogni cluster viene scelto quel medoide che minimizza la parte della SSE_{tot} attribuibile al cluster c_k .

Problemi riscontrati: Medoidi Ombra

Come scelta implementativa, ogni cluster ha almeno un elemento: il suo medoide.

Se avvengono contemporaneamente le seguenti condizioni:

- Nella fase iniziale di estrazione casuale dei medoidi vengono estratti due uguali o molto simili tra loro;
- Ad uno dei due sono assegnati tutti gli elementi del dataset simili a entrambi;

  Definiamo medoide *attrattivo* quello a cui vengono assegnati tutti gli elementi simili a entrambi, e medoide *ombra* l'altro.
- Nelle iterazioni successive e fino a convergenza dell'algoritmo, il medoide *attrattivo* continua a essere il medoide del suo cluster.
- Il medoide *ombra* rimane l'unico elemento del suo cluster a causa della presenza del medoide *attrattivo* nell'altro cluster.

Analizzando il problema risulta che il cluster del medoide ombra:

- Ha un unico elemento a causa della forte attrattività del primo cluster;
- Non può cambiare il valore del medoide perché questo è l'unico elemento del cluster.

Questo porta a convergenza l'algoritmo senza una corretta gestione di questa fonte di inefficienza; in altri termini esiste una soluzione migliore e con un minore valore di sse;

Una soluzione migliore avrebbe:

- Il medoide *ombra* associato al cluster del medoide *attrattivo, v*isto che il medoide

ombra e quello attrattivo sono uguali o simili;

- Come nuovo medoide del cluster *ombra* l'elemento che influisce maggiormente sul valore dell'SSE.

Presento una tecnica che ho sperimentato per il problema dei medoidi *ombra:*

Sia K l'insieme dei cluster, $\forall c_j \in K$, m_j il suo medoide.

Ricordiamo valore dell'SSE:

$$SSE = \sum_{c_j \in K} \sum_{x_i \in c_j} distanza(x_i, m_j)^2$$

Mostriamo due casi, una con la presenza di un medoide *ombra* e l'altra senza:

Cluster Ombra Presente

Siano:

- c_o il cluster contenente il medoide *ombra* m_o con: $c_o = \{m_o\}$
- c_a il cluster contenente il medoide *attrattivo* m_a ed altri elementi x_l , con:

$$c_a = \{m_a, x_0, \dots, x_i\}$$

Sia:

x_m l'elemento del cluster c_a più distante dal medoide m_a , cioè:

$\forall x_i \in c_a - \{m_a, x_m\}$ $distanza(x_m, m_a)^2 > distanza(x_i, m_a)^2$

Siano SSE_{c_a} SSE_{c_o} gli errori attribuibili ai cluster c_a e c_o :

- $SSE_{c_o} = distanza(m_o, m_o)^2 = 0$
- $SSE_{c_a} = distanza(m_a, m_a)^2 + \sum_{x_j \in c_a - \{m_a\}} distanza(x_j, m_a)^2 = \sum_{x_j \in c_a - \{m_a\}} distanza(x_j, m_a)^2$

In cui è possibile scomporre l'errore in quello attribuibile all'elemento più distante del cluster x_m e quello non attribuibile:

$$SSE_{c_a} = distanza(x_m, m_a)^2 + \sum_{x_j \in c_a - \{m_a, x_m\}} distanza(x_j, m_a)^2$$

Allora l'SSE è:

$$SSE_{tot} = \sum_{c_j \in K - \{c_a, c_o\}} \sum_{x_i \in c_j} distanza(x_i, m_j)^2 + SSE_{c_a} + SSE_{c_o}$$

Sostituendo:

$$SSE_{tot} = \sum_{c_j \in K-\{c_a, c_o\}} \sum_{x_i \in c_j} distanza(x_i, m_j)^2 + \sum_{x_j \in c_a - \{m_a, x_m\}} distanza(x_j, m_a)^2 + distanza(x_m, m_a)^2$$

In cui l'errore può essere scomposto e attribuito a:

- L'addendo più a destra relativo all'elemento x_m
- Quello centrale, attribuibile ai due cluster c_a e c_o, meno l'elemento x_m
- Quello a sinistra attribuibile a gli altri clusters.

Rimozione del Cluster Ombra

Osserviamo una soluzione con valore SSE minore ottenuta sostituendo al posto dei due clusters precedenti due così costruiti:

- c_o^1 con medoide x_m, quindi: $c_o = \{x_m\}$
- c_a^1 come il vecchio c_a a cui è stato tolto l'elemento x_m ed è stato aggiunto l'elemento m_o:

$$c_a^1 = (c_a - \{x_m\}) \cup \{m_o\}$$

Quindi calcolando gli errori attribuibili ai cluster c_a^1 e c_o^1:

- $SSE_{c_o}^1 = distanza(x_m, x_m)^2 = 0$
- $SSE_{c_a}^1 = distanza(m_a, m_a)^2 + distanza(m_o, m_a)^2 + \sum_{x_j \in c_a^1 - \{m_a, m_o\}} distanza(x_j, m_a)^2$

Sostituendo con:

- $c_a^1 - \{m_a, m_o\}$ con: $c_a - \{m_a, x_m\}$
- $distanza(m_a, m_a)^2$ con 0

Abbiamo:

$$SSE_{c_a}^1 = distanza(m_o, m_a)^2 + \sum_{x_j \in c_a - \{m_a, x_m\}} distanza(x_j, m_a)^2$$

L'SSE totale del secondo caso é:

$$SSE_{tot}^1 = \sum_{c_j \in K-\{c_a, c_o\}} \sum_{x_i \in c_j} distanza(x_i, m_j)^2 + \sum_{x_j \in c_a - \{m_a, x_m\}} distanza(x_j, m_a)^2 + distanza(m_o, m_a)^2$$

Quindi se vogliamo che:

$$SSE_{tot} > SSE_{tot}^1$$

Sostituendo, dobbiamo avere che:

$$\sum_{c_j \in K-\{c_a, c_o\}} \sum_{x_i \in c_j} distanza(x_i, m_j)^2 + \sum_{x_j \in c_a - \{m_a, x_m\}} distanza(x_j, m_a)^2 + distanza(x_m, m_a)^2 > \ldots$$
$$\sum_{c_j \in K-\{c_a, c_o\}} \sum_{x_i \in c_j} distanza(x_i, m_j)^2 + \sum_{x_j \in c_a - \{m_a, x_m\}} distanza(x_j, m_a)^2 + distanza(m_o, m_a)^2$$

Che è vera se e solo se:

$$distanza(x_m, m_a)^2 > distanza(m_o, m_a)^2$$

E' quindi possibile sostituire un medoide *ombra* con l'elemento più distante di un cluster generico c_a (con almeno un'elemento x_m diverso da m_a) ed ottenere una soluzione con un SSE minore, solo se la distanza tra il medoide *ombra* m_o e quello *attrattivo* m_a è minore della distanza tra l'elemento x_m e il suo medoide *attrattivo* m_a .

Quindi allo scopo di eliminare i medoidi *ombra,* ho implementato una procedura che ad ogni iterazione dell'algoritmo verifichi che ci siano clusters *ombra* e, in tal caso e se possibile, proceda alla loro rimozione secondo la procedura precedente.

Descrizione della procedura:

Per ogni clusters i-esimo *ombra* e per ogni cluster j-esimo, j diverso da i, la procedura effettua lo scambio tra il medoide ombra e l'elemento più distante del cluster j-esimo solo se è verificata la seguente condizione:
cioè che la distanza tra i due medoidi sia minore della distanza tra l'elemento più distante dal medoide ed il medoide stesso del cluster j-esimo.

Convergenza

La convergenza dell'algoritmo k-means è nota nella letteratura.
k-Medoids differisce da k-means solo nella scelta dei nuovi medoidi che comunque, vengono scelti decrementando lo SSE ad ogni iterazione.
Questo assicura la convergenza dell'algoritmo k-Medoids.
L'introduzione della procedura di rimozione dei medoidi *ombra,* mantiene assicurata la convergenza perché diminuisce il valore dello SSE ad ogni iterazione; si tratta di un minore stretto e non di un minore uguale.
Questa procedura effettua una modifica della soluzione solo se quella ottenuta è migliore di quella attuale: è una ricerca di tipo *greedy* ed esplora lo spazio delle soluzioni verso l'ottimo locale.

Pseudo-Codice dell'algoritmo

L'algoritmo è composto dai seguenti passi:

```
doClustering( k , elements )
    n_iterations = 0
    medoids = sceltaMedoidiCasuali( k , elements )
    clusters = costruisciClustersVuoti( medoids )
    do
        n_iterations++
        if ( n_iterations != 1)
            pulisciClusters( clusters )
        foreach element in elements
            associaAlMedoidePiùVicino( element , clusters )
        if ( n_iterations != 1)
            rimuoviMedoidiOmbra( clusters )
        newMedoids = emptyVector()
        foreach cluster in clusters
            newMedoids.add( ricalcolaMedoide( cluster ) )
        end = controlloConvergenza( medoids , newMedoids , n_iterations )
        medoids = newMedoids
    while(not end)
```

Siano n numero di elementi da clusterizzare e k il numero dei clusters.

Abbiamo effettuato l'analisi della complessità dell'algoritmo prescindendo considerazioni sulla complessità del calcolo della distanza e supponendo il suo costo computazionale di quest'ultimo come costante: $O(1)$.

Mostriamo di seguito una breve descrizione delle procedure implementate e un'analisi della loro complessità:

SceltaMedoidiCasuali(k , elements)

Vengono estratti, con probabilità uniforme e senza reinserimento, k elementi da elements.

Complessità: $O(k)$.

CostruisciClustersVuoti(medoids)

Vengono generate le strutture dati che rappresentano i clusters, per costruzione ogni cluster ha sempre un elemento: il suo medoide.

Complessità: $O(k)$

PulisciClusters(clusters)

Ad ogni cluster vengono rimossi tutti gli elementi associati, viene mantenuto solo l'elemento medoide.

Complessità: $O(k)$

Questa procedura non viene effettuata nella prima iterazione perché i clusters sono appena stati creati.

AssociaAlMedoidePiùVicino(element , clusters)

Viene associato element al medoide più vicino calcolando la distanza con ognuno dei medoidi dei clusters; in caso di cluster con medoidi a distanza uguale, viene preso quello con valore hashcode più alto, in maniera tale che l'algoritmo non oscilli tra soluzioni diverse con sse uguale.

Complessità: $O(k)$

Dentro il suo ciclo for: $O(n*k)$

RimuoviMedoidiOmbra(clusters)

In questa fase vengono eliminati i clusters con medoidi ombra.

Per ogni cluster ombra, ne viene cercato uno che soddisfi i requisiti per lo scambio. Nel caso in cui venga trovato, viene effettuato lo scambio.

La complessità dello scambio è: $O(n)$, in quanto per uno dei due cluster devo ricalcolare l'elemento più distante dal suo medoide, facendo una scansione di tutti i suoi elementi.

Complessità: $O(k*k*n)$

E' stato riscontrato che questa procedura funziona meglio se applicata a partire dalla seconda iterazione.

RicalcolaMedoide(cluster)

Questa procedura, per ogni elemento del cluster, calcola la distanza verso tutti gli altri elementi del cluster. Il nuovo medoide sarà l'elemento con distanza minima.

Se è stato trovato un nuovo medoide nel cluster, viene ricalcolato l'elemento del cluster più distante dal nuovo medoide: $O(n)$.

Complessità: $O(n*n)+O(n)=O(n*n)$

Dentro il suo ciclo for: $O(n*n*k)$

ControlloConvergenza(medoids, newMedoids, n_iterations)

L' algoritmo termina se almeno una di queste condizioni non è soddisfatta:

Condizioni di terminazione:

- Numero di iterazioni < Massimo numero di iterazioni
- Percentuale di medoidi cambiati > Percentuale minima di medoidi cambiati
- Distanza media tra medoidi > Minima distanza media tra medoidi

Per ogni medoide viene calcolata la distanza e l'uguaglianza con il corrispettivo medoide dell'iterazione precedente, in maniera tale da poter calcolare la percentuale di medoidi cambiati nell'attuale iterazione, e la distanza media tra questi.

Complessità: $O(k)$

Analisi della Complessità dell'algoritmo:

Visto che k è minore di n , tra le procedure del ciclo principale la più dispendiosa è: **RicalcolaMedoide.**

Sia I il numero di iterazioni che l'algoritmo effettua; la complessità totale dell'algoritmo di clustering è:

$$O(I*n*n*k)$$

Con n che deriva dalla dimensione del cluster con cardinalità massima.

In realtà grazie alla procedura di rimozione dei cluster *ombra,* n si avvicina molto al valore $\frac{n}{k}$, e cioè la cardinalità dei clusters sembra essere molto bilanciata.

Quindi la complessità media tende a:

$$\Theta(I*\frac{n}{k}*n)=\Theta(I*\frac{n^2}{k})$$

Descrizione dei test effettuati

Per individuare il valore di k ottimale per ogni data set ho utilizzato la seguente tecnica euristica:

Ho fatto crescere k e, per ogni valore, ho eseguito l'algoritmo 5 volte in modo da poter stimare lo SSE medio per quel valore di k.

Al variare di k, per ogni dataset ho studiato l'andamento dello sse-medio attraverso un opportuno grafico; in questo modo ho potuto individuare un valore orientativo di k tale per cui il guadagno in termini di sse, tra k e (k-1), fosse trascurabile.

Per per ogni dataset ho scelto i seguenti valori di k:

Dataset:			
economia	125	245	350
italia	160	230	
mondo	137	230	
scienza	137	245	
spettacoli	125	245	
sport	89	137	233

I risultati sono forniti in allegato alla relazione.

Prima di ogni cluster è visualizzato l'insieme dei titoli delle pagine di Wikipedia che rappresentano le news del cluster.

Da notare la presenza delle date.

Parametri utilizzati:

- `MAX_ITERATIONS = 30;`
- `MIN_CHANGED_MEDOIDS_PERCENTAGE = 0;`
- `MIN_AVERAGE_DISTANCE_BETWEEN_MEDOIDS = .000000000001;`

In maniera tale da fermare la procedura se la distanza media tra medoidi è quasi zero oppure sono superate le 30 iterazioni.

Raramente ho riscontrato più di 10 iterazioni.

Miglioramenti Futuri

- Risolvere il problema della Word Sense Disambiguation.
- Come misura di similarità tra insiemi di termini sarebbe interessante utilizzare una che tenga conto dei collegamenti ipertestuali tra pagine di Wikipedia.
- Valutare la fattibilità e l'efficacia di un adattamento dell'algoritmo al problema del clustering di grafi.
- Valutare la fattibilità e l'efficacia di una re-implementazione dell'algoritmo che faccia uso di un framework che permetta la scalabilità necessaria alla gestione di grandi moli di dati, per esempio Hadoop, Mahout o Giraph.