ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA
CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

TITOLO DELL'ELABORATO

TECNICHE DI TEXT MINING PER L'AUTOORGANIZZAZIONE DELLA CONOSCENZA

Tesi in

Sistemi Multi-Agente LM

Relatore:
Chiar.mo Prof. Ing.
Andrea Omicini

Presentata da:
Michele Pratiffi

Correlatore:
Ing. Stefano Mariani

Sessione seconda
Anno Accademico 2012/2013

Indice

Introduzione

L'elaborato, che verrà qui esposto, ha come scopo l'analisi delle tecniche di Text Mining e la loro applicazione all'interno di processi per l'auto-organizzazione della conoscenza. La prima parte della tesi si concentra sul concetto del Text Mining. Viene fornita la sua definizione, i possibili campi di utilizzo, il processo di sviluppo che lo riguarda e vengono esposte le diverse tecniche di Text Mining. Si analizzano poi alcuni tools per il Text Mining e infine vengono presentati alcuni esempi pratici di utilizzo. Il macro-argomento che viene esposto successivamente riguarda TuCSoN, una infrastruttura per la coordinazione di processi: autonomi, distribuiti e intelligenti, come ad esempio gli agenti. Si descrivono innanzi tutto le entità sulle quali il modello si basa, vengono introdotte le metodologie di interazione fra di essi e successivamente, gli strumenti di programmazione che l'infrastruttura mette a disposizione. La tesi, in un secondo momento, presenta MoK, un modello di coordinazione basato sulla biochimica studiato per l'auto-organizzazione della conoscenza. Anche per MoK, come per TuCSoN, vengono introdotte le entità alla base del modello. Avvalendosi MoK, dell'infrastruttura TuCSoN, viene mostrato come le entità del primo vengano mappate su quelle del secondo. A conclusione dell'argomento viene mostrata un'applicazione per l'auto-organizzazione di news che si avvale del modello. Il capitolo successivo si occupa di analizzare i possibili utilizzi delle tecniche di Text Mining all'interno di infrastrutture per l'auto-organizzazione, come MoK. Nell'elaborato vengono poi presentati gli esperimenti effettuati sfruttando tecniche di Text Mining. Tutti gli esperimenti svolti hanno come scopo la clusterizzazione di articoli scientifici in base al loro contenuto, vengono quindi analizzati i risultati ottenuti. L'elaborato di tesi si conclude mettendo in evidenza alcune considerazioni finali su quanto svolto.

Capitolo 1

Text Mining

1.1 Definizione

Il Text Mining può essere definito come, un processo di scoperta di nuova conoscenza o conoscenza nascosta da collezioni o singoli documenti di testo. L'estrazione della conoscenza è supportata da strumenti informatici e deve essere il più automatica possibile. Utilizzando un insieme di strumenti di analisi si cerca di identificare ed esplorare pattern interessanti nell'insieme di documenti. Per questo il Text Mining ha molto in comune con il Data Mining. Con il termine Data Mining ci si riferisce infatti ad un insieme di tecniche e metodologie che hanno come obiettivo l'estrazione di informazioni sconosciute partendo da grandi moli di dati. La principale differenza è che, mentre il Data Mining parte da dati generalmente strutturati in database relazionali, il Text Mining si basa su dati, testi, che sono generalmente non strutturati. La maggior parte del processo di sviluppo di applicazioni di Text Mining è identico a quello di Data Mining. É nella parte di preprocessing dei dati che si trovano le maggiori differenze. Mentre nel Data Mining in questa fase ci si concentra sulla selezione dei dati da usare e sulla loro normalizzazione, nel Text Mining ci si concentra nell'estrazione di caratteristiche rappresentative del documento. Nella pratica si cerca di trasformare i dati non strutturati (i testi) in dati strutturati, per far ciò, nel Text Mining si usano molte tecniche riconducibili ad altri campi come quello della ricerca di informazioni, l'estrazione di informazioni e di Natural Language Processing (NPL).

1.2 Il processo di sviluppo dell'applicazione

Il processo di sviluppo di una applicazione di Text Mining segue quello delle applicazioni di Data Mining. Si può quindi seguire il processo CRoss Industry Standard Process for Data Mining o Crisp DM, il quale si suddivide nei seguenti passi principali: comprensione del problema, raccolta e comprensione dati, preparazione dati, modellazione, valutazione

del modello e deployment. Questo è un processo a spirale per cui è possibile prima del deployment tornare ai passi precedenti e modificare alcune scelte fatte, come ad esempio cambiare il modello scelto i suoi parametri. Nei paragrafi sottostanti verranno una ad una spiegate le varie fasi del processo.

1.2.1 Comprensione del problema

In questa fase si analizza il problema che si vuole risolvere per capirne le peculiarità e decidere infine ciò che si vorrebbe ottenere come risultato del processo di Text Mining. Ad esempio riassumere i concetti principali di un insieme di testi o raggruppare testi che contengono informazioni correlate.

1.2.2 Raccolta e comprensione dati

Presentato il problema che si vuole affrontare, la raccolta di dati per il Text Mining può andare dal semplice utilizzo di raccolte di testi già ben definite, alla raccolta nel web di informazioni di vario tipo (tweet, mail, blog). La ricerca e raccolta di informazioni in maniera automatica da Internet richiede l'utilizzo di crawlers che per alcune applicazioni vengono creati ad-hoc. Una delle fonti di dati per il Text Mining, che risulta oggi essere la più usata e citata nei vari articoli di riferimento, è PubMed un repository online del National Library of Medicine che contiene informazioni (abstract, citazioni, articoli) relative ad articoli di ricerca nel campo biomedico. Il suo database contiene gli abstract di circa 12 milioni di articoli ed è altamente dinamico dato che vengono aggiunti circa 40000 abstract al mese.

1.2.3 Preparazione dati

La preparazione dei dati è il passo più importante e caratteristico del Text Mining. In questo processo i dati non strutturati o parzialmente strutturati vengono elaborati per trasformarli in strutturati. In questa fase vengono utilizzati algoritmi di Natural Language Processing, metodi matematici [8] e stemming. Generalmente come risultato di questo processo per ogni testo si ottiene un vettore di feature che lo caratterizza, spesso parole o frasi. Nella fase di preparazione dati, in tutti i metodi, tranne in quelli matematici, come primo step si applica un processo di tokenizzazione del testo. Successivamente si può procedere o meno alla rimozione delle stop word, parole non rilevanti per il processo (come congiunzioni, avverbi, articoli), in maniera tale da ridurre la dimensione dei vettori. Per ridurre ulteriormente le dimensioni dei vettori si può applicare lo stemming, che altro non è che un processo col quale si raggruppano termini correlati, ad esempio si possono aggregare tutte le varie coniugazioni di un verbo. Uno degli algoritmi più famosi per lo stemming è quello proposto da Porter [15] che viene utilizzato all'interno di molti lavori sul Text Mining. Con l'ausilio di ontologie si può ulteriormente ridurre il numero

di feature aggregando le parole che esprimono lo stesso concetto come i sinonimi. In alcuni progetti di Text Mining vengono selezionati solo le feature più rilevanti che caratterizzano il testo prima di passare alla fase di modellazione. Ad esempio, negli algoritmi di classificazione si scelgono sono le parole che permettono una miglior assegnazione delle classi con il metodo del guadagno di informazione [7].

1.2.4 Modellazione

Questa è la fase dove vengono applicati gli algoritmi di Text Mining, sui dati ora strutturati, con l'intento di scoprire conoscenza nascosta e pattern ricorrenti all'interno dell'insieme di documenti. Gli algoritmi applicati si dividono in due classi: supervisionati e non supervisionati. Fanno parte degli algoritmi supervisionati, ad esempio, alcuni algoritmi di classificazione; mentre la maggior parte dei non supervisionati sono gli algoritmi di clustering.

1.2.5 Valutazione

I risultati ottenuti nella fase di modellazione vengono quindi analizzati e valutati sulla base di vari parametri che variano a seconda dell'algoritmo utilizzato. Se si sfrutta la classificazione la valutazione avverrà sulla base dell'accuratezza, del richiamo e della precisione. Nel caso in cui i risultati non siano soddisfacenti si riparte da una fase precedente nel processo a spirale in maniera tale da apportare modifiche alle scelte fatte, nell'intento di migliorarli.

1.2.6 Deployment

In questa fase il sistema viene messo in opera e consegnato all'utilizzatore finale, il cliente che lo aveva richiesto. Spesso in questa fase vengono anche forniti metodi di visualizzazione che permettono un miglior utilizzo ed una maggior comprensione dei risultati.

1.3 Algoritmi di Text Mining

Verranno qui di seguito, nel dettaglio, presentati alcuni degli algoritmi più utilizzati nel campo del Text Mining.

1.3.1 Algoritmi di preparazione dati

Questi algoritmi sono spesso creati ad-hoc, sfruttando al loro interno algoritmi e concetti estrapolati da campi come il Natural Language Processing. Alcuni algoritmi sfruttano

dizionari che contengono al loro interno un insieme di parole non rilevanti dette "stop word" che verranno rimosse dall'insieme delle feature che caratterizzano il testo. Un algoritmo che ricorre spesso nelle varie applicazioni è il Porter stemmer [15], che ora verrà descritto più in dettaglio. Il Porter stemmer è un algoritmo, per la lingua inglese, che permette il raggruppamento di parole che derivano l'una dall'altra come ad esempio connect, connected e connection. L'algoritmo ha alla sua base due concetti, quello di consonante e vocale che vengono, così definite:

- consonante : è una lettera di una parola diversa da A, E, I, O, U e Y preceduta da una consonante;
- vocale : tutte le altre lettere che non sono consonanti sono vocali.

Le consonanti vengono denotate con c mentre le vocali con v, una lista di consonanti ccc viene denotata con C mentre una lista di vocali vvv con V. Ogni parola può essere rappresentata dalla seguente produzione

```
[C](VC)^m [V]
```

gli elementi dentro parentesi quadre possono apparire 0 o 1 volta mentre l'elevazione alla m denota che l'elemento può essere ripetuto m volte. Per una miglior comprensione presento alcune parole con relativo valore di m:

- m = 0 : tr, ee, tree, y, by;
- m = 1 : trouble, oats, trees, ivy;
- m = 2 : troubles, private, oaten, orrey.

Il valore di m viene detto "misura della parola" e viene utilizzato all'interno delle regole per rimuovere i suffissi. Le regole hanno la seguente forma

$$\text{(condizione) suffisso1} \rightarrow \text{suffisso2}$$

e il loro significato è il seguente: se la parte della parola prima del suffisso1 soddisfa la condizione allora sostituisco il suffisso1 con il suffisso2. Ad esempio la regola

$$(m > 1)\ \text{ement} \rightarrow$$

applicata alla parola replacement fornisce come risultato replac. All'interno delle condizioni si possono usare:

- *G il tronco della parola finisce con g;
- *v* il tronco della parola contiene una vocale;

- *d il tronco della parola finisce con una doppia consonante;
- *o il tronco della parola finisce con una sequenza cvc con la seconda c diversa da W, X e Y.

L'algoritmo si basa su 5 step di elaborazione delle parole in ogni step possono essere applicate delle regole di troncatura. Ad esempio, il primo step si occupa principalmente dell'eliminazione delle forme plurali e dei participi passati.

1.3.2 Clustering

Gli algoritmi di clustering possono essere utilizzati per trovare gruppi di documenti con caratteristiche simili. Come risultato di questi algoritmi si ottengono degli insiemi di documenti, i quali essendo all'interno dello stesso cluster devono essere il più possibile simili tra loro mentre i documenti di cluster differenti devono essere il più diversi possibile. É quindi necessario avere degli strumenti che permettano di calcolare la similarità fra i documenti.

K-means

É uno degli algoritmi di clustering ad oggi più usato e, sopratutto nel campo del Text Mining, ottiene risultati ottimi. Il concetto alla base dell'algoritmo è assai semplice, si parte da una soluzione con un numero ben definito di cluster e si prova poi a migliorarla. I passi che l'algoritmo attua sono i seguenti:

Algorithm 1 K-means

```
1: si scelgono k punti in maniera casuale come centroidi
2: while i centroidi non sono stabili, cioè non si spostano do
3:     assegna ogni punto dello spazio dei dati al centroide più vicino
4:     ricalcola i centroidi
5: return l'insieme dei cluster
```

Generalmente come risultato si ottiene un ottimo locale. Per il funzionamento dell'algoritmo sono necessari un metodo intelligente per assegnare i centroidi iniziali ma, sopratutto, una funzione che permetta di calcolare la similarità fra i testi.

Clustering gerarchico

Questo metodo è detto gerarchico perché il risultato ottenuto altro non è che una gerarchia di cluster, il processo può essere:

- top down: si parte da un cluster che contiene tutti i documenti e si continua fino ad avere un cluster per ogni documento;

- bottom up: lo stato di partenza prevede un cluster per ogni documento e si procede fino ad avere un cluster che li contenga tutti.

L'approccio top down è poco utilizzato dato che i suoi risultati non sono buoni. Nell'approccio bottom up ad ogni passo i due cluster più simili vengono fusi in un singolo cluster. Il risultato ottenuto grazie a queste metodologie può essere presentato come una gerarchia grazie ai dendogrammi, che altro non sono che dei diagrammi ad albero.

Bi-section-k-means

É un algoritmo di clustering veloce e di buona qualità che ottiene anche risultati migliori del k-means. L'algoritmo, come si poteva prevedere dal nome, si basa sul k-means; ad ogni iterazione divide il cluster più grosso, ottenuto con il k-means, in due, finché non si raggiunge il numero di cluster predefiniti. A volte invece del cluster più grosso l'algoritmo divide il cluster con la maggior varianza.

Self Organizing Map o SOM

Sono una speciale architettura di reti neurali che clusterizzano vettori di grosse dimensioni in base ad una misura di similarità. Una delle sue caratteristiche principali è che non solo i testi all'interno dello stesso cluster sono simili ma, anche i documenti appartenenti a cluster vicini sono più simili di quelli appartenenti a cluster distanti. La struttura della rete neurale è su due livelli:

- i neuroni del livello di input sono pari alla dimensione dell'input, nel caso del Text Mining, il vettore di parole;
- i neuroni del livello di output sono pari al numero di cluster che si vuole ottenere.

Tutti i neuroni del livello di input sono collegati a tutti i neuroni del livello di output. I pesi delle connessioni dipendono dalla posizione nello spazio bidimensionale dei dati correlati. Prima della fase di apprendimento i pesi sono inizializzati casualmente. Nella fase di apprendimento i vettori che rappresentano ogni singolo documento vengono propagati nella rete, i pesi del neurone più simile $\overrightarrow{ws}$ vengono modificati seguendo la seguente regola: $\overrightarrow{ws}' = \overrightarrow{ws} + \sigma \cdot (\overrightarrow{ws} - \overrightarrow{wi})$ dove $\overrightarrow{wi}$ è il vettore di input. Per conservare le relazioni di vicinanza, anche gli altri neuroni vicini rispetto alla disposizione bidimensionale a quello vincitore cambiano i loro pesi. Man mano che ci si allontana dal neurone vincitore la variazione diminuisce.

1.3.3 Classificazione

Nella classificazione l'obiettivo è quello di assegnare classi predefinite ai testi. Ad esempio associare a notizie una etichetta come "politica", "sport", "gossip", eccetera. Tutti gli

algoritmi di classificazione partono da un training set con un insieme di testi a cui è già stata applicata l'etichetta. L'obiettivo è quello di creare un modello che sia in grado di assegnare ai testi sprovvisti, le corrette etichette. I parametri di valutazione di questi algoritmi sono l'accuratezza, che altro non è che la percentuale dei testi correttamente assegnati; la precisione, che calcola la frazione di documenti che sono rilevanti sul totale di quelli correttamente assegnati e la racall che calcola quanti dei testi rilevanti sono stati ben assegnati.

Naïve Bayes Classifier

I classificatori probabilistici come questo partono dall'assunzione che le parole di un documento si sono generate da un meccanismo probabilistico. Si suppone quindi che la classe $L(d_i)$ del documento ha una relazione probabilistica con le parole contenute al suo interno. La formula Bayesiana calcola la probabilità di appartenenza ad una classe, date le parole contenute nel testo:

$$p(L_c|t_1, ..., t_n) = \frac{p(t_1,...,t_n|L_c)p(L_c)}{\sum_{L_i \in L} p(t_1,...,t_n|L)p(L)}$$

Dove $p(t_1, ..., t_n|L_i)$ è la distribuzione condizionale delle n_i parole data la classe L_i e $p(L_i)$ denota la probabilità che un documento arbitrario appartenga alla classe L_i. In generale l'ordine delle parole all'interno del testo non è rilevante per la corretta assegnazione della classe, si può quindi assumere che una parola appaia, in questo caso, a priori dalla presenza di altre parole. Ciò ci permette di calcolare la probabilità delle parole data una classe con una semplice formula:

$$p(t_1, ..., t_n|L_c) = \prod_{j=i}^{n_i} p(t_j|L_c)$$

Combinando le due formule si ottiene il naïve Bayes Classifier, che necessita di una fase di training nella quale vengono calcolate in base alla presenza delle parole nelle classi tutte le $p(t_j|L_i)$. Nonostante questo metodo sia irrealistico perché assume una totale indipendenza fra le parole, i valori ottenuti dalla classificazione risultano essere buoni.

Nearest Neighbor

Invece di costruire modelli espliciti in questo algoritmo, la classe a cui appartiene il documento viene scelta sulla base della similarità dello stesso con i testi del training set, dei quali si conosce la classe. Il metodo più semplice per calcolare la similarità tra due documenti nel Text Mining è contare le parole che hanno in comune, il valore deve essere normalizzato in base alla lunghezza dei testi. Un altro metodo molto usato per calcolare la similarità denominato "similarità del coseno" ha la seguente formula:

$$S(d_1, d_2) = \sum_{k=1}^{m} w(d_1, t_k) \cdot w(d_2, t_k)$$

Dove d_1 e d_2 sono i due documenti da confrontare, t_k rappresenta la k-esima parola e $w(d_1, t_k)$ il peso della k-esima parola nel documento d_1. Se la parola non è presente in un documento il suo peso è pari a 0. Viene quindi calcolata la similarità con tutti i documenti nel training set e si assegna la classe uguale a quella del testo più simile. La principale pecca dell'algoritmo è lo sforzo di calcolo necessario.

Decision Tree

I classificatori di tipo decision tree si basano su un insieme di regole che in una data sequenza permettono di scegliere la classe di appartenenza. Nella fase di training l'algoritmo usa un approccio di tipo dividi e conquista, partendo dall'intero insieme di documenti, ad ogni passo sceglie la regola che da la migliore separazione fra testi di diverse classi e spezza l'insieme in due parti. Il procedimento viene ripetuto ricorsivamente alle singole parti finché ogni partizione non contiene documenti appartenenti tutti alla medesima classe. Viene così generato un albero di decisioni che contiene nelle sue foglie le assegnazioni alle classi. Questo algoritmo è veloce e scalabile ma, generalmente la scelta finale viene presa sulla base di poche variabili.

Support Vector Machine

L'algoritmo Support Vector Machine o SVM è un algoritmo di classificazione supervisionato. Una singola SVM può separare i testi in due classi: la classe positiva indicata con y = 1 e la classe negativa y = -1. Può essere quindi definito un iperpiano nell'insieme dei vettori in input trovando y = 0 nella seguente equazione:

$$y = f(\overrightarrow{t_d}) = b_0 + \sum_{j=1}^{N} b_j t_{dj}$$

I parametri b_j sono impostati in maniera tale che la distanza, chiamata margine, tra l'iperpiano e il documento positivo e negativo più vicino, sia massima. Si pone così un problema di ottimizzazione quadratica che può essere risolto efficientemente anche per un gran numero di vettori in input. I documenti che hanno distanza dall'iperpiano pari al margine vengono detti vettori di supporto, ogni nuovo documento viene assegnato alla classe positiva se la sua $f(\overrightarrow{t_d}) > 0$ o altrimenti, andrà in quella negativa. La caratteristica più importante di questo algoritmo è che nella fase di apprendimento esso è quasi indipendente dal numero di feature utilizzate, non sarà quindi nella maggior parte dei casi necessario fare una loro selezione per ridurne la dimensionalità. Ciò permette una buona generalizzazione e rende le SVMs uno degli algoritmi oggi più usati nel Text Mining.

1.3.4 Algoritmi per l'estrazione di informazioni

Gli algoritmi di estrazione delle informazioni sono un sottoinsieme degli algoritmi di comprensione del linguaggio naturale. L'obiettivo principale di questi algoritmi è estrarre

parti del testo e assegnare loro specifici attributi. Gli algoritmi si suddividono in vari step che tipicamente sono: tokenizzazione, segmentazione delle frasi, part-of-speech assignement e identificazione delle entità con un nome proprio, come ad esempio nomi di persone, luoghi, aziende. Le frasi devono essere analizzate sintatticamente, interpretate semanticamente ed integrate. Tutte le parti vengono poi immagazzinate all'interno di un database.

Classificazione per l'estrazione di informazioni

Il problema dell'estrazione di informazioni del testo può essere vista come un problema di tagging basato sulle parole. La parola dove l'entità inizia viene taggata con (B), le parole che ne fanno parte con (I) e le parole all'esterno con (O). Ciò viene fatto per ogni entità di interesse ad esempio l'entità persona "di (O) Marco (B) Rossi (I) e (O)". Si otterrà così un problema di classificazione sequenziale per le etichette di ogni parola con le parole che circondano la parola come vettore di caratteristiche. Ogni componente del vettore delle caratteristiche è in pratica un booleano pari a 1, se il pattern è presente, 0 nel caso contrario. Si può quindi scegliere di applicare uno dei metodi di classificazione presentati nella Sezione 1.3.3.

Hidden Markov Models

Uno dei maggiori problemi dei metodi classici di classificazione è che non tengono conto delle etichette appartenenti alle parole circostanti. Ciò può essere fatto usando un modello probabilistico di sequenze di etichette e caratteristiche. Uno dei modelli oggi più usati e quello degli Hidden Markov Models il quale si basa sulle distribuzioni condizionali delle etichette correnti $L^{(j)}$ date le etichette precedenti $L^{(j-1)}$ e la distribuzione delle parole correnti $t^{(j)}$ usando le seguenti formule:

$$L^{(j)} \sim p(L^{(j)}|L^{(j-1)})$$

$$t^{(j)} \sim p(t^{(j)}|L^{(j)}, L^{(j-1)})$$

É necessario un training set di parole correttamente etichettate, l'algoritmo prende in esame tutte le possibili sequenze di parole e ne calcola la probabilità.

1.3.5 Metodi matematici

L'approccio adottato dai metodi matematici è diverso rispetto a quello utilizzato in tutti gli algoritmi precedenti. Viene fatta infatti un'analisi quantitativa e non qualitativa. Il concetto alla base è che i testi possono essere analizzati con strumenti matematici che ne rilevano gli aspetti quantitativi. Questo approccio può essere utilizzato in problemi dove gli approcci tradizionali non portano a buoni risultati come il riconoscimento della

paternità in articoli anonimi, come nell'articolo [8] che verrà brevemente introdotto nella Sezione 1.5.2. I testi, in questo metodo, sono visti come sequenze di simboli: sono simboli, non solo le lettere ma, anche la punteggiatura e gli spazi. Ogni autore quando scrive sceglie i simboli da usare in base a regole probabilistiche che lo distinguono dagli altri. L'obiettivo è quello di trovare tramite l'analisi di queste regole probabilistiche lo scrittore che ha redatto il testo. I metodi più utilizzati in questa branca di algoritmi sono gli n-grammi e l'entropia informativa.

N-grammi

Un n-gramma è una sequenza di n segni alfanumerici presenti in un testo. Il valore di n è pari al numero di simboli che si prendono in esame, più è alto il suo valore più si ottiene una approssimazione simile al testo di partenza. Sequenze di 3 simboli (3-grammi) sono in grado di riprodurre con un discreto grado di precisione le regole di produzione all'interno di un testo. É sufficiente confrontare gli n-grammi di testi diversi per trovare le congruenze. Gli n-grammi all'interno di un testo vengono individuati facendo scorrere, un carattere alla volta, una finestra che può contenere n segni. N-grammi piuttosto lunghi corrispondono a parti direttamente riconoscibili nel testo. La lunghezza di questi n-grammi viene scelta empiricamente, non esistendo oggi criteri ben definiti.

Entropia informativa relativa

L'entropia a livello informativo è stata definita da C. E. Shannon nel 1984, che definisce la quantità di informazione contenuta in un messaggio come il numero minimo di bit necessari a rappresentarlo. L'entropia è il numero di bit necessari per codificare un messaggio, ad esempio, se il messaggio è una sequenza di DNA per la quale si usano quattro caratteri, CGAT, l'entropia è pari a due bit. I programmi di compressione come winzip o winrar fanno uso della ricerca della sequenza minima di bit per rappresentare un testo. Questi programmi per la compressione costruiscono un dizionario del testo in cui, in sintesi, le sequenze di caratteri più usate vengono sostituite da altre più brevi. Il rapporto di peso fra testo originale e testo compresso fornisce una stima precisa dell'entropia del testo; maggiore è la compressione minore è l'entropia. La misurazione dell'entropia può essere usata per confrontare due testi differenti ottenendo una misurazione dell'entropia relativa. Una misura dell'entropia relativa fra due testi può essere ottenuta comprimendo inizialmente entrambi i testi, si passa poi a comprimere un testo con il dizionario dell'altro e confrontando il peso delle due compressioni si trova l'entropia relativa [2].

1.4 Tool per il Text Mining

Esistono oggi tantissimi tool già sviluppati e funzionanti che permettono di applicare il Text Mining. Ne verranno in questa sezione presentati brevemente alcuni.

1.4.1 Kea: Keyphrase automatic extractor

È un tool, come si può comprendere dal nome, per l'estrazione automatica di frasi chiave da documenti [4]. Questo strumento sfrutta, al suo interno, parte della libreria del tool Weka, il quale è molto utilizzato nel Data Mining. Kea usa una procedura di estrazione automatica delle frasi chiave basata sull'algoritmo naïve Bayes. Il concetto di frase chiave è al centro di molti algoritmi di Text Mining ma pochi autori le assegnano ai propri documenti. Le frasi chiave danno una descrizione ad alto livello del testo al quale appartengono e possono essere usate per misurare la differenza fra due testi. L'estrazione di frasi chiave altro non è che un problema di classificazione, nel quale si deve decidere se una frase è rilevante o meno. Come molti algoritmi di classificazione, è necessario avere a disposizione un training set di frasi delle quali si sa se sono chiave o meno. Verrà così generato un modello applicabile ai documenti dai quali si vogliono estrarre le frasi chiave però non tutte quelle che appaiono in un testo hanno le stesse probabilità di essere chiave a priori. Come prima cosa, Kea prende il testo e lo divide in frasi, in base ai vari delimitatori come punteggiatura o parentesi. I caratteri alfanumerici (tranne quelli interni ai periodi) e i numeri vengono eliminati. Kea analizza quali fra tutte le possibili sotto-sequenze, composte da un massimo di tre frasi, siano effettivamente chiavi, quindi elimina le frasi che iniziano o terminano con una parola non rilevante (stop word) e le frasi che consistono solamente in un sostantivo. Nello step successivo, tutte le parole vengono troncate con lo stemmer iterato di Lovins; le parole che dopo questo passo appaiono solo una volta vengono rimosse. Le frasi in se e per se non sono utili, infatti, sono le loro caratteristiche che spesso permettono di distinguere tra il fatto di essere chiave o meno. Nello sviluppo di Kea si è scoperto che due degli attributi di una frase sono importanti per questo scopo. Il valore TFxIDF della frase è un parametro molto comune nei problemi di ricerca di informazioni e misura quanto una frase P possa essere specifica per un documento D:

TFxIDF(P,D) = Pr[che P si trovi in D] x -log Pr[P si trovi in un qualsiasi documento].

La prima probabilità si stima semplicemente contando quante volte la frase P appare nel documento D, mentre la seconda è stimata contando in quanti dei documenti del training set appare la frase P. Il secondo parametro più importante è la distanza dall'inizio del testo della prima apparizione della frase. Il valore si calcola contando le parole che la precedono divise per il numero totale di parole del documento. Prima di applicare il metodo naïve Bayes si applicano algoritmi di discretizzazione dei valori numerici trovati. Si assume quindi con questo metodo che il valori di TFxIDF e la distanza siano indipendenti e si applicano le formule come mostrato nella Sezione 1.3.3.

1.4.2 Maui

È uno strumento che identifica i principali argomenti di un testo. A seconda dell'obiettivo gli argomenti possono essere tags, parole chiave, frasi chiave, descrittori, termini di un

vocabolario, eccetera. Questo strumento al suo interno sfrutta Kea Sezione 1.4.1, la libreria completa di Weka e Jena per leggere i vocabolari in formato RDF. Maui può affrontare i seguenti task:

- assegnamento di termini tramite un vocabolario;
- indicizzazione dei soggetti;
- indicizzazione di argomenti con termini Wikipedia;
- estrazione di frasi-chiave;
- tagging automatico;
- estrazione di terminologia.

Per fare tutto ciò Maui usa un algoritmo che prevede due passi: nella prima fase, genera gli argomenti candidati a diventare principali; nella seconda, filtra i candidati analizzandone le caratteristiche per ottenere il risultato voluto. Nella seconda fase Maui sfrutta diversi parametri per effettuare il filtraggio:

- statistiche di frequenza;
- posizione di occorrenza;
- keypraseness;
- correlazione semantica.

Altri tools che vale la pena citare, senza soffermarsi a descriverli, sono Feature Lens, River Glass ReCon, D2K ma sopratutto DISCUSS.

1.5 Applicazioni del Text Mining

1.5.1 Un modello di Text clustering basato su concetti

Nell'articolo [1], a differenza della maggior parte dei progetti nel campo del Text clustering, invece di analizzare parole o frasi chiave, si analizzano i concetti. I concetti vengono studiati su tre livelli: rispetto alla frase, rispetto al documento e rispetto all'intero insieme di documenti. Nella pratica si analizza la semantica di ogni concetto all'interno della frase e documento. Ogni periodo viene etichettato tramite un algoritmo che trova i verbi e i relativi argomenti che contribuiscono semanticamente al periodo. I concetti possono essere sia parole che frasi intere e dipendono fortemente dalla semantica del periodo. Per ogni nuovo documento che viene immesso nel sistema vengono individuati

i suoi concetti e vengono confrontati con quelli dei documenti già esistenti per decidere il cluster di appartenenza. Nello specifico nella fase di analisi dei concetti, a livello di periodo, gli autori introducono una nuova misura di frequenza detta conceptual term frequency o ctf. Il ctf altro non è che il numero di occorrenze di un concetto denominato c nelle strutture "verbo-argomento" che fanno parte del periodo. Il concetto che appare in più strutture verbo argomento dello stesso periodo è il principale contributore al concetto del periodo. Ad esempio consideriamo il seguente periodo: "Texas and Australia researchers have created industry-ready sheets of materials made from nanotubes that could lead to the development of artificial muscles." l'etichettatore individua tre verbi che rappresentano la struttura semantica e che sono created, made e lead. Ogni verbo ha i propri argomenti:

- (arg1 Texas and Australia researchers) (verb1 have) (arg2 created industry-ready sheets of materials made from nanotubes that could lead to the development of artificial muscles.);
- (arg1 materials) (verb2 made) (arg2 from nanotubes that could lead to the development of artificial muscles.);
- (arg1 nanotubes) (r-arg1 that]) (arg1-mod could) (verb3 lead) (arg2 to the development of artificial muscles.)

Per analizzare invece i concetti sulla base dei documenti è stato introdotto il parametro detto concept-based term frequency o tf che è pari al numero di volte che il concetto appare nell'intero documento. Per estrarre i concetti che permettono di discriminare fra due documenti diversi è stato introdotto anche il parametro detto concept-based document frequency o df che calcola il numero di documenti che contengono il concetto c. In generale vengono prediletti i concetti che sono presenti in pochi documenti in maniera tale da avere più discriminazione. Viene quindi introdotta una nuova misura di similarità sulla base dei parametri precedentemente calcolati. Il suo valore dipende dai seguenti fattori:

- il numero di concetti che fanno matching fra i due documenti (m);
- il numero totale di periodi nei documenti (s);
- il numero di strutture verbo argomento di tutti i periodi (v);
- i valori tf_i per ogni concetto c_i in ogni documento con i = 1, 2, ..., m;
- i valori ctf_i per ogni c_i nel periodo in ogni documento con i = 1, 2, ..., m;
- la lunghezza (l) di ogni concetto in ogni documento d;
- la lunghezza (s) di ogni periodo che contiene un concetto.

Il fattore più importante tra tutti quelli elencati è però il ctf i valori di questo parametro rispetto ai concetti dell'esempio precedentemente fatto sono quelli mostrati in figura.

Row Number	Sentence Concepts	CTF
(1)	texas australia researchers	1
(2)	created	1
(3)	industry ready sheets materials nanotubes lead development artificial muscles	1
(4)	materials	2
(5)	nanotubes lead development artificial muscles	2
(6)	nanotubes	3
(7)	lead	3
(8)	development artificial muscles	3
	Individual Concepts	CTF
(9)	texas	1
(10)	australia	1
(11)	researchers	1
(12)	industry	1
(13)	ready	1
(14)	sheets	1
(15)	development	3
(16)	artificial	3
(17)	muscles	3

Figura 1.1: Valori del ctf dell'esempio

1.5.2 Riconoscimento paternità articoli anonimi

Usando i metodi degli n-grammi e dell'entropia informativa trattati nella Sezione 1.3.5, è possibile creare un sistema che permetta il riconoscimento della paternità di articoli anonimi. Nell'articolo [8] viene presentato un sistema che ha permesso di assegnare la paternità di alcuni scritti anonimi a Gramsci. Entrambi i metodi vengono utilizzati nell'algoritmo finale per decidere se uno scritto è gramsciano o meno. Solo i testi che in entrambi gli algoritmi vengono assegnati a Gramsci sono effettivamente riconosciuti come gramsciani. In questa maniera si è cercato sopratutto di evitare l'assegnazione della paternità a Gramsci degli scritti non gramsciani. Per essere certi del risultato del modello sono stati fatti due test separati nel primo si conoscevano gli autori degli articoli nel secondo no. Nel primo test si sono presi 50 articoli sicuramente gramsciani e 50 articoli di altri scrittori che scrivevano sugli stessi giornali nei medesimi anni. Sfruttando

i risultati di questo test confrontando le differenze tra i testi gramsciani e non, è stato possibile fare un tuning dei parametri tali da massimizzare le attribuzioni corrette e minimizzare quelle erronee. Nel secondo test, denominato cieco, invece sono stati presi in esame quaranta articoli dei quali non si conosceva l'autore. I risultati di questo test sono stati poi presentati all'istituto Fondazione Istituto Gramsci che ne conosceva la paternità. In entrambi i test i risultati ottenuti sono stati ottimi, nel primo 43 dei 50 articoli gramsciani sono stati correttamente assegnati; nel test cieco ben 18 dei 20 articoli gramsciani sono stati riconosciuti. Questi risultati secondo gli autori dell'articolo sono sorprendenti per due motivi:

- gli argomenti trattati negli articoli erano ricorrenti;
- i vari autori dei testi avevano un punto di vista simile rispetto agli argomenti trattati.

In casi come questi secondo gli autori l'approccio da loro adottato è l'unico in grado di individuare con un certo grado di certezza la paternità degli articoli.

Capitolo 2

TuCSoN

TuCSoN, acronimo di Tuple Centres Spread over the Network è un modello di coordinazione per processi distribuiti, autonomi, intelligenti e agenti mobili. TuCSoN sfrutta un media di coordinazione detto "centro di tuple" che altro non è che uno spazio di tuple con la possibilità di specificarne il comportamento. Il modello di TuCSoN si basa sul linguaggio di programmazione Java ed è un progetto open source sotto licenza LGPL.

2.1 Entità del modello

Le entità alla base del modello TuCSoN sono le seguenti:

- gli agenti TuCSoN sono le entità coordinabili;
- i centri di tuple ReSpecT che sono i media di coordinazione;
- i nodi TuCSoN i quali sono che l'astrazione topologica, all'interno della quale si possono trovare i centri di tuple.

Tutte le entità sopra elencate hanno un identificatore che permette di individuarli univocamente. Gli agenti TuCSoN sono le entità pro-attive che agiscono autonomamente, non a causa di modifiche del sistema. I centri di tuple ReSpecT sono invece reattivi cioè reagiscono a stimoli, cambiamenti, che provengono dall'esterno o da essi stessi. Sia gli agenti che i centri di tuple sono sparsi all'interno della rete. Ogni centro di tuple è assegnato ad un nodo della rete che non può cambiare. Gli agenti invece possono essere ovunque nella rete ed interagire con tutti i centri di tuple che si trovano in essa, gli agenti durante la loro vita si possono anche spostare da un nodo ad un altro.

2.2 Specifica degli identificatori univoci

Come accennato in precedenza ogni entità di TuCSoN ha un'identificatore univoco, qui di seguito verranno riportate le specifiche relative agli identificatori delle varie entità.

2.2.1 Nodi

Ogni nodo TuCSoN è univocamente identificato dalla coppia < IdentificatoreRete, NumeroPorta >. L'identificatore di rete può essere l'indirizzo IP o l'entry del DNS relativo del dispositivo che ospita il nodo. Il numero della porta è la porta di rete sulla quale il servizio di coordinazione TuCSoN si mette in ascolto delle richieste di esecuzione delle operazioni. La sintassi astratta per identificare un nodo TuCSoN è la seguente

identificatorerete : numeroporta.

2.2.2 Centri di Tuple

Esistendo in ogni nodo più centri di tuple, è necessario, per identificarne uno, specificare oltre all'identificatore di rete e la porta, un nome univoco. Sono nomi ammissibili per tutti i termini logici di primo ordine di tipo ground. L'identificatore di un centro di tuple sarà quindi il seguente

nomecentro @ identificatorerete : numeroporta.

2.2.3 Agenti

Per quanto riguarda gli agenti, l'identificatore è composto da un UUID(un'identificatore unico universale) che viene assegnato quando l'agente entra nel sistema e da un nome. Come per i centri di tuple, sono nomi ammissibili tutti i termini logici di primo ordine di tipo ground. L'identificatore completo sarà quindi

nomeagente : uuid

2.3 Interazione fra i componenti del modello

Le entità attive del sistema, per interagire con il media di coordinazione, necessitano di operazioni apposite. Queste operazioni sono state codificate all'interno del linguaggio di coordinazione TuCSoN. Gli agenti interagiscono scambiandosi tuple attraverso i centri di tuple usando delle primitive di coordinazione TuCSoN, che assieme, creano il relativo linguaggio. I centri di tuple mettono a disposizione uno spazio condiviso per la comunicazione e uno spazio per specificare il comportamento per la coordinazione. Gli agenti possono leggere, scrivere e consumare tuple dai centri di tuple e sincronizzarsi grazie ad

esse. Le operazioni di comunicazione sfruttano le primitive di coordinazione e i linguaggi di comunicazione: il linguaggio di specifica delle tuple e il linguaggio di specifica dei tuple template. Questi linguaggi dipendono dalla tipologia di centro di tuple dato che TuCSoN si basa sulla specifica dei centri di tuple su ReSpecT, ambedue i linguaggi sono di tipo logico. Ogni atomo Prolog può essere sia una tupla TuCSoN, che un template; i linguaggi, quindi, sono sovrapponibili. Tutte le operazioni TuCSoN sono invocate da un agente su un centro di tuple obiettivo, che si occuperà dell'esecuzione. Le operazioni sono divise in due fasi:

- invocazione la richiesta giunge dall'agente sorgente allo spazio di tuple di destinazione, portando con se tutte le informazioni necessarie per il compimento della stessa;
- completamento il risultato della operazione viene inviato all'agente dal centro di tuple assieme a tutte le informazioni relative alla esecuzione.

La sintassi di un'operazione (op) su u dato centro di tuple (tid) è la seguente

tid ? op

riprendendo la sintassi dell'identificativo di un centro di tuple la forma completa è

nomecentro @ identificatorerete : numeroporta ? operazione

TuCSoN prevede però delle configurazioni di default che permettono una sintassi più abbreviata per l'invocazione delle operazioni. Ad esempio il valore di default per il numeroporta è 20504, per ogni nodo TuCSoN inoltre viene creato un centro denominato default. È quindi possibile per un'agente invocare un'operazione usando la seguente sintassi

@ identificatorerete ? operazione

l'operazione verrà così demandata al centro di tuple di default ospitato sul nodo indicato dall'identificatorerete alla porta di default. Lo spazio globale di coordinazione in TuCSoN viene definito come l'intero assieme di centri tuple raggiungibili nella rete. Lo spazio locale è invece definito come l'insieme dei centri di tuple che risiedono su tutti i nodi TuCSoN che hanno lo stesso identificatorerete. É possibile quindi per un agente che risiede sullo stesso dispositivo del centro di tuple invocare operazioni su esso nella seguente maniera

nomecentro @ numeroporta ? operazione.

2.3.1 Operazioni di base

Il linguaggio di coordinazione TuCSoN mette a disposizione nove operazioni di base che sono:

- **out(tupla)** immette la tupla nel centro di tuple bersaglio, se l'operazione è andata a buon fine, al completamento la tupla viene rinviata indietro;
- **rd(templatetupla)** si cerca nel centro di tuple una tupla che faccia matching con il template. Se si trova una tupla che soddisfi il matching questa viene ritornata al completamento; in caso contrario l'esecuzione viene sospesa e riprenderà solo quando nel centro sarà presente una tupla che lo soddisfi;
- **in(templatetupla)** il comportamento è simile alla rd(templatetupla), l'unica differenza è che questa volta la tupla non viene semplicemente letta ma viene rimossa dal centro di tuple;
- **rdp(templatetupla)** si cerca nel centro di tuple una tupla che faccia matching con il template; se si trova una tupla che soddisfi il matching questa viene ritornata al completamento; in caso contrario l'esecuzione fallisce e rinvia al mittente il templatetupla;
- **inp(templatetupla)** il comportamento è simile alla rdp(templatetupla). La differenza è che se l'operazione va a buon fine la tupla viene rimossa dal centro di tuple;
- **no(templatetupla)** controlla se nel centro di tuple esiste una tupla che fa match con il templatetupla, se non esiste quando l'operazione viene servita, l'esecuzione ha successo e viene rinviato al mittente il templatetupla; in caso contrario l'esecuzione viene sospesa finché non esistono più tuple che matchano;
- **nop(templatetupla)** nel caso in cui quando viene servita l'operazione, non esistono tuple che matchano, il comportamento è uguale a quello dell'operazione no(templatetupla). La differenza si ha se l'operazione non ha successo immediato, in questo caso l'esecuzione fallisce e viene ritornata la tupla che fa match;
- **get** legge tutte le tuple presenti in un dato centro di tuple e le ritorna sotto forma di lista; se non ci sono tuple viene rinviata all'agente la lista vuota;
- **set(tuple)** riscrive il centro di tuple inserendo tutte le tuple contenute nella lista tuple; al completamento viene ritornata la lista di tuple.

2.3.2 Operazioni avanzate

Operazioni bulk

Per ottenere migliori prestazioni in alcuni problemi di coordinazione, come quelli che prevedono la gestione di più di una tupla per operazione, sono necessarie primitive di tipo "bulk". Queste operazioni invece di ritornare una delle tuple che matcha al completamento, ritornano la lista di tutte le tuple che matchano. Se non c'è nessuna tupla che matcha ritornano una lista vuota. Il linguaggio di coordinazione di TuCSoN mette a disposizione quattro primitive di tipo bulk: rd_all, out_all, in_all e no_all.

Operazioni uniformi

Se fosse necessario inserire meccanismi probabilistici all'interno del sistema di coordinazione per ottenere un comportamento stocastico, sarebbe utile garantire che le tuple ritornate vengano scelte in maniera non deterministica tra quelle che matchano il template. La distribuzione deve essere inoltre uniforme; TuCSoN mette a disposizione del programmatore sei primitive uniformi: urd, urdp, uin, uinp, uout, uoutp, uno e unop.

2.4 Strumenti di programmazione TuCSoN

TuCSoN è un middelware che si basa sul linguaggio Java e su Prolog. Sono stati messi a disposizione del programmatore vari strumenti utili per la programmazione: classi Java per la specifica degli agenti e librerie Prolog che permettono di estendere tuProlog con le primitive di coordinazione di TuCSoN. Il middelware mette inoltre a disposizione delle API per permettere di utilizzare i concetti alla base di TuCSoN in programmi Java. Le principali classi a disposizione sono:

- **TucsonAgentID** fornisce metodi per ottenere un identificatore di un agente TuCSoN e permette l'accesso ad alcuni dei suoi campi;
- **TucsonMetaACC** permette di ottenere un agente con ACC strumento obbligatorio per interagire con un centro di tuple TuCSoN;
- **TucsonTupleCentreID** serve per specificare un identificativo di un centro di tuple e permette di accedere ai suoi vari campi;
- **ITucsonOperation** mette a disposizione metodi per accedere al risultato di un'operazione;
- **AbstractTucsonAgent** classe astratta dalla quale un utilizzatore può partire per specificare i propri agenti TuCSoN;

- **SpawnActivity** classe di base astratta che permette la specifica di attività TuCSoN generate da una specifica operazione;
- **Tucson2PLibrary** consente agli agenti tuProlog di accedere alla infrastruttura di TuCSoN;
- **LogicTuple** dispone di metodi per creare un template e accedere ai suoi argomenti;
- **TupleArgument** rappresenta gli argomenti delle tuple TuCSoN;
- **TucsonNodeService** permette di creare o distruggere un nodo TuCSoN.

2.5 Meta-coordinazione

Il linguaggio di meta-coordinazione di TuCSoN permette agli agenti di programmare i centri di tuple ReSpecT eseguendo operazioni su di essi. TuCSoN fornisce primitive che permettono di leggere, inserire e rimuovere le tuple di specifica ReSpecT dai centri di tuple. Grazie a queste primitive è anche possibile coordinarsi in base alle tuple di specifica ReSpecT. Come per le tuple, sono previsti due linguaggi separati: il linguaggio di specifica e il linguaggio di specifica dei template. Questi dipendono dal tipo di spazi di tuple adottati da TuCSoN; nel nostro caso dato che il medium di coordinazione è specificato in linguaggio ReSpecT ambedue si basano su di esso. Come per le operazioni normali, anche quelle di meta-coordinazione possono essere divise in due fasi: invocazione e completamento. Sono sempre gli agenti che richiedono l'esecuzione delle primitive di meta-coordinazione ad un dato centro di tuple, TuCSoN mette a disposizione nove primitive di meta-coordinazione: rd_s, in_s, out_s, rdp_s, inp_s, no_s, nop_s, get_s e set_s. Il comportamento delle primitive è lo stesso di quelle di base, la differenza risiede nel fatto che queste operazioni lavorano e ritornano tuple di specifica del tipo reaction(E, G, R), invece di quelle normali.

2.6 Tools

L'infrastruttura TuCSoN mette a disposizione due tools per i programmatori: il primo è il Command Line Interpreter o CLI, una shell che permette di operare come un agente sui centri di tuple. Il secondo è l'inspector che è una GUI che permette di monitorare i centri di coordinazione TuCSoN e la virtual machine ReSpecT. Specificando l'identificatore di rete il numero di porta e il nome del centro di tuple è possibile controllare

- lo stato dello spazio di tuple;
- lo stato delle tuple nello spazio di specifica ReSpecT;

- la lista delle operazioni sospese;
- l'insieme delle reazioni ReSpecT lanciate a causa delle operazioni TuCSoN.

Capitolo 3

Molecules of Knowledge MoK

MoK è un modello di coordinazione ispirato dalla biochimica che permette l'auto-organizzazione della conoscenza [11]. La motivazione alla base del modello MoK è l'idea che la conoscenza si deve autonomamente aggregare e diffondere fino, a raggiungere il suo consumatore senza essere ricercata esplicitamente. Il modello si ispira alla biochimica dato che ormai si riconosce ai sistemi biochimici, la capacità di raggiungere un "ordine partendo dal caos", grazie alla loro adattabilità e all'auto-organizzazione.

3.1 Concetti principali alla base del modello

Il modello MoK si basa su diversi concetti presi dalla biochimica [13] come:

- gli atomi: sono la parte di conoscenza più piccola; un atomo contiene informazioni provenienti da una sorgente e appartiene ad un compartimento nel quale risiede e dove è soggetto a date leggi;
- le molecole: sono le unità dove vengono aggregate le informazioni, all'interno delle molecole si trovano informazioni (atomi) tra loro correlate;
- gli enzimi: emessi dai catalizzatori, gli enzimi influenzano le reazioni all'interno di MoK in maniera tale da modificare le dinamiche di evoluzione della conoscenza del compartimento per andare incontro a ciò che interessa all'utente;
- le reazioni: sono le leggi biochimiche che regolano il compartimento; hanno un certo rate con cui vengono applicate. Attraverso le reazioni vengono specificate regole per l'aggregazione, la diffusione e il decadimento delle informazioni;
- i compartimenti: rappresentano il luogo concettuale che contiene tutte le entità di MoK è sulla base di questo concetto che si possono definire la località e la vicinanza;

- le sorgenti: sono le origini della conoscenza che viene continuamente iniettata ad un dato rate sotto forma di atomo all'interno del relativo compartimento;
- i catalizzatori: rappresentano i consumatori-utilizzatori o prosumer di conoscenza, emettono enzimi che rappresentano le loro azioni che andranno ad impattare sulle dinamiche della conoscenza all'interno del proprio compartimento, in modo tale da incrementare la probabilità che giungano informazioni per lei/lui rilevanti.

I concetti pilastri del modello sono gli atomi, le molecole, gli enzimi e le reazioni che verranno analizzati un po' più in dettaglio.

3.1.1 Gli atomi

Gli atomi, oltre a contenere al loro interno la conoscenza devono avere anche delle informazioni rispetto al contesto da dove questa è presa come l'origine del contenuto in maniera tale da mantenere il significato originale. Un atomo all'interno del modello MoK è una tripla del tipo $atom(src, val, attr)_c$ dove "src" identifica in modo non ambiguo la sorgente, "val" è il corrente pezzo di conoscenza e "attr" è un attributo relativo alla conoscenza contenuta che permette una sua migliore comprensione. Questo attributo generalmente si basa su una ontologia ben definita o su un vocabolario. L'ultimo termine la "c" a pedice, rappresenta la corrente concentrazione dell'atomo, cioè il numero di atomi dello stesso tipo all'interno del compartimento.

3.1.2 Le molecole

Le molecole sono degli aggregatori stocastici guidati dall'ambiente di atomi, il loro obiettivo è quello di estrapolare relazioni semantiche fra gli atomi e, sulla base di queste, aggregarli, creando così nuova conoscenza. Ogni molecola è semplicemente un insieme di atomi non ordinati tra di loro semanticamente correlati. La struttura di una molecola nel modello MoK è la seguente $molecule(Atoms)_c$ dove c rappresenta la concentrazione della molecola e Atoms la collezione di atomi correlati dalla molecola.

3.1.3 Gli enzimi

Gli enzimi sono una delle entità chiave del modello MoK perché interpretano le azioni sulla conoscenza da parte del prosumer come feedback positivi in modo tale da incrementare, all'interno del compartimento, la concentrazione delle molecole correlate a quelle di interesse. La prima rappresentazione degli enzimi all'interno di MoK era del tipo $enzyme(Atoms)_c$, dove "c" è la concentrazione dell'enzima e "Atoms" è un insieme di atomi ai quali in qualche modo il prosumer è interessato.

3.1.4 Le reazioni

All'interno del modello MoK sono presenti diverse reazioni di base. Dato che MoK è un modello per la correlazione semantica di informazioni, la reazione più importante è quella che, dati due atomi, permette di calcolarne la correlazione, la quale può essere un semplice booleano o un valore compreso tra 0 e 1, e che è definita come $mok(atom_1, atom_2)$. Un'altra relazione alla base del modello e quella di aggregazione, la quale date due molecole le aggrega ed è definita come:

$$molecule(Atoms_1) + molecule(Atoms_2) \xrightarrow{r-agg} molecule(Atoms_1 \cup Atoms_2) + residual(Atoms_1, Atoms_2)$$

Nella relazione, r-agg rappresenta la frequenza con la quale la reazione viene applicata, inoltre è necessario che esista almeno una coppia di atomi $atom_1$, $atom_2$, che appartengono ai rispettivi insiemi $Atoms_1$, $Atoms_2$, con una certa correlazione sulla base della regola semantica sopra citata. In residual vengono messi tutti gli atomi di $Atoms_1$ e $Atoms_2$ che non sono stati aggregati. Grazie alla reazione di rinforzo viene attuato il feedback positivo, la reazione consuma un enzima e produce una molecola/atomo relativa

$$enzyme(Atoms_1) + molecule(Atoms_2)_c \xrightarrow{r-reinf} molecule(Atoms_2)_{c+1}$$

dove r-reinf rappresenta la frequenza con la quale la reazione viene applicata. L'enzima $enzyme(Atoms_1)$ e la molecola $molecule(Atoms_2)_c$ devono essere all'interno dello stesso compartimento, con $c \neq 0$ e $mok(atom_1, atom_2)$ deve essere valida per $atom_1 \in Atoms_1$, $atom_2 \in Atoms_2$. Per seguire la metafora biochimica, le molecole devono svanire con il passare del tempo, diminuendo pian piano la loro concentrazione secondo una legge di decadimento

$$molecule(Atoms)_c \xrightarrow{r-decay} molecule(Atoms)_{c-1}$$

É stato necessario implementare anche un meccanismo per la diffusione delle molecole fra i vari compartimenti vicini. La relativa reazione può essere rappresentata come segue

$$\| Molecules_1 \bigcup molecule_1 \|_{\sigma^i} + \| Molecules_2 \|_{\sigma^{ii}} \xrightarrow{r-diff} \| Molecules_1 \|_{\sigma^i} + \| Molecules_2 \bigcup molecule_1 \|_{\sigma^{ii}}$$

dove la struttura $\|\|_\sigma$ significa all'interno del compartimento σ. Come nelle reazioni precedenti r-diff e la frequenza della reazione in esame e σ^i e σ^{ii} sono compartimenti vicini.

3.2 Mapping di MoK su TuCSoN

TuCSoN [3] è una infrastruttura che usa spazi di tuple programmabili detti tuple centers. Il comportamento di ogni spazio di tuple può essere specificato grazie al linguaggio logico chiamato ReSpecT [14]. Dato che i centri di tuple basati su ReSpecT sono ottimi per rappresentare la conoscenza in generale, MoK è stato costruito su TuCSoN. È stato semplice implementare gli elementi base della coordinazione biochimica sfruttando TuCSoN. L'evoluzione stocastica ispirata alla chimica è stata ottenuta realizzando l'algoritmo di simulazione esatto di Gillespie [5] come specifica di ReSpecT. La rappresentazione delle leggi biochimiche attraverso le tuple si è ottenuta interpretando tutte le tuple come parte di un sistema biochimico e usando dei template per rappresentare i reagenti delle reazioni. Il mapping finale ottenuto è il seguente:

- gli individui del sistema atomi, molecole, enzimi sono stati mappati sulle tuple logiche;
- le astrazioni spaziali come compartimenti/catalizzatori altro non sono che i centri di tuple di TuCSoN;
- infine le reazioni biochimiche sono state implementate come reazioni ReSpecT.

Ogni centro di tuple è un compartimento di MoK che viene programmato per funzionare come una macchina virtuale biochimica, implementando il simulatore di Gillespie, dove le reazioni biochimiche vengono espresse tramite reazioni ReSpecT.

3.3 MoK e Behavioural Implicit Comunication

I sistemi socio-tecnici, cioè i sistemi dove l'interazione umana gioca il ruolo principale, sono sempre più complessi e spesso sono anche sistemi che si rapportano con quantità di informazioni notevoli. Le interazioni alla base di questi sistemi sono spesso fonte di non determinismo che, per essere gestito, necessita di nuovi modelli di coordinazione. Negli ultimi anni si sta provando a rapportarsi a questi sistemi sfruttando metafore ispirate alla natura. In natura esistono infatti molti sistemi di questo tipo che riescono a raggiungere un ordine a partire dal caos sfruttando l'auto-organizzazione e l'adattabilità. Nei sistemi naturali generalmente non c'è interazione diretta fra gli individui ma, questa avviene attraverso l'ambiente il quale tiene traccia di ciò che gli individui hanno fatto, a questo tipo di interazione è stato associato il concetto di Stigmergia. Nei sistemi umani, le traccie lasciate diventano segni, e il concetto di stigmergia si evolve in stigmergia cognitiva, avendo gli individui del sistema abilità cognitive che possono essere sfruttate all'interno del processo di coordinazione. Si fa un ulteriore passo oltre la stigmergia cognitiva se si pensa alla coordinazione del sistema sulla base delle azioni in toto, piuttosto che solo sugli effetti che queste hanno sull'ambiente. Questo tipo di coordinazione prende il nome

di Behavioural Implicit Comunication o BIC. Un sistema di coordinazione che intende modellare questi meccanismi naturali deve avere certe caratteristiche:

- stigmergia:
  - deve essere possibile registrare le tracce;
  - devono essere presenti meccanismi che fanno da controparte all'emissione come l'evaporazione;
  - le tracce devo essere visibili per gli altri individui;
  - l'ambiente deve avere una topologia che permetta di descrivere il concetto di località.
- stigmergia cognitiva:
  - sono necessari strumenti di supporto per l'interpretazione dei segnali da parte degli individui, come le ontologie;
  - gli individui devono essere intelligenti.
- BIC:
  - le azioni che esprimono il comportamento di un individuo devono essere visibili agli altri individui che si trovano nello stesso ambiente;
  - anche le proprietà relative alle azioni, come "chi, dove, quando" è stata fatta l'azione devono essere visibili.

Il modello MoK supportava già molte delle caratteristiche richieste ma è stato necessario apportare qualche piccola modifica [12]. Ad esempio, per tenere traccia delle informazioni contestuali riguardanti le interazioni tra agenti, è stato sufficiente associare ad ogni enzima un descrittore $descriptor(\sigma)$ del compartimento dove l'enzima è stato rilasciato che verrà ora rappresentato come $enzyme(\sigma, Atoms)_c$. Il descrittore può contenere ogni meta-informazione relativa al compartimento utile per capire l'azione che l'ha generato come ad esempio: timestamp di emissione, luogo di emissione, eccetera. Per rendere gli enzimi disponibili ad essere percepiti dagli altri individui all'interno dell'ambiente MoK, è sufficiente permettere la diffusione degli enzimi fra compartimenti vicini. Mentre per rendere gli enzimi più simili a tracce di azioni invece che ad azioni nell'insieme, si è deciso di produrre un enzima "morto" per gli enzimi che si trovano all'interno di reazioni di rinforzo. Ciò permette di distinguere fra tracce di azioni e azioni vere e proprie, le reazioni di rinforzo cambieranno quindi la loro forma diventando:

$$enzyme(\sigma, Molecule_1) + Molecule_1^c \xrightarrow{r-reinf} Molecule_1^{c+1} + dead(enzyme(\sigma, Molecule_1))$$

Per quanto riguarda le altre reazioni la diffusione potrà essere applicata sia ad entrambe le tipologie di enzimi, mentre il decadimento potrà essere applicato solo agli enzimi "morti".

3.4 Applicazioni che sfruttano MoK

MoK è un modello per l'organizzazione della conoscenza in generale ma può essere sfruttato anche in applicazioni domain specific. Il modello è stato sfruttato per organizzare notizie [11, 13] che pervadono la rete.

3.4.1 MoK-News

In questo caso di studio, il modello MoK è stato utilizzato con l'intento di gestire notizie. Oggi le notizie pervadono la rete, sono una ottima fonte di informazione e per questo sono state scelte come primo caso di studio dagli sviluppatori del modello. Esistono due standard principali in utilizzo nel campo delle notizie che facilitano la loro organizzazione, questi standard sono sviluppati dall'organizzazione IPTC e sono denominati NewsML e News Industry Text Format o NITF. Lo standard NewsML sfrutta al suo interno XML e altro non è che un linguaggio per il tagging delle notizie che non si concentra solo sul contenuto della notizia ma anche sui suoi meta-dati. NewsML crea un insieme di Controlled Vocabularies (CVs) che vengono raccolti assieme in NewsCodes in modo tale da rappresentare i concetti e categorizzare gli oggetti delle notizie in maniera consistente. Anche NITF sfrutta al suo interno XML ma, lo usa per arricchire il contenuto delle notizie, aggiungendo la descrizione per delle caratteristiche tipiche delle notizie. Alcune di queste caratteristiche sono quelle che rispondono alle seguenti domande:

- chi detiene i diritti di copyright? come può essere ripubblicata la notizia?
- di quali soggetti, organizzazioni, eventi si occupa la notizia?
- quando la notizia è accaduta, quando è stata pubblicata?
- dove è accaduta la notizia? dove può essere pubblicata? dove è stata scritta?

Dato che NITF non offre nessuna forma di tagging inline per aggiungere informazione alla notizia, i due standard possono essere utilizzati insieme per avere informazioni ancora più dettagliate sulla notizia. I vari tag assieme al contenuto sono rappresentati all'interno del modello come atomi, che possono essere raggruppati in molecole. Un atomo di MoK-News ha una struttura $atom(src, val, sem(tag, catalog))_c$ dove src è l'uri della sorgente della notizia, val è il contenuto della notizia, tag è il tag nel formato NITF o NewsML e infine catalog è l'uri del NewsCode relativo. Il contenuto di un atomo è principalmente individuato dalla coppia "val-tag"; grazie al contributo del tag ora il contenuto di val è semanticamente non ambiguo. L'aggregazione di due atomi ha luogo se esistono fra di loro tag delle correlazioni semantiche. Le molecole diventano quindi un repository di notizie auto-organizzate. L'obiettivo è quello di portare al prosumer le notizie che a lui più interessano analizzando le azioni da lui effettuate in precedenza.

Capitolo 4

Text Mining e MoK

Dopo un'attenta analisi dei vari articoli riguardanti il Text Mining, siamo arrivati alla conclusine che le varie tecniche possono essere utilizzate sia all'interno del modello MoK, che ai sui suoi confini, cioè su tutte le fonti di informazioni grezze. La parte sicuramente più rilevante e nella quale si possono applicare algoritmi quasi di ogni genere è nel preprocessing delle fonti di conoscenza. Ma anche all'interno del modello possono essere sfruttate tecniche prese in prestito dal Text Mining.

4.1 Preprocessing delle fonti di conoscenza

In tutti gli algoritmi di Text Mining studiati è necessaria una fase di ricerca, di parole, frasi e concetti che contraddistinguono i ducumenti, testi o news su cui vengono applicati gli algoritmi. Il migliore elemento del testo per discriminare i vari testi è secondo me l'astrazione di concetto, come esposta nell'articolo [1]. L'autore introduce la metafora di concetto che può essere riferita ad una singola parola, una frase o una sua parte. L'entità concetto dipende fortemente dalla semantica del periodo, ogni periodo del documento viene diviso in strutture del tipo "verbo-argomento". Ogni argomento è un concetto, viene quindi calcolata la frequenza con la quale il concetto appare da vari punti di vista. Inizialmente si calcola la frequenza all'interno delle strutture che compongono il periodo, si trovano così i concetti più importanti dello stesso. Viene poi calcolata la frequenza dei vari concetti all'interno del testo di appartenenza e infine la frequenza nel corpus dei testi nel complesso. É possibile in base a questi dati scegliere i concetti più significativi per il documento e, sulla base di questi, confrontare due fonti di informazione e calcolarne la correlazione. Nello stesso articolo viene proposto un algoritmo di clustering sulla base dei parametri calcolati. Se si volessero invece trovare le frasi chiave di un documento non strutturato si potrebbe comodamente ricorrere all'utilizzo dello strumento Kea (Sezione 1.4.1). Questo strumento sfrutta alcuni degli algoritmi che si trovano all'interno del sistema Weka (Waikato Environment for Knowledge Analysis) nato per applicazioni

di Data Mining. Utilizzando questi, in combinazione con indicatori che generalmente si utilizzano nel campo della ricerca delle informazioni, come il TFxIDF, classifica le frasi all'interno del testo assegnato in due classi: chiave e non chiave. Tutto ciò viene fatto in maniera automatica, è sufficiente fornire allo strumento un insieme di documenti dei quali si conoscono già le frasi chiave, che fungeranno da training set. Sulla base di questi documenti, viene generato un modello che viene poi utilizzato per discriminare le frasi chiave da quelle normali di un qualsiasi documento. Se si volesse invece caratterizzare un testo sulla base delle parole chiave, sarebbe necessaria una fase di tokenizzazione grazie alla quale si rimuove dal testo tutto ciò che non è di interesse (come punteggiatura, spazi, numeri, eccetera). Si ottiene così una lista delle parole che si trovano all'interno del testo che vengono raccolte in un dizionario. Questo elenco di parole viene poi ridotto grazie ai processi di filtraggio, lemmatizzazione e stemming. Nella fase di filtraggio vengono rimosse parole non rilevanti, ad esempio si può applicare il metodo della rimozione delle stop word. Che sono parole generalmente non rilevanti come congiunzioni, articoli, avverbi, eccetera. Esistono in internet varie fonti che forniscono elenchi di stop word per le varie lingue. Nel processo di lemmatizzazione si cerca di portare tutte le forme verbali alla forma infinita e i sostativi tutti alla forma singolare, però questo passo richiede generalmente molto tempo, è quindi preferibile usare gli stemmer essendo questi molto meno onerosi da questo punto di vista. L'obiettivo degli stemmer è ottenere le forme basi delle parole e raggruppare quelle con significato simile in uno stesso insieme detto stem, uno degli stemmer più usato e citato è quello ideato da Porter (Sezione 1.3.1). Questo stemmer sfrutta diverse regole divise in gruppi e applicate in vari step, le quali sono del tipo (condizione) suffisso1 $\rightarrow$ suffisso2. Se la condizione vale per la parte di parola che precede il suffisso1, allora si sostituisce il suffisso1 con il suffisso2. Il suffisso2 può essere anche vuoto, nel caso si voglia semplicemente eliminare il suffisso1 invece che sostituirlo. Ogni parola può essere vista come una sequenza del tipo $[C](VC)^m[V]$ dove C rappresenta una sequenza di consonanti e V una sequenza di vocali. Una consonante c è una lettera di una parola diversa da A, E, I, O, U, Y preceduta da una consonante, tutte le altre lettere sono vocali v. Le condizioni invece sono generalmente possono contenere diversi parametri:

- m detto anche misura della parola;
- *G il tronco della parola finisce con g;
- *v* il tronco della parola contiene una vocale;
- *d il tronco della parola finisce con una doppia consonante;
- *o il tronco della parola finisce con una sequenza cvc con la seconda c diversa da W, X e Y.

Per ridurre ulteriormente la dimensionalità dei vettori delle parole rappresentative si può ricorrere ad algoritmi di selezione, un buon metodo è quello di selezionarli sulla base della loro entropia, come definita nell'articolo [10]. Grazie a questo metodo vengono selezionate le parole chiave che meglio permettono di discriminare un testo da un altro.

4.2 Tecniche utilizzabili all'interno del modello

All'interno del modello, in primis, ritengo sia utile testare varie metodologie per calcolare la correlazione fra due atomi di conoscenza. In questo modo si può controllare l'evoluzione del sistema al variare della funzione di correlazione usata; sarà così poi possibile analizzare quanto la scelta della funzione di correlazione influenzi l'andamento del sistema. La stragrande maggioranza delle funzioni di correlazione si basano sulla rappresentazione "vector space" dei documenti da analizzare. Con questa metodologia di rappresentazione tutti gli scritti vengono rappresentati da un vettore composto dal singolo peso di ogni elemento del vocabolario (parola) all'interno dell'articolo. I pesi sono stati calcolati sulla base della formula ad-hoc presentata nell'articolo [9], che combina il risultato dalla funzione di Okapi e la normalizzazione di quanto ottenuto dal calcolo del valore idf della INQUERY. Ogni peso viene calcolato in base alla seguente funzione:

$$v_{i,j} = \frac{tf_{i,j} \log(\frac{colsize+0.5}{docf_i})}{tf_{i,j}+0.5+1.5\frac{doclen_j}{avgdoclen}\log(colsize+1)}$$

dove $v_{i,j}$ è il peso dell'i-esimo termine del vocabolario all'interno del documento j-esimo, $tf_{i,j}$ è il numero di volte che il termine appare nel documento, $docf_i$ il numero di documenti in cui è presente il termine, $doclen_j$ è il numero di termini del documento, $avgdoclen$ è il numero medio di termini per i documenti della collezione. Uno dei metodi più citati ed usati è la similarità del coseno che sfrutta la seguente equazione [1]:

$$Sim(C_i, C_j) = \frac{\sum_{k=1}^{n} c_i(t_k) \cdot c_j(t_k)}{\sqrt{\sum_{k=1}^{n} c_i(t_k)^2 \sum_{k=1}^{n} c_j(t_k)^2}}.$$

Dove C_i e C_j sono i vettori dei pesi assegnati ai vari termini per il documento i-esimo e j-esimo $C_j = \{c_j(t_1), c_j(t_2), ..., c_j(t_n)\}$ e $t_1, t_2, ..., t_n$ sono i termini. Un altro metodo è la differenza media dei quadrati che da più peso alle grandi differenze fra i pesi [6]:

$$d(C_i, C_j) = \frac{\sum_{k=1}^{n} (c_i(t_k) - c_j(t_k))^2}{n}.$$

Un ulteriore metodo per calcolare la distanza fra due documenti è la distanza euclidea che può essere espressa come nell'articolo [7]:

$$dist(C_i, C_j) = \sqrt{\sum_{k=1}^{n} |c_i(t_k) - c_j(t_k)|^2}.$$

4.3 Algoritmi non utilizzabili

A mio parere non ritengo utili all'interno di MoK gli algoritmi matematici (Sezione 1.3.5) a meno che non si voglia ottenere una organizzazione della conoscenza sulla base dell'autore. Questi metodi a differenza di tutti gli altri algoritmi di Text Mining usano un approccio diverso e particolare, viene infatti effettuata un'analisi qualitativa e non quantitativa dei testi. L'analisi viene effettuata usando dei metodi di derivazione matematica, ogni parte dell'informazione è considerata come un simbolo, il testo viene quindi analizzato in dettaglio alla ricerca di schemi di scrittura che variano tra un autore e l'altro. Ogni scrittore, infatti nell'intento di scrivere, usa probabilisticamente termini e segni in maniera completamente diversa dagli altri. Due metodi matematici: sono l'entropia informativa relativa e la tecnica degli n-grammi. Una semplice misura dell'entropia informativa relativa, si può ottenere confrontando la differenza di dimensioni nella compressione di un testo usando un qualsiasi programma. Gli attuali strumenti di compressione, infatti, cercano di ridurre le dimensioni dei testi sostituendo alle sequenze di caratteri più ricorrenti, altre più brevi. Si crea così un vocabolario di corrispondenza fra le due sequenze. Comprimendo uno dei due testi usando il suo vocabolario e successivamente ripetendo la compressione con il vocabolario dell'altro, si ottengono due compressioni di dimensioni diverse. La differenza di dimensioni fra le due compressioni è una buona stima dell'entropia informativa relativa. La tecnica degli n-grammi, invece, cerca di trovare schemi di scrittura ricorrenti all'interno del testo confrontando le sequenze di n simboli (lettere, spazi, punteggiatura) relative a due testi. Ogni n-gramma viene ricavato dal testo prendendo una finestra di n simboli che scorre un simbolo alla volta. Più la dimensione di n è grande, più l'n-gramma relativo conterrà una parte facilmente riconducibile al testo, ad esempio parole intere. Il valore n varia da applicazione ad applicazione, non esistono metodi sperimentali per trovare un suo valore ottimale, quindi viene scelta empiricamente. Sia gli n-grammi che l'entropia informativa relativa, analizzano la struttura dei testi e non i loro contenuti per questo motivo li ritengo poco adatti per l'utilizzo nell'aggregazione della conoscenza in generale. Ritengo inoltre poco utili, a meno che non si vogliano avere informazioni molto dettagliate e specifiche riguardo all'informazione, l'utilizzo degli algoritmi per l'estrazione di informazioni (Sezione 1.3.4). Questi algoritmi hanno infatti come scopo principale quello di estrarre da testi non strutturati informazioni relativi a nomi propri di entità come persone, luoghi, imprese, eccetera. Le informazioni ottenute a mio avviso non sono quelle secondo le quali è meglio aggregare conoscenza, ciò nonostante, in alcuni casi specifici potrebbero essere utilizzabili. Se ad esempio per qualche motivo si volessero aggregare informazioni che riguardano uno stesso soggetto, questi metodi sarebbero sicuramente i più adatti.

Capitolo 5

Esperimenti

5.1 Introduzione

La domanda a cui gli esperimenti da me svolti intendono rispondere è: come possono essere sfruttate le tecniche di Text Mining all'interno di sistemi che hanno come obiettivo principale l'auto-organizzazione della conoscenza? Tutti i test hanno come obbiettivo quello di ottenere un insieme di cluster a partire da articoli scientifici in lingua inglese. Come infrastruttura di coordinazione si è scelto di utilizzare TuCSoN e non MoK. Questa scelta deriva da vari motivi; in primis perché fra i due è il sistema più generale. Un altro aspetto che ha portato a questa scelta è il fatto che, essendo MoK basato su TuCSoN, non dovrebbe essere difficile tradurre quanto fatto in TuCSoN e applicarlo all'interno di MoK. Gli algoritmi di clustering realizzati sono due: uno da me ideato e un altro, che è un'implementazione di un algoritmo preso da un articolo di letteratura scientifica. Del secondo sono state implementate due varianti: una prevede un calcolo poco distribuito (solo pochi agenti) mentre l'altra prevede una fortissima distribuzione del calcolo (un agente per articolo). Utilizzando questi algoritmi sono stati realizzati vari esperimenti, in modo tale da poter tarare i parametri, osservare la dipendenza degli algoritmi da questi e confrontali al fine di individuare i pregi e i difetti delle varie tecniche utilizzate.

5.2 Analisi

Durante l'analisi del problema, si è ritenuto necessario analizzare i vari formati di rappresentazione degli articoli così da poter scegliere quale tipologia rappresentativa utilizzare nei vari contesti applicativi. Esistono oggi una grande quantità di formati, nel progetto si è deciso di studiare solo tre modelli: il formato BibTex, l'abstract e l'articolo nella sua interezza, questi sono infatti quelli più usati nella letteratura scientifica. Si è pensato quindi inizialmente di realizzare vari esperimenti, sfruttando un semplice algoritmo di clustering, per scegliere quale di queste rappresentazioni fosse la migliore da utilizzare

all'interno del contesto di interesse. Una volta presa questa decisione sono stati messi a punto altri algoritmi di clustering più complessi. Come base dati per i test sono stati utilizzati gli articoli presenti sul portale APICe che contiene un vasto numero di articoli relativi alla ricerca svolta nella seconda facoltà di ingegneria di Cesena. Sono stati scelti circa cento articoli del portale APICe, che coprono l'ultimo periodo di ricerca dal 2010 al 2013.

5.3 Implementazione

5.3.1 Metodi di estrazione informazioni

Come primo passo dell'implementazione, è necessario estrarre le informazioni dai vari formati di rappresentazione degli articoli. È questa la fase dove vengono applicati la maggior parte degli algoritmi di text-mining. Ciò risulta ovvio, dato che lo scopo delle relative tecniche è quello di estrarre informazioni dal testo strutturato o non strutturato. Sono state qui applicate tecniche come lo stemming, la selezione delle parole più rilevanti e la rimozione delle stop word. Il formato BibTex e gli abstract degli articoli sono stati salvati come semplici file di testo txt, gli articoli nel loro formato full text sono stati invece recuperati in formato pdf. Per ogni tipologia di rappresentazione fra quelle scelte, è stata implementata una semplice classe in Java che estrae da essi le informazioni più rilevanti. Ognuna di queste classi, attraverso varie metodologie, legge le informazioni grezze (testo non strutturato, tag, valori), individua le parti che potrebbero essere di interesse e le organizza all'interno di apposite strutture dati. La struttura dati scelta è una mappa, le chiavi della stessa sono i titoli degli articoli, mentre i valori sono i relativi contenuti rielaborati che erano presenti nel testo.

BibTex

I file di testo contenenti il formato BibTex, presentando su ogni riga una coppia tag valore, sono stati interpretati riga per riga. Ogni riga viene trascritta in una stringa, che, successivamente, subisce un processo di elaborazione per l'estrazione dei dati rilevanti. La stringa, durante questo processo di raffinazione, viene quindi suddivisa in due stringhe una contenente il tag e l'altra il suo valore. Fatto ciò si controlla se il tag trovato è di interesse o meno, se fa parte dell'insieme dei tag ritenuti poco rilevanti si passa a leggere e analizzare la coppia di valori successiva. La scelta dei tag rilevanti o meno è stata da me fatta, ho deciso di rimuovere i tag, con i relativi valori, che contengono informazioni sulle quali difficilmente si potrebbe fare clustering. I tag che ho deciso di rimuovere sono: Pages, Isbn, Isbn-10, Chapter, Doi, Issn, Volume, Url, Url-pdf, Pdf-local, Issn-online, Isbn-online, eccetera. Dato che nel formato BibTex di alcuni articoli è presente anche l'abstract, questo viene rimosso volendolo analizzare solo successivamente. Nel caso il tag sia di interesse viene elaborata la stringa contenente i valori. Per prima cosa vengono

rimossi i simbolismi come parentesi, caratteri speciali e punteggiatura a parte la virgola. La virgola non è stata rimossa perché all'interno del formato BibTex viene utilizzata come separatore fra i vari valori, se ne esistono più di uno. Spesso alcuni tag hanno come valori frasi, ad esempio il titolo dell'elaborato o le note, per questo, viene applicato un filtro che elimina le stop word cioè le parole che non contengono concetti rilevanti. Sono ritenute ad esempio stop word gli articoli, le preposizioni e le congiunzioni.

Abstracts

Gli abstract dei vari articoli sono sotto forma di testo non strutturato, scritto in inglese. Per ottenere le informazioni più significative, che altro non sono che le parole chiave, è sufficiente leggerne il contenuto e attuare tecniche di Text Mining come lo stemming e filtrare le stop word. Il testo in questo caso può essere letto interamente in una singola volta dall'algoritmo e non necessariamente riga per riga. Dal testo ottenuto vengono rimossi tutti i simboli di punteggiatura, le parentesi, i caratteri speciali e numerici. Si ottiene così un testo composto solamente da parole. Si controlla quindi se tra le parole si trovano stop word, che vengono direttamente rimosse. Viene quindi applicato un l'algoritmo di stemming che ha come obiettivo quello di accorpare le parole con la stessa radice (stem) in uno stesso gruppo di parole. Si riduce così il numero di parole presenti nel testo. Grazie allo stemming si rimuovono:

- tutte le coniugazioni dei verbi che vengono portati alla forma infinita;
- i plurali, tutte le parole vengono portate al singolare;
- vezzeggiativi, diminutivi, eccetera.

Se un termine appare più di una volta, si tiene traccia del numero di volte in cui appare, che verrà associato alla parola e inserito nella struttura dati.

Plain text

Essendo l'articolo, nella sua interezza, in formato pdf, è stato in primis necessario trasformarlo da tale formato, non direttamente utilizzabile come file di input in un programma Java, a un semplice file di testo. Per questo passaggio è stata utilizzata una libreria Java pdfbox, la quale con un semplice comando, trascrive in formato txt un qualsiasi file salvato come pdf. L'argomento che viene passato al main della classe org.apache.pdfbox.ExtractText, che si occupa della traduzione, è il seguente:

-force <path assoluta file sorgente> <path assoluta file destinazione>

Una volta ottenuto il formato txt si procede alla lettura del contenuto dell'articolo come singola stringa di testo. Vengono poi rimossi punteggiatura, caratteri speciali e numeri.

Si ottiene così un vasto insieme di parole, che devono essere filtrate per ridurne il numero. Innanzi tutto si è scelto di ignorare le parole composte da tre o meno caratteri dato che generalmente non sono rilevanti, sono state poi eliminate dall'elenco tutte le stop words. È stato poi implementato e applicato l'algoritmo di stemming di Porter. Dopo un'attenta analisi si è deciso di non usare gli ultimi due passaggi dell'algoritmo, dato che il numero di parole rimosse era basso a fronte di uno stravolgimento della loro forma. Per ogni parola si è tenuto conto delle sue occorrenze sulla base delle quali la lista di parole finali è stata ordinata dalla più alla meno frequente. Essendo l'elenco delle parole così trovate molto grande, con dimensioni comprese fra le cento e le mille parole, a seconda della lunghezza dell'articolo in esame, si è deciso dare la possibilità di selezionare solo le n parole più frequenti. Dove n è un parametro che l'utente può impostare a suo piacimento prima di eseguire il programma.

5.3.2 Struttura dei centri di tuple e loro inizializzazione

Per l'esperimento in questione, si è deciso di partire da un unico centro di tuple. Durante l'evoluzione dell'esperimento, a run time verranno creati dinamicamente altri centri che rappresenteranno i vari cluster. L'obbiettivo finale è quello di avere un centro di tuple per ogni relativo cluster di articoli "simili". Nel centro di tuple iniziale vengono inserite tutte tuple che rappresentano ogni articolo scientifico ognuna delle quali ha la seguente struttura:

article(nome file, parola chiave 1, parola chiave 2, ..., parola chiave n).

Un esempio di tupla inserita è il seguente:

article(paper2, proceedings, 13th, workshop, object, agent, ..., september).

Questa semplice mansione è svolta da un singolo agente, che, alla fine del suo processo, farà partire l'agente o gli agenti che si occuperanno del vero e proprio clustering.

5.4 Esperimenti con BibTex, abstract o plain text

La prima scelta implementativa che è stata presa in questo esperimento è l'assegnazione dei pesi. In base al risultato che si vuole ottenere dalla organizzazione si possono infatti assegnare pesi diversi alle varie parole chiave. La somma dei pesi delle parole di ogni singolo articolo sarà pari a 1. In questo esperimento si è scelta un'assegnazione dei pesi assai semplice che non tiene conto della diversa rilevanza delle parole chiave. Ad ogni parola chiave dell'articolo, infatti, viene dato il medesimo peso pari a 1/n dove n è il numero di parole chiave dell'articolo. Questa scelta è stata presa in maniera autonoma e non tratta dalla letteratura, pareva infatti essere questo l'approccio migliore, non volendo avere negli esperimenti una funzione per l'assegnazione dei pesi troppo complessa. La

funzione di correlazione prende le tuple di due articoli e ne confronta una ad una le parole chiave, se una parola chiave esiste in entrambi gli articoli il suo peso viene moltiplicato per uno, nel caso contrario per zero. Il fattore di correlazione totale fra i due testi è pari alla somma dei singoli pesi così ottenuti. Il valore ottenuto sarà così sempre compreso tra zero e uno. Se è pari a zero fra i due articoli non esiste nessuna correlazione, quindi nessuna delle parole chiave di un articolo appare nell'altro. Se invece il risultato è pari a uno i due scritti sono identici, tutte le parole chiave di uno appaiono nell'altra. Il confronto fra le parole chiave, viene attuato tramite un semplice match sintattico, che è quello fornito dalla classe alice.logictuple.TupleArgument.

5.4.1 Algoritmo di clustering

L'algoritmo di clustering è interamente svolto da un singolo agente istanza della classe denominata OrganizationAgent. L'agente prende una tupla, inizialmente la prima tupla del centro di tuple appena inizializzato e ne calcola la congruenza rispetto alle altre contenute nel cluster di partenza. Se il valore della congruenza è superiore alla soglia di congruenza specificata non fa nulla e passa alla tupla successiva. Se invece è inferiore, controlla la congruenza della tupla rispetto alle tuple di ogni altro centro di tuple esistente. Se non ne esistono altri, oltre a quello di origine, o se non ne esiste uno che possa contenere la tupla, crea un nuovo centro di tuple. Terminato il controllo, la tupla viene rimossa dal suo centro di tuple di origine e viene inserito nel centro individuato al passo precedente. Il processo continua fino a che tutte le tuple del centro di tuple iniziale non sono state controllate.

Algorithm 2 Pseudo codice relativo all'algoritmo di clustering

```
1:  while Esistono nel cluster di partenza tuple di cui non si è calcolata la congruenza do
2:      prendi la prima tupla non ancora analizzata
3:      calcolane la congruenza rispetto alle tuple del cluster di partenza
4:      if la congruenza è inferiore alla soglia then
5:          for ogni cluster già esistente escluso quello di partenza do
6:              inizializza campo congruenza migliore a 0 e esiste cluster con false
7:              controlla la congruenza della tupla rispetto al cluster attuale
8:              if la congruenza calcolata è maggiore della soglia then
9:                  esiste cluster = true
10:                 if congruenza calcolata¿ congruenza migliore then
11:                     aggiorna il valore di congruenza migliore
12:                     aggiorna il cluster migliore
13:         rimuovi tupla da cluster di partenza
14:         if esiste un cluster == true then
15:             aggiungi tupla nel cluster migliore
16:         else
17:             crea nuovo cluster ( centro di tuple )
18:             aggiungi tupla al nuovo cluster
```

5.5 Esperimenti con algoritmi di clustering più complessi

Vengono qui presentati gli altri due algoritmi di clustering realizzati. Tutti gli esperimenti sono stati realizzati su un PC portatile con: processore Intel Core i7-720QM (1.6 GHz, 6MB L3 cache) e 4 GB di ram DDR3. Il sistema operativo utilizzato è Sabayon-Linux kernel 3.9.11 versione 64 bit. Il primo algoritmo si differenzia da quello descritto sopra per quel che riguarda la funzione di matching fra le tuple che rappresentano gli articoli, viene infatti utilizzato la formula della similarità del coseno. Il secondo ed ultimo è un'ulteriore evoluzione dell'algoritmo, oltre alla funzione di matching modificata, il numero di agenti che si occupano del clustering è molto più alto. Ad ogni tupla viene infatti associato un agente che si occuperà della sua sistemazione nel centro di tuple (cluster) più indicato.

5.5.1 Clustering con matching a similarità del coseno

Questo algoritmo, come sopra accennato, ha un sistema di matching diverso rispetto al mero matching sintattico. Il matching viene fatto utilizzando i concetti e la formula della similarità del coseno. IInizialmente ogni articolo viene rappresentato come un vettore di pesi ognuno associato ad una singola parola, si ottiene cosi uno spazio di vettori di numeri reali che verranno poi inseriti nella formula del calcolo della similarità del coseno. Nel processo per calcolare la similarità del coseno è necessario redigere in primis un vocabolario che contiene al suo interno tutte le parole presenti nei vari testi, tenendo traccia per ogni singola parola degli articoli che la contengono. Il peso di ogni singola parola per ogni singolo articolo viene calcolato secondo la seguente formula:

$$v_{i,j} = \frac{tf_{i,j}\log(\frac{colsize+0.5}{docf_i})}{tf_{i,j}+0.5+1.5\frac{doclen_j}{avgdoclen}\log(colsize+1)}$$

dove $v_{i,j}$ è il peso dell'i-esimo termine del vocabolario all'interno del documento j-esimo, $tf_{i,j}$ è il numero di volte che il termine appare nel documento, $docf_i$ il numero di documenti in cui è presente il termine, $doclen_j$ è il numero di termini del documento, $avgdoclen$ è il numero medio di termini per i documenti della collezione. Se ad esempio una parola non è contenuta all'interno del testo il relativo peso (valore) sarà pari a zero. Si ottengono così dei vettori di lunghezza uguale per ogni articolo in esame. Quando vogliamo calcolare la similarità fra due vettori che rappresentano ognuno un articolo è sufficiente applicare la seguente equazione:

$$Sim(C_i, C_j) = \frac{\sum_{k=1}^{n} c_i(t_k)\cdot c_j(t_k)}{\sqrt{\sum_{k=1}^{n} c_i(t_k)^2 \sum_{k=1}^{n} c_j(t_k)^2}}.$$

Dove C_i e C_j sono i vettori dei pesi assegnati ai vari termini per il documento i-esimo e j-esimo. Il risultato ottenuto ha valori compresi fra -1 ed 1, se il valore ottenuto è -1 vuol dire che i due articoli sono totalmente differenti, se è pari a 0 significa che i vettori

in esame sono scorrelati ed infine se pari ad 1, che sono identici in tutte le loro parti. La parte restante dell'algoritmo rimane invariata, tutto il lavoro di organizzazione dei cluster è svolto da un singolo agente.

5.5.2 Clustering con calcolo distribuito

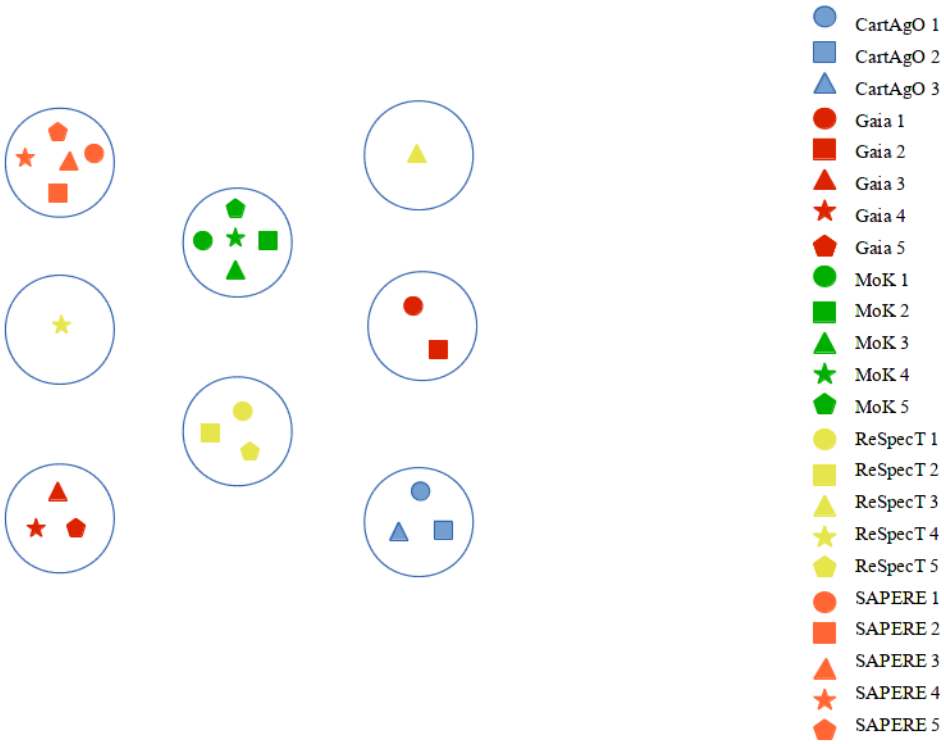
In questa implementazione, vengono ripresi i concetti alla base del clustering con matching a similarità del coseno. Il carico di lavoro viene però distribuito fra più agenti invece che su uno solo come negli algoritmi precedenti. L'algoritmo prevede che ad ogni tupla, inizialmente inserita nel centro di partenza, sia associato un agente istanza della Classe TupleOrganizationAgent. Si è reso necessario, per fare ciò, modificare l'agente che inizializza il sistema, in modo tale da associare durante l'inserimento nel centro delle tuple un nuovo agente ad ogni tupla. Ogni singolo agente svolgerà una sola volta, per una sola tupla, il calcolo e il controllo della relativa congruenza. Per fare ciò però non essendo l'unica entità esistente nel sistema, dovrà coordinarsi con gli altri agenti. Ho deciso quindi di sfruttare un centro di tuple, dedicato, per la coordinazione fra agenti. Prima di calcolare la congruenza con altri centri di tuple esistenti, l'agente preleva dal centro di tuple di coordinazione una tupla del tipo sem(token), grazie ad una operazione di tipo in(template tupla). Essendo presente un solo esemplare di questa tupla il comportamento dell'operazione è bloccante, se al momento dell'eseuzione essa non è presente nel centro di tuple. Tutti gli altri agenti che vogliono fare la medesima operazione devono attendere che l'agente reimmetta alla fine dei suoi calcoli la tupla nello spazio.

Capitolo 6

Risultati ottenuti

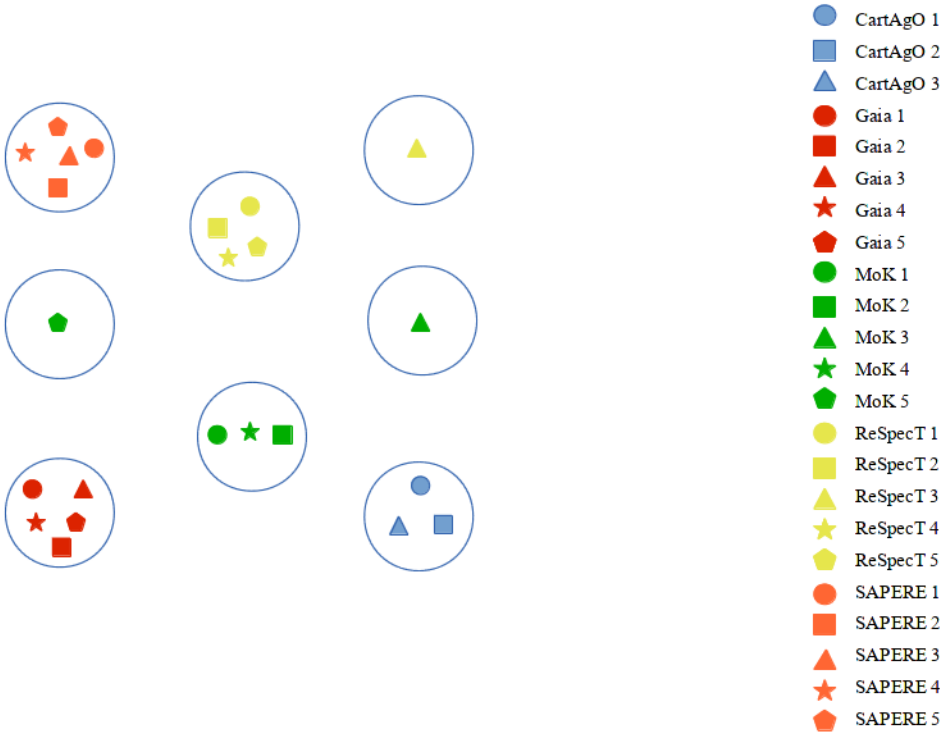
Qui di seguito vengono presentati i risultati in termini di cluster ottenuti di alcuni degli algoritmi sopra citati. Per analizzare i tempi di calcolo degli esperimenti si è utilizzato il tool Java VisualVM, che permette di analizzare i tempi di calcolo, anche delle singole parti del processo. Il primo risultato presentato è la composizione dei cluster ottenuta utilizzando l'algoritmo di clustering con matching a similarità del coseno con soglia pari a 0.10. Il tempo totale di calcolo impiegato è di due minuti, la maggior parte del tempo totale, viene impiegato per l'elaborazione delle richieste e delle risposte ai messaggi TuCSoN. Analizzando i tempi di calcolo più nel dettaglio, si nota come, il tempo impiegato per fare stemming è pari a 200 millisecondi, mentre sono necessari 1800 millisecondi per calcolare la similarità. I cluster ottenuti da questo primo esperimento, sono riportati all'interno dell'appendice A. Si può notare già a occhio nudo come articoli che, dal titolo, si presumono simili, si trovino nello stesso cluster. Un altro buon risultato è stato ottenuto anche sfruttando l'algoritmo più semplice di tutti: quello con semplice matching sintattico delle parole con soglia di similarità pari a 0.40. I cluster ottenuti da questo esperimento, sono riportati all'interno dell'appendice B. I risultati differiscono in maniera evidente l'uno dall'altro ma si nota come in generale gli articoli che presumibilmente dovrebbero appartenere allo stesso cluster effettivamente lo fanno. Il tempo totale impiegato per culsterizzare tutti gli articoli è pari a tre minuti, di cui, la maggior parte del tempo viene utilizzato per le operazioni TuCSoN. Le parti algoritmiche prese dal Text Mining, invece, hanno tempi di esecuzione più ragionevoli: 795 millisecondi per il calcolo delle similarità, il quale viene richiesto 180 volte e circa 300 millisecondi che servono, invece, per applicare l'algoritmo di stemming. La grande differenza nel valore della soglia di similarità non deve spaventare, dato che nel caso del matching con similarità del coseno si tiene conto di molte più parole. Infatti si tiene conto dell'intero insieme di parole contenute nei vari testi, e quindi i pesi sono di gran lunga più bassi. Quanto ottenuto sopra, si osserva che, nonostante ci sia una buona clusterizzazione degli articoli, i risultati hanno un'attendibilità limitata perchè non si è a conoscenza in realtà di quanto alcuni articoli siano in realtà correlati. Gli articoli non sono stati letti in modo

tale da analizzarne i contenuti effettivi, alcuni articoli che magari dal titolo si potrebbero presumere correlati in realtà non lo sono. Per questo si è deciso di attuare altri esperimenti, nei quali si è scelto di utilizzare articoli con tag ben distinti nel database di APICe. Il numero di articoli presi in esame è stato ridotto a ventitré e le classi di tag sono cinque: ReSpecT, Gaia, MoK, SAPERE, e CArtAgO. Per ogni tag sono stati presi cinque articoli, eccetto per CArtAgO, per il quale ne vengono presi in esame solo tre; ipotizzando che i tag siano stati assegnati correttamente, il risultato che ci si aspetta è quello di ottenere cinque cluster distinti. Il miglior risultato ottenuto con l'algoritmo di clustering con matching a similarità del coseno, è quello con soglia pari a 0.145.



Osservando questo risultato, si nota che i cluster non sono cinque come ci si aspettava ma otto e che vengono correttamente posizionati nello stesso cluster, tutti gli articoli con tag: SAPERE, CArtAgO e MoK. Gli articoli appartenenti alle altre due classi di tag vengono invece divisi in più cluster, si può però notare come non ci sia sovrapposizione fra gli articoli appartenenti a tag diversi. Diminuendo il valore della soglia, come ci si poteva aspettare, il numero di cluster si riduce ma ciò è dovuto al fatto che articoli appartenenti a classi di tag diverse si sovrappongono, all'aumentare invece il numero di cluster che vengono generati aumenta. Il tempo totale impiegato per l'esecuzione è pari a circa 20 secondi. Studiando i campionamenti dei vari tempi cpu si può osservare però che la maggior parte del tempo di calcolo viene impiegato per inviare le richieste e ricevere i

risultati delle operazioni; 200 millisecondi vengono invece impiegati per inizializzare tutti gli space vector, mentre per il calcolo della similarità sono necessari circa 40 millisecondi. Un altro buon risultato lo si ottiene anche utilizzando l'algoritmo di clustering da me ideato con un valore di soglia di 0.35 analizzando le cento parole chiave più frequenti.



In questo esperimento vengono correttamente clusterizzati gli articoli delle classi SAPERE, CArtAgO e Gaia. Per questo esperimento il tempo totale di calcolo necessario è pari a circa 20 secondi. Analizzano i tempi cpu campionati da Java VisualVM si nota come la maggior parte del tempo di calcolo è stato impiegato per l'invio e le risposte ai messaggi TuCSoN. Il tempo impiegato per calcolare i valori di similarità è pari a 57 millisecondi, mentre sono 50 i millisecondi impiegati per applicare lo stemming. Come in precedenza anche qui l'articolo *Prototyping A&A ReSpecT in Maude* risulta essere isolato in un cluster. Il problema maggiore che si riscontra da questo risultato è che non vengono ben clusterizzati gli articoli con tag MoK, infatti invece di avere un unico cluster vengono suddivisi in tre cluster.

Capitolo 7

Conclusioni

Dopo aver analizzato attentamente i risultati ottenuti dai vari test, si può affermare che alcune delle tecniche studiate sono utilizzabili all'interno di sistemi per l'auto-organizzazione della conoscenza come MoK. Sicuramente si potranno sfruttare algoritmi di stemming, come quello implementato o altri simili, per ridurre la dimensionalità delle possibili parole chiave di un testo non strutturato. I tempi necessari per questa elaborazione sono infatti del tutto ragionevoli: in media 3 millisecondi per fare lo stemming di un articolo. Inoltre, anche le due implementazioni per il calcolo della similarità potrebbero essere usate all'interno di MoK come funzioni di matching. C'è però da sottolineare che ambedue non tengono conto della semantica del testo e delle parole, per questo non apporterebbero al sistema i miglioramenti sperati. Gli algoritmi di clustering implementati in toto non sono strumenti di supporto utilizzabili, dati gli alti tempi computazionali. Ciò non esclude che altri algoritmi di clustering possano essere utilizzati all'interno di applicazioni MoK. Come si può notare infatti dai risultati dei test, la maggior parte del tempo di calcolo viene sfruttata per richieste e risposte ai messaggi TuCSoN. Questo potrebbe dipendere dal fatto che le implementazioni degli algoritmi usano a dismisura lo scambio di tuple TuCSoN. Le tecniche di Text Mining testate, ma anche quelle solamente studiate, potrebbero essere dei validi strumenti di supporto a sistemi di auto-organizzazione della conoscenza come MoK. Se si vuole però analizzare anche l'aspetto semantico dei testi, tra le tecniche studiate, quelle che potrebbero essere di aiuto sono poche: l'unico esempio citato e presente nella Sezione 1.5.1 è il clustering basato su concetti e non su parole chiave.

Appendice A:

Risultati esperimento con matching a similarità del coseno e soglia di similarità pari a 0.10.
Ogni gruppo o singolo articolo separato da linea vuota rappresenta un cluster del sistema.

```
'Molecules of Knowledge: Self-Organisation in Knowledge-Intensive Environments'
'Molecules of Knowledge: A Novel Perspective over Knowledge Management'
'Self-Organising News Management: The Molecules of Knowledge Approach'

'Probabilistic Modular Embedding for Stochastic Coordinated Systems'
'Towards the Analysis & Prediction of Complex System Behaviour in SAPERE'
'Probabilistic Embedding: Experiments with Tuple-based Probabilistic Languages'

'Simulation of caspases apoptotic signalling pathway in a tuple space-based bioinformatics infrastructure'

'Cross-Network Opportunistic Collection of Urgent Data in Wireless Sensor Networks'
'Self-organising Semantic Resource Discovery for Pervasive Systems'
'A Simulation Framework for Pervasive Services Ecosystems'
'Gradient-based Self-organisation Patterns of Anticipative Adaptation'
'Composing gradients for a context-aware navigation of users in a smart-city'
'Combining self-organisation, context-awareness and semantic reasoning: the case of resource discovery in opportunistic networks'

'Self-adaptive software needs quantitative verification at runtime'
'Adaptive organizational changes in agent-oriented methodologies'
'Adaptable Multi-Agent Systems: The Case of the Gaia Methodology'
'Engineering Pervasive Multiagent Systems in SAPERE'
'Stability Assessment of Aspect-Oriented Software Architectures: A Quantitative Study'
'Agent-based Conference Management: A Case Study in SODA'
'Processes Engineering & AOSE'
'Software Engineering for Self-Organizing Systems'
'Agents, Multi-Agent Systems and Declarative Programming: Who, What, When, Where, Why, How?'
'Building an Agent Methodology from Fragments: the MEnSA experience'

'Interdependent Artificial Institutions in Agent Environments'

'Description and Composition of Bio-Inspired Design Patterns: the Gradient Case'
'Towards Situated Awareness in Urban Networks: A Bio-inspired Approach'
'Description and composition of bio-inspired design patterns: a complete overview'
'Description and Composition of Bio-Inspired Design Patterns: the Gossip Case'
'BIO-CORE: Bio-inspired Self-organising Mechanisms Core'

'Operational Semantics of Proto'
'Engineering Confluent Computational Fields: from Functions to Rewrite Rules'
'Chemical-Inspired Self-Composition of Competing Services'
'Linda in space-time: an adaptive coordination model for mobile ad-hoc environments'
'A Framework for Modelling and Simulating Networks of Cells'
'Core Operational Semantics of Proto'
'An Agent-based Model for the Pattern Formation in Drosophila Melanogaster'

'Implicit: A Multi-agent Recommendation System for Web Search'
'Situation Identification Techniques in Pervasive Computing: A Review'
'Deep diving into BitTorrent locality'

'Programming abstractions for integrating autonomous and reactive behaviors: an agent-oriented approach'
'Designing a general-purpose programming language based on agent-oriented abstractions: the simpAL project'
'From Actors and Concurrent Objects to Agent-Oriented Programming in simpal'
'Environment Programming in Multi-Agent Systems -- An Artifact-Based Perspective'
'Multi-agent Oriented Programming with JaCaMo'
'JaCa-Android: an agent-based platform for building smart mobile applications'
'Formalising the Environment in MAS Programming: A Formal Model for Artifact-Based Environments'
'Typing Multi-Agent Programs in simpAL'
'simpA: An Agent-oriented Approach for Programming Concurrent Applications on top of Java'
'Exploiting Agent-Oriented Programming for Building Advanced Web 2.0 Applications'

'Coordination Models and Languages: From Parallel Computing To Self-Organisation'
'Nature-inspired Coordination for Complex Distributed Systems'
'Dynamic Composition of Coordination Abstractions for Pervasive Systems: The Case of LogOp'
'Nature-inspired Coordination Models: Current Status, Future Trends'
'Agents Writing on Walls: Cognitive Stigmergy and Beyond'

'Using SOA Governance Design Methodologies to Augment Enterprise Service Descriptions'

'Sustainable Biomass Power Plant Location in the Italian Emilia-Romagna region'
'Debt Deleveraging and Business Cycles. An Agent-Based Perspective'

'Reaction Factoring and Bipartite Update Graphs
```

Accelerate the Gillespie Algorithm for Large-Scale
Biochemical Systems'

'Promoting Space-Aware Coordination: ReSpecT as a Spatial-Computing Virtual Machine'

'Collaborative Learning and ICT: A Prototypal Learning Environment'

'Infrastructures and Tools for Multiagent Systems for the New Generation of Distributed Systems'

'Towards Temporal Verification of Emergent Behaviours in Swarm Robotic Systems'
'Verifying the Evolution of Probability Distributions Governed by a DTMC'

'Unsupervised Learning of True Ranking Estimators using the Belief Function Framework'

'Simulation and Analysis of Distributed Systems in Klaim'
'Toward Approximate Stochastic Model Checking of Computational Fields for Pervasive Computing Systems'

'Towards a Coordination Approach to Adaptive Pervasive Service Ecosystems'
'Self-aware Pervasive Service Ecosystems'

'Towards a Logic Framework for Web Programming'

'Risk Analysis and Deployment Security Issues in a Multi-Agent System'

'Exploiting the JaCaMo framework for realising an adaptive room governance application'

'From Space to Stage: How Interactive Screens Will Change Urban Life'

'Description Spaces with Fuzziness'
'Coordination in Open and Dynamic Environments with TuCSoN Semantic Tuple Centres'
'Semantic Tuple Centres'
'Coordinating e-Health Systems with TuCSoN Semantic Tuple Centres'

'BaSi: Multi-Agent Based Simulation for Medieval Battles'
'HomeManager: Testing Agent-Oriented Software Engineering in Home Intelligence'
'Simulation in Agent-Oriented Software Engineering: The SODA'

'Spatial Coordination of Pervasive Services through Chemical-inspired Tuple Spaces'
'A Coordination Approach to Adaptive Pervasive Service Ecosystems'
'A Computational Framework for Multilevel Morphologies'
'On the Space-time Situation of Pervasive Service Ecosystems'
'Towards a comprehensive approach to spontaneous self-composition in pervasive ecosystems'
'Injecting Self-organisation into Pervasive Service Ecosystems'
'Self-organising Pervasive Ecosystems: A Crowd Evacuation Example'
'On Competitive Self-composition in Pervasive Services'
'A Survey on Nature-inspired Metaphors for Pervasive Service Ecosystems'
'A Model for Drosophila Melanogaster Development from a Single Cell to Stripe Pattern Formation'
'Pervasive Ecosystems: a Coordination Model based on Semantic Chemistry'
'From SOA to Pervasive Service Ecosystems: An Approach based on Semantic Web technologies'
'A Coordination Approach to Spatially-Situated Pervasive Service Ecosystems'
'A Chemical Inspired Simulation Framework for Pervasive Services Ecosystems'

Appendice B:

Risultati dell'esperimento con matching sintattico e soglia di similarità pari a 0.40.
Ogni gruppo o singolo articolo separato da linea vuota rappresenta un cluster del sistema.

```
'Molecules of Knowledge: Self-Organisation in Knowledge-Intensive Environments'
'Molecules of Knowledge: A Novel Perspective over Knowledge Management'
'Self-Organising News Management: The Molecules of Knowledge Approach'

'Probabilistic Modular Embedding for Stochastic Coordinated Systems'
'Probabilistic Embedding: Experiments with Tuple-based Probabilistic Languages'

'Simulation of caspases apoptotic signalling pathway in a tuple space-based bioinformatics infrastructure'

'Towards the Analysis & Prediction of Complex System Behaviour in SAPERE'

'Cross-Network Opportunistic Collection of Urgent Data in Wireless Sensor Networks'

'Self-adaptive software needs quantitative verification at runtime'

'Interdependent Artificial Institutions in Agent Environments'

'Self-organising Semantic Resource Discovery for Pervasive Systems'
'On the Space-time Situation of Pervasive Service Ecosystems'
'Composing gradients for a context-aware navigation of users in a smart-city'
'Towards Situated Awareness in Urban Networks: A Bio-inspired Approach'
'Towards a comprehensive approach to spontaneous self-composition in pervasive ecosystems'
'Combining self-organisation, context-awareness and semantic reasoning: the case of resource discovery in opportunistic networks'
'Injecting Self-organisation into Pervasive Service Ecosystems'
'Pervasive Ecosystems: a Coordination Model based on Semantic Chemistry'

'Description and Composition of Bio-Inspired Design Patterns: the Gradient Case'
'Description and composition of bio-inspired design patterns: a complete overview'
'Description and Composition of Bio-Inspired Design Patterns: the Gossip Case'
'BIO-CORE: Bio-inspired Self-organising Mechanisms Core'

'Operational Semantics of Proto'
'Core Operational Semantics of Proto'

'Implicit: A Multi-agent Recommendation System for Web Search'

'Adaptive organizational changes in agent-oriented methodologies'
'Adaptable Multi-Agent Systems: The Case of the Gaia Methodology'
'Agent-based Conference Management: A Case Study in SODA'
'BaSi: Multi-Agent Based Simulation for Medieval Battles'
'HomeManager: Testing Agent-Oriented Software Engineering in Home Intelligence'

'Programming abstractions for integrating autonomous and reactive behaviors: an agent-oriented approach'
'Designing a general-purpose programming language based on agent-oriented abstractions: the simpAL project'
'From Actors and Concurrent Objects to Agent-Oriented Programming in simpal'
'Environment Programming in Multi-Agent Systems -- An Artifact-Based Perspective'
'Multi-agent Oriented Programming with JaCaMo'
'JaCa-Android: an agent-based platform for building smart mobile applications'
'Typing Multi-Agent Programs in simpAL'
'simpA: An Agent-oriented Approach for Programming Concurrent Applications on top of Java'
'Exploiting Agent-Oriented Programming for Building Advanced Web 2.0 Applications'

'Spatial Coordination of Pervasive Services through Chemical-inspired Tuple Spaces'
'A Coordination Approach to Adaptive Pervasive Service Ecosystems'
'A Computational Framework for Multilevel Morphologies'
'Towards a Coordination Approach to Adaptive Pervasive Service Ecosystems'

'Coordination Models and Languages: From Parallel Computing To Self-Organisation'
'Nature-inspired Coordination for Complex Distributed Systems'
'Nature-inspired Coordination Models: Current Status, Future Trends'
'Agents Writing on Walls: Cognitive Stigmergy and Beyond'

'Using SOA Governance Design Methodologies to Augment Enterprise Service Descriptions'

'A Simulation Framework for Pervasive Services Ecosystems'
'Self-organising Pervasive Ecosystems: A Crowd Evacuation Example'

'Sustainable Biomass Power Plant Location in the Italian Emilia-Romagna region'

'Reaction Factoring and Bipartite Update Graphs
Accelerate the Gillespie Algorithm for Large-Scale
```

Biochemical Systems'

'Gradient-based Self-organisation Patterns of Anticipative Adaptation'

'Promoting Space-Aware Coordination: ReSpecT as a Spatial-Computing Virtual Machine'

'Collaborative Learning and ICT: A Prototypal Learning Environment'

'Engineering Confluent Computational Fields: from Functions to Rewrite Rules'
'Linda in space-time: an adaptive coordination model for mobile ad-hoc environments'

'Chemical-Inspired Self-Composition of Competing Services'
'On Competitive Self-composition in Pervasive Services'

'Infrastructures and Tools for Multiagent Systems for the New Generation of Distributed Systems'

'Engineering Pervasive Multiagent Systems in SAPERE'

'A Framework for Modelling and Simulating Networks of Cells'
'A Model for Drosophila Melanogaster Development from a Single Cell to Stripe Pattern Formation'

'Dynamic Composition of Coordination Abstractions for Pervasive Systems: The Case of LogOp'

'Towards Temporal Verification of Emergent Behaviours in Swarm Robotic Systems'

'Unsupervised Learning of True Ranking Estimators using the Belief Function Framework'

'Formalising the Environment in MAS Programming: A Formal Model for Artifact-Based Environments'

'Situation Identification Techniques in Pervasive Computing: A Review'

'Stability Assessment of Aspect-Oriented Software Architectures: A Quantitative Study'

'Simulation and Analysis of Distributed Systems in Klaim'

'Towards a Logic Framework for Web Programming'

'Toward Approximate Stochastic Model Checking of Computational Fields for Pervasive Computing Systems'

'Risk Analysis and Deployment Security Issues in a Multi-Agent System'

'Exploiting the JaCaMo framework for realising an adaptive room governance application'

'Processes Engineering & AOSE'
'Building an Agent Methodology from Fragments: the MEnSA experience'
'Simulation in Agent-Oriented Software Engineering: The SODA'

'From Space to Stage: How Interactive Screens Will Change Urban Life'

'A Survey on Nature-inspired Metaphors for Pervasive Service Ecosystems'
'From SOA to Pervasive Service Ecosystems: An Approach based on Semantic Web technologies'
'A Coordination Approach to Spatially-Situated Pervasive Service Ecosystems'

'Software Engineering for Self-Organizing Systems'

'Deep diving into BitTorrent locality'

'Description Spaces with Fuzziness'
'Coordination in Open and Dynamic Environments with TuCSoN Semantic Tuple Centres'
'Semantic Tuple Centres'
'Coordinating e-Health Systems with TuCSoN Semantic Tuple Centres'

'An Agent-based Model for the Pattern Formation in Drosophila Melanogaster'

'Agents, Multi-Agent Systems and Declarative Programming: Who, What, When, Where, Why, How?'

'Verifying the Evolution of Probability Distributions Governed by a DTMC'

'Debt Deleveraging and Business Cycles. An Agent-Based Perspective'

'Self-aware Pervasive Service Ecosystems'

'A Chemical Inspired Simulation Framework for Pervasive Services Ecosystems'

Bibliografia

[1] Subhash K. Shinde Aruna Jadhav. A concept base mining model for npl using text clustering. *International Journal of Engineering Research e Technology (IJERT)*, 2013.

[2] Dario Benedetto, Emanuele Caglioti, and Vittorio Loreto. Language Trees and Zipping, December 2001.

[3] Enrico Denti, Andrea Omicini, and Vladimiro Toschi. Coordination technology for the development of multi-agent systems on the Web. In Evelina Lamma and Paola Mello, editors, *6th AI*IA Congress of the Italian Association for Artificial Intelligence (AI*IA'99)*, pages 29–38, Bologna, Italy, 14–17 September 1999. Pitagora Editrice Bologna.

[4] Eibe Frank, Gordon W. Paynter, Ian H. Witten, Carl Gutwin, and Craig G. Nevill-Manning. Domain-specific keyphrase extraction. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, IJCAI '99, pages 668–673, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[5] Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, December 1977.

[6] Jonathan L. Herlocker, Joseph A. Konstan, Al Borchers, and John Riedl. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '99, pages 230–237, New York, NY, USA, 1999. ACM.

[7] Andreas Hotho, Andreas Nürnberger, and Gerhard Paaß. A brief survey of text mining. *LDV Forum - GLDV Journal for Computational Linguistics and Language Technology*, 2005.

[8] Maurizio Lana. Come scriveva gramsci? metodi matematici per riconoscere scritti gramsciani anonimi. 2010.

[9] Anton Leuski. Evaluating document clustering for interactive information retrieval. In *Proceedings of the tenth international conference on Information and knowledge management*, CIKM '01, pages 33–40, New York, NY, USA, 2001. ACM.

[10] Karen E. Lochbaum and Lynn A. Streeter. Comparing and combining the effectiveness of latent semantic indexing and the ordinary vector space model for information retrieval. *Inf. Process. Manage.*, 25(6):665–676, November 1989.

[11] Stefano Mariani and Andrea Omicini. Molecules of Knowledge: A novel perspective over knowledge management. In Paolo Liberatore, Michele Lombardi, and Floriano Scioscia, editors, *Proceedings of the Doctoral Consortium of the 12th Symposium of the Italian Association for Artificial Intelligence*, volume 926 of *CEUR Workshop Proceedings*, pages 23–27, Rome, Italy, 15 June 2012. AI*IA, Sun SITE Central Europe, RWTH Aachen University.

[12] Stefano Mariani and Andrea Omicini. MoK: Stigmergy meets chemistry to exploit social actions for coordination purposes. In Harko Verhagen, Pablo Noriega, Tina Balke, and Marina de Vos, editors, *Social Coordination: Principles, Artefacts and Theories (SOCIAL.PATH)*, pages 50–57, AISB Convention 2013, University of Exeter, UK, 3–5 April 2013. The Society for the Study of Artificial Intelligence and the Simulation of Behaviour.

[13] Stefano Mariani and Andrea Omicini. Self-organising news management: The *Molecules of Knowledge* approach. In Jeremy Pitt, editor, *Self-Adaptive and Self-Organizing Systems Workshops (SASOW)*, pages 235–240. IEEE CS, April 2013. 2012 IEEE Sixth International Conference (SASOW 2012), Lyon, France, 10-14 September 2012. Proceedings.

[14] Andrea Omicini and Enrico Denti. From tuple spaces to tuple centres. *Science of Computer Programming*, 41(3):277–294, November 2001.

[15] M. F. Porter. Readings in information retrieval. chapter An algorithm for suffix stripping, pages 313–316. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.