POLITECNICO DI MILANO

Facoltà di Ingegneria

Dipartimento di Elettronica e Informazione

Corso di Laurea Specialistica in Ingegneria Informatica

CLASSIFICAZIONE SINTATTICA CON METODI INSIEMISTICI

Relatore: Prof.ssa Chiara Francalanci

Co-relatori: Ing. Francesco Merlo, Ing. Alessandro Poli

Tesi di laurea di:

Claudio Carcaci

Matricola 725004

Anno Accademico 2012 - 2013

Sommario

Il presente lavoro di tesi e l'applicativo realizzato ha come obiettivo quello di fornire un'analisi guidata totalmente dalla sintassi che permetta di classificare messaggi, post, tweet provenienti dai social network.

Riuscire a classificare testi basandosi sulla sintassi è un problema che è stato affrontato fino agli inizi degli anni 2000 in contemporanea con la nascita dei primi motori di ricerca e quindi nell'ottica della soluzione del problema relativo all'Information Retrieval (IR) e in seguito alla Text Categorization (TC).

Con questo lavoro di tesi si vuole fornire un'elencazione dei metodi sintattici presenti in letteratura testando i più completi ed efficienti a testi di breve lunghezza.

Identificate in seguito nuove caratteristiche utili all'ambito di testi brevi si vuole proporre un nuovo metodo di classificazione puramente sintattico, che ci si augura sia maggiormente performante sia dal punto di vista computazionale che dal punto di vista qualitativo rispetto alle procedure già esistenti e a procedure semantiche che classificano i testi valutandone il significato con ovvia ricaduta in termini computazionali.

Ringraziamenti

Un sentito ringraziamento alla Prof.sa Chiara Francalanci, all'Ing. Francesco Merlo e Alessandro Poli e ad Andrea Palla per avermi seguito in questo lavoro fornendo utili spunti e supporto.

Non meno importante è la gratitudine al Politecnico Di Milano, istituzione impegnata in prima linea e con risultati notevoli nella ricerca in Italia, nonostante le mille difficoltà relative alla fase storica che stiamo attraversando.

Infine una menzione particolare ai miei genitori, che mi hanno sempre sostenuto durante questi anni di studio, sotto tutti i punti di vista. Ma soprattutto sono riusciti a farmi capire con pazienza e attenzione quanto sia importante a livello personale e sociale lo studio e le attività culturali in genere.

Indice

Indice delle figure

Indice delle tabelle

Capitolo 1 Introduzione

Il presente lavoro di tesi propone una metodologia per la classificazione di testi brevi provenienti dai social network. Tale classificazione si focalizza sull'utilizzo di tecniche puramente sintattiche che garantiscono prestazioni migliori rispetto a tecniche di tipo semantico.

I capitoli che seguono tratteranno nel Capitolo 2 le basi teoriche e l'inquadramento contestuale di questo lavoro ma soprattutto una rassegna esaustiva delle tecniche già presenti in letteratura. Nel Capitolo 3 invece vi sarà l'esposizione dell'intero modello di analisi con i necessari riferimenti teorici. Infine nel Capitolo 4 sono esposti i risultati con i relativi commenti sui metodi confrontati che possono introdurre eventuali futuri lavori.

1.1 Nomenclatura

Si è scelta la dicitura “classe” per quanto riguardo la categorizzazione rispetto alla più naturale “categoria” data l’etichettatura dei testi in base a caratteristiche comuni. Per avere delle definizioni su cui basarsi si è consultato (1) e (2).

Inoltre nel presente lavoro viene usato genericamente il termine “testo” per fare riferimento indistintamente a messaggi, post, tweet e ogni altro genere di comunicazione presente nei social network di seguito riferito come “sorgente”. Ad eccezione del paragrafo 2.1 dove la dicitura “documento” è stata scelta per motivi storici.

Capitolo 2 **Stato dell'Arte**

Il problema posto è quello che consiste nel classificare con precisione i testi raccolti tramite crawling sui social network. I testi, come è facile immaginare, si riferiscono alle categorie più varie. Inoltre, vista la crescita esponenziale dei social network, essi sono numerosi e pertanto si rende necessario automatizzare il lavoro di classificazione.

Tale lavoro di classificazione, fino ad ora realizzato col supporto di dizionari e ontologie, è prettamente semantico, legato quindi al significato che il messaggio reca. Con questo lavoro di tesi invece si vuole provare a percorrere la strada della sintassi, analizzare in modo automatico la sintassi dei testi raccolti e provare a classificarli aiutandosi eventualmente con dati aggiuntivi, come ad esempio parole chiave e informazioni relative ad un dominio preselezionato di riferimento.

É facile quindi rilevare il background teorico di questo lavoro di tesi: Information Retrieval, Text Categorization. Questi argomenti riguardano l'evoluzione tecnologica che ha portato alla creazione dei moderni motori di ricerca e, come attività parallela, al web 2.0.

2.1 Information Retrieval

Con il crescere della quantità di contenuti testuali in forma cartacea ma soprattutto con l'avvento delle tecnologie digitali è sorto il problema di gestire tale mole di informazioni.

É diventato indispensabile poter catalogare e rendere possibile trovare informazioni tramite una "query" che esprime il bisogno informativo dell'utente. La catalogazione e l'implementazione di algoritmi di IR permettono quindi l'estrazione e la segnalazione all'utente delle fonti associabili alla query espressa, con un relativo grado di rilevanza.

Come espresso da Grossman e Frieder in (3) "*Information retrieval is devoted to finding relevant documents, not finding simple match to patterns*".

Col tempo hanno inoltre assunto sempre più importanza le metriche di valutazione della bontà dei risultati, su tutte in particolare la coppia formata da "precision" e "recall".

Si può quindi considerare "sistema di IR" un sistema che permette, con riferimento ad una fonte di dati, l'estrapolazione di informazioni relative ad un bisogno informativo di un utente espresso mediante query. Oltre alle metriche scelte assume quindi particolare importanza anche come è conservata la fonte dati e rappresentate le informazioni. Tali informazioni possono essere testi, insiemi numerici, statistiche, brevi messaggi o elementi multimediali.

2.1.1 Conservazione delle informazioni

A partire dai primi grossi sistemi di IR implementati tra i decenni 1950 e 1960 fino alla diffusione della rete internet a partire dalla metà degli anni 1990 la conservazione delle informazioni avveniva in grossi database o addirittura in archivi cartacei con codici di identificazione della posizione fisica.

In seguito alla nascita della rete internet e della crescita esponenziale delle informazioni è divenuto impensabile procedere nel salvataggio locale dell'intero insieme di dati, sia per la sua mole sia per la struttura per definizione distribuita della rete. Le informazioni quindi vengono conservate tramite un riassunto delle stesse che contiene necessariamente un riferimento all'originale completo. Si è quindi iniziato a

parlare di metadati ossia dati associati al sistema IR necessari a rappresentare ontologicamente l'insieme di dati.

2.1.2 Sistemi di Information Retrieval automatizzati

Se inizialmente la procedura di ricerca delle informazioni era svolta manualmente o attraverso sistemi primitivi in cui era necessario conoscere in maniera precisa almeno il titolo o il nome dell'autore dei testi conservati, in seguito sono stati introdotti sistemi di IR automatizzati.

Se consideriamo l'immensità della rete internet e pensiamo di sottoporre una query al sistema IR esso estrarrà un insieme di risultati che idealmente saranno il risultato di una equazione che sottrae dall'insieme di dati quello che viene chiamato "overload informativo" della query, in una formula molto concettuale:

$$Q(web) - Q(overload) = Q(risultati)$$

Dove la funzione $Q(\cdot)$ esprime la quantità di informazioni relative al suo argomento.

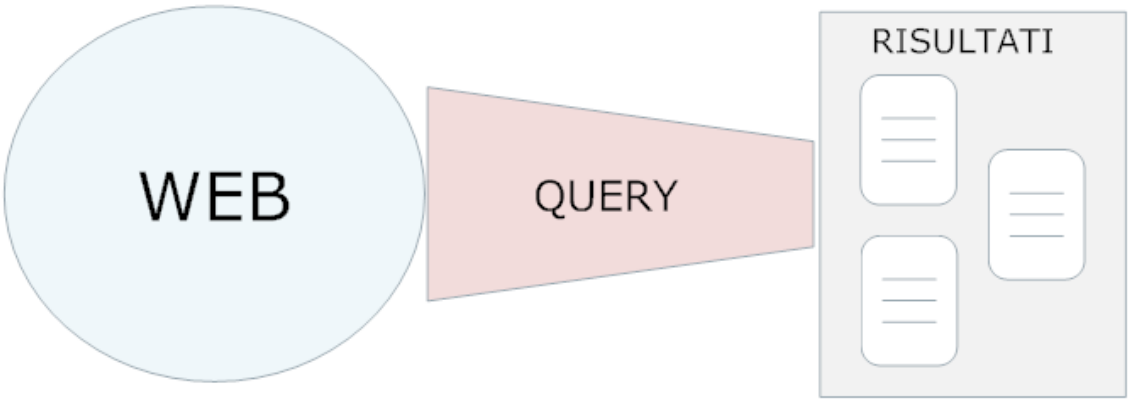

Figura 2-1 Sistema Information Retrieval Concettuale

Questo concetto di scrematura delle informazioni è insito nella definizione fornita in (4) molto significativa:"*Information Retrieval is finding material (usually documents) of an unstructured nature (usually text) that satisfy an information need from within large collections (usually stored on computers).*"

2.1.3 Elementi di Information Retrieval

Gli elementi da definire per caratterizzare l'attività di IR sono:

1. **Tipologia della collezione di dati:** come visto in precedenza assume particolare importanza la dimensione dell'insieme di dati. Con l'avvento della rete internet le collezioni sono diventate poco gestibili data la loro grandezza e di conseguenza non memorizzabili per intero in locale. Una collezione quindi può essere centralizzata o distribuita.

   Lateralmente rispetto a questa classificazione bisogna considerare il fatto che una collezione dati può crescere dinamicamente durante la sua esistenza, oppure può essere statica e rimanere uguale dalla sua creazione, come ad esempio collezioni storiche.
2. **Struttura dei documenti:** innanzi tutto bisogna considerare il fatto che i dati dal punto di vista semantico possono presentare ambiguità. Possono quindi avere una meta-struttura associata ad essi che permette di disambiguarne il contenuto.

   Come visto in precedenza i dati possono essere di due tipi: testuali o multimediali. Per quanto riguarda i dati multimediali poco è stato fatto fino ad ora e quello che viene preso in considerazione nell'ambito IR sono le informazioni testuali associati ad essi: titolo, didascalia, eventuali tag. I dati testuali invece possono essere categorizzati in base alla lingua, umana o non, con cui sono scritti e alla meta-struttura eventualmente associata.
3. **Query:** una query esprime un desiderio informativo da parte dell'utente. Spesso essa è espressa in modo molto impreciso al punto che può avere elementi inutili o addirittura dannosi rispetto al bisogno informativo originale.

   Una query può quindi avere due tipi di discrepanze ontologiche: "sensory gap" e "semantic gap". Sensory gap è identificato come la differenza tra l'oggetto nel mondo reale e l'informazione che lo descrive nell'insieme dati. Semantic gap invece rappresenta la lontananza tra la sua descrizione e la sua rappresentazione.

   Le query possono essere create con vari gradi di formalità: da un'espressione discorsiva informale fino ad un'espressione logica completamente

formalizzata. É chiaro che più la query è espressa in modo informale più aumenta la possibilità di avere errori nei risultati. Anche se di "errori" veri e propri è inopportuno parlare visto che la valutazione dei risultati è fatta mediante misurazioni relative e talvolta complementari come precision e recall. Bisogna pertanto tenere a mente che il grado di rilevanza cercato può variare facilmente da utente ad utente.

A questo problema risponde la costruzione di un "ranking" secondo la rilevanza dei risultati.

2.1.4 Metriche di valutazione

Le metriche usate per la valutazione di un sistema IR si basano principalmente sul concetto che non tutto ciò che viene ritrovato è utile all'utente e non tutto ciò che è utile viene effettivamente ritrovato. Questo porta alla definizione di due metriche complementari: precision e recall, con riferimento alla Figura 2-2:

1. **Precision**: frazione di documenti ritrovati che risultano rilevanti:

$$Pr = \frac{P(A \cap R)}{P(A)}$$

2. **Recall**: frazione di documenti rilevanti che sono stati ritrovati:

$$Re = \frac{P(A \cap R)}{P(R)}$$

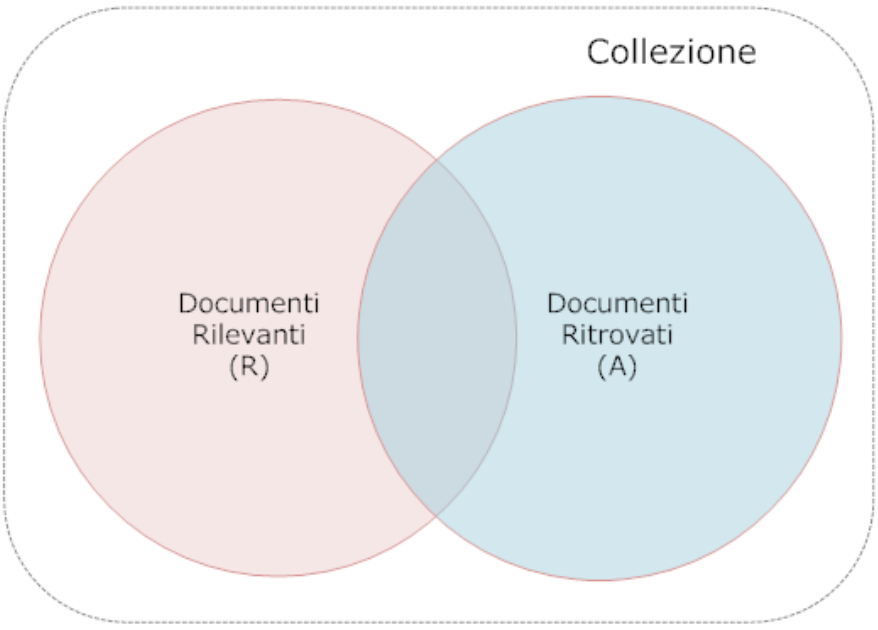

Figura 2-2 Diagramma di Venn Precision Recall

Vi è un trade-off tra precision e recall in quanto se l'insieme dei documenti ritrovati diventa grande più ingloberà quello dei documenti rilevanti migliorando la recall ma si andrà ad inglobare anche altri documenti inutili peggiorando la precision.

Tramite misurazioni empiriche la curva che meglio interpola questo effetto è quella della funzione:

$$Pr = 1 - Re^2, Re \in [0,1]$$

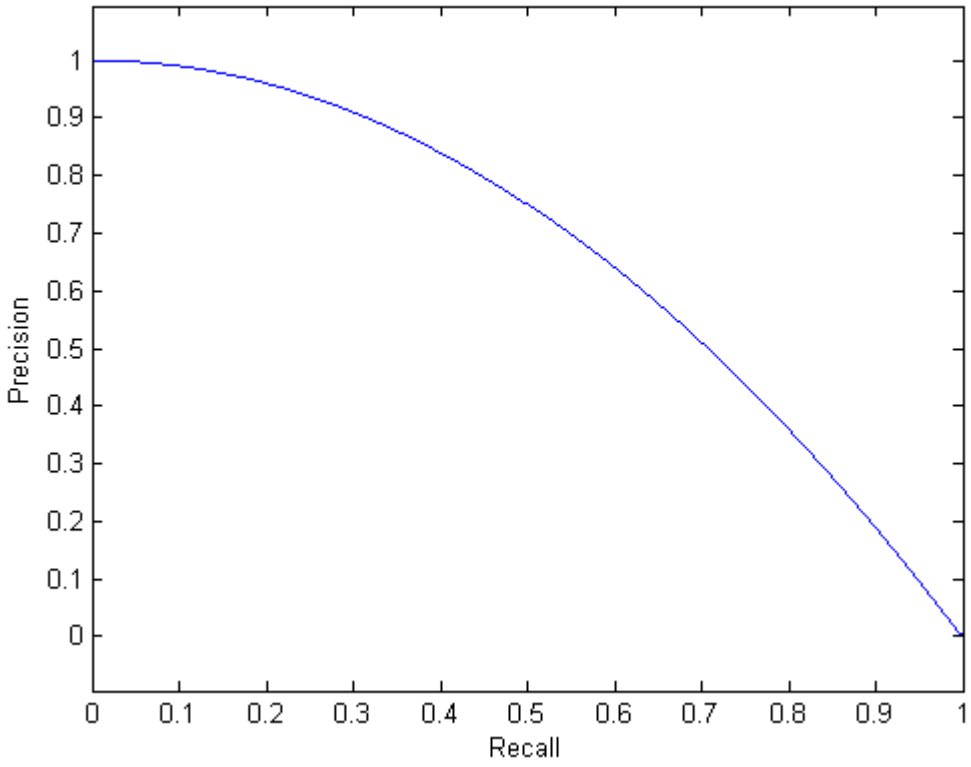

Figura 2-3 Precision-Recall Trade-off

Ovviamente la posizione all'interno della curva di trade-off tra precision e recall è valutata soggettivamente in maniera positiva o negativa in base alle esigenze dell'utente che usa il sistema.

Oltre a queste due metriche che valutano la bontà dei risultati ve ne sono altre, via via più raffinate e complesse per cui si rimanda ad una bibliografia più specifica a riguardo.

È ovviamente definibile anche un insieme di metriche prestazionali temporali e spaziali in quanto il processo di IR è pur sempre un algoritmo.

2.1.5 Schema generale di un sistema di Information Retrieval per i testi

Dopo aver visto cos'è l'IR e quali misure sono disponibili per selezionare i risultati di questo processo è possibile delineare lo schema generale di un sistema di IR.

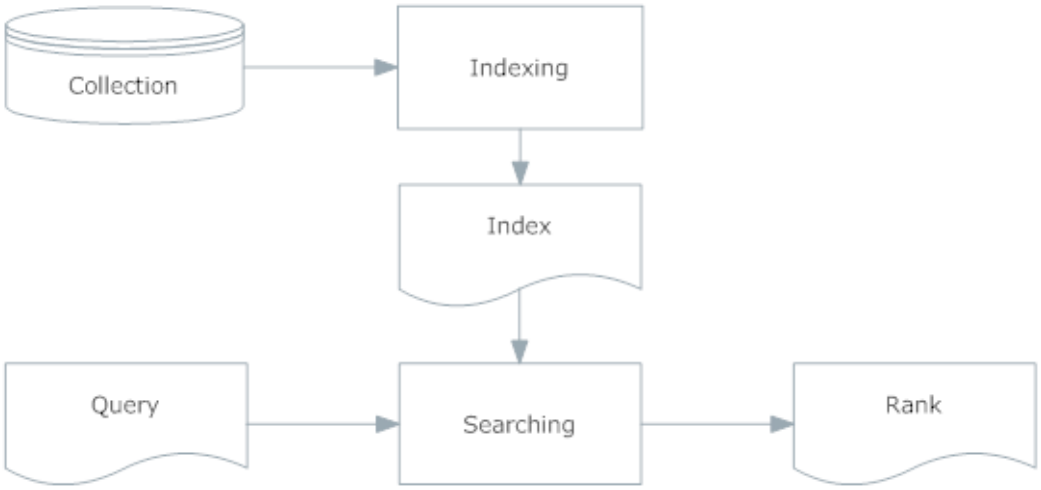

Figura 2-4 Sistema di Information Retrieval

Abbiamo quindi visto come sono memorizzati i dati della collezione, come e cosa esprimono le query e come viene valutata la qualità dei documenti ritrovati.

Quello che manca sono le componenti centrali del sistema quindi ciò che realizza le fasi di indicizzazione (indexing) e le strategie di IR per effettuare la ricerca (searching).

Vi è quindi un Indexer che, presi i documenti della collezione, crea l'indice: una struttura che contiene un sottoinsieme dei dati della collezione più eventuali altre sotto-strutture per facilitare la ricerca e l'accesso ai dati originali, in sostanza viene fornita una vista logica. La ricerca può risultare molto costosa su grosse collezioni di dati e quindi viene trovato un compromesso tra onerosità di calcolo e qualità dei risultati. Questo aspetto può migliorare esponenzialmente a patto di trovare rappresentazioni logiche maggiormente ottimizzate come ad esempio l'Inverted Index.

L'Inverted Index è la struttura dati più diffusa nei sistemi di IR e in prima approssimazione è la base di tutte le altre strutture dati più complesse che sono state via via sviluppate, esso è un elenco di termini con associati i riferimenti ai testi in cui compaiono.

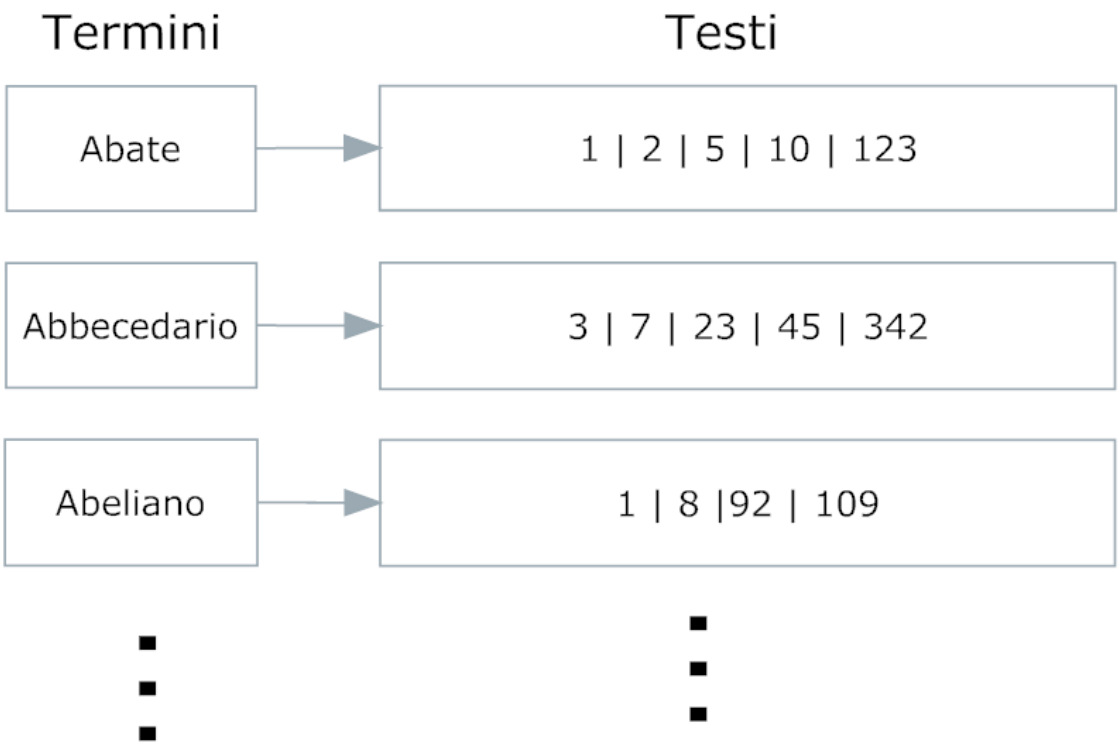

Figura 2-5 Esempio di Inverted Index

Un Inverted Index è quindi esprimibile come un insieme di coppie del tipo:

$$(t_i, d_j)$$

Dove d_j è un testo in cui compare il termine t_i.

E' quindi chiaro che data una query diventa pressoché banale estrarre i testi in cui compaiono il numero maggiore di termini contenuti nella query. Questa è una semplificazione forse un po' troppo forzata ma rappresenta un ottimo punto di partenza per un processo di IR. É chiaro che nella costruzione dell'Inverted Index e più in generale in ogni processo di analisi dei testi vengono adottate tecniche di ottimizzazione che qui non vedremo.

2.1.6 Strategie di Information Retrieval

Il processo di ricerca vero e proprio presa la query espressa dell'utente la traduce in una query a livello di sistema ossia formalizzata secondo regole ben definite. Quindi, data la vista logica dell'indice creato, viene combinata con la query a livello sistema.

Una strategia di IR assegna una misura di similarità tra la query espressa e i documenti trovati. Per adempiere a questo compito sono state ideate varie strategie esposte nella Figura 2-6 di seguito.

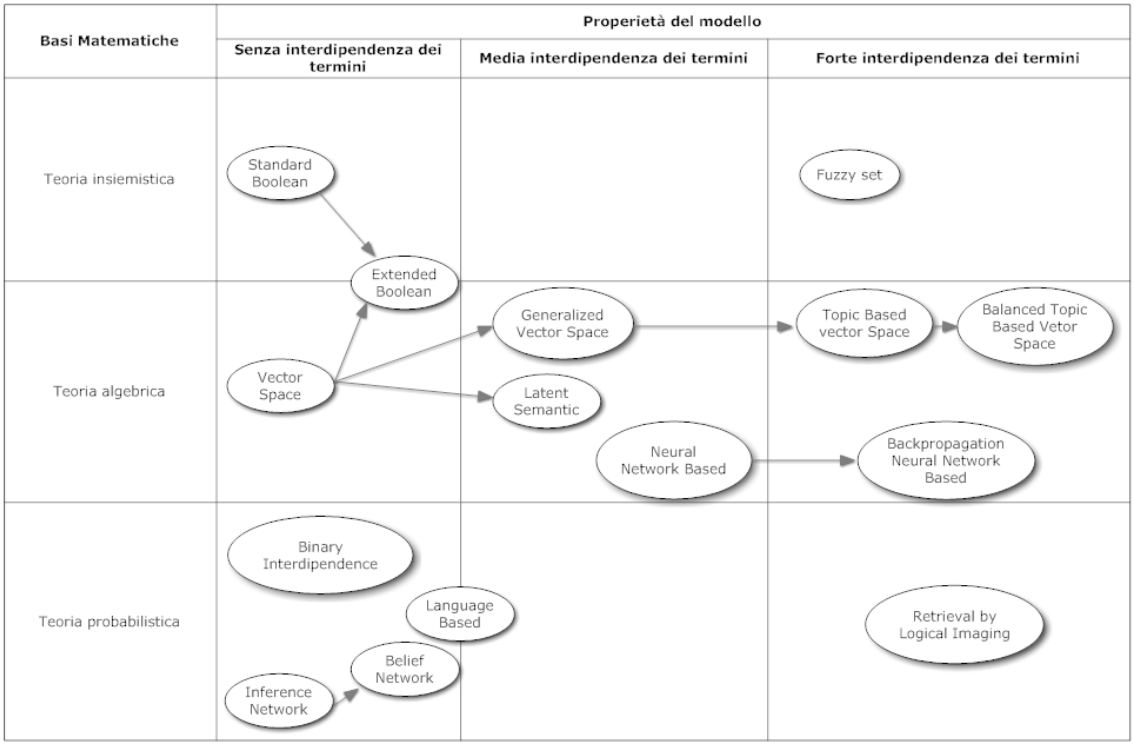

Figura 2-6 Tassonomia dei Modelli di Information Retrieval

Quello che differenzia i vari modelli sono due caratteristiche ad essi associate: la base matematica su cui basano l'elaborazione l'uso di informazioni semantiche legate all'interdipendenza dei termini.

Per quanto concerne gli obiettivi della tesi non è necessario valutare tecniche di IR di tipo semantico, concentrandoci quindi sui vari tipi di modelli matematici possiamo, con riferimento alla Figura 2-6, distinguere tre tipi di modelli:

1. **Modello booleano**: si basa sull'uso della teoria degli insiemi unita all'algebra booleana su logica proposizionale.
   Definiti:
   $w_{ij} \in \{0,1\}$ associato alla coppia: (t_i, d_j).
   S_{ti} insieme di documenti contenenti il termine t_i.
   $SC(q, d_j) \in \{0,1\}$ coefficiente di similarità tra la query q e il documento d_j.

Si trasforma la query q in una asserzione logica a che rappresenta la query a livello sistema: a sarà quindi una formula in logica proposizionale che esprime in forma normale disgiuntiva la query come presenza o non presenza di termini specifici. Ad esempio:

$$a = (t_a \wedge t_b) \vee (t_a \wedge (\sim t_c) \wedge t_d)$$

Ogni disgiunzione avrà un insieme di documenti che la soddisfano ossia che per i termini specificati nella disgiunzione il documento d_j ha associato il corretto valore di w_{ij}.

Un modo per estrarre i documenti che soddisfano una disgiunzione è usare gli insiemi S_{ti}. Se ad esempio prendiamo la prima disgiunzione $(t_a \wedge t_b)$ consideriamo gli insiemi:

$$S_{ta} = \{d_1, d_2\}$$
$$S_{tb} = \{d_2, d_3\}$$

Siccome t_a e t_b sono in AND tra loro farò l'intersezione tra i due insiemi ottenendo: $S_{ta} \cap S_{tb} = \{d_2\}$.

La stessa cosa potrò ripeterla per la disgiunzione $(t_a \wedge (\sim t_c) \wedge t_d)$ unendo poi i risultati.

Ovviamente vi sono le dovute ottimizzazioni sia nella trasformazione della query che nell'elaborazione dei documenti.

É chiaro che il modello booleano presenta grossi limiti, i più evidenti sono la difficoltà nel trasformare una query complessa in una formula proposizionale e la mancanza di una valutazione numerica che permette di creare un ranking. Il modello vettoriale presenta un approccio più raffinato e quindi più oneroso in termini di peso computazionale.

2. **Modello vettoriale**: l'idea alla base del modello vettoriale è che un documento e una query possono essere espressi come vettori dove le componenti rappresentano un valore di pesatura del termine all'interno del testo. Minore quindi è la distanza tra i vettori maggiore sarà la similarità tra i documenti, o tra un documento e una query.

   Quindi ogni termine rappresenta una dimensione nello spazio vettoriale che rappresenta il testo, si parla quindi di Vector Space Model. Assume quindi particolare importanza la pesatura usata per calcolare le componenti.

Pertanto una volta ottenuti i vettori associati ai documenti e "vettorizzata" la query è possibile stimare una correlazione tra ogni documento e il vettore che rappresenta la query:

$\underline{d_i} = (t_1, t_2, \dots t_n)$ documento i-esimo,

$\underline{q} = (t_1, t_2, \dots t_n)$ query,

$d_{di,q} = f\left(\underline{d_i}, \underline{q}\right)$ similarità tra il documento i-esimo e la query.

Una volta ottenuti i valori per ogni coppia documento-query è possibile, a differenza del modello booleano, fornire un rank.

Il calcolo della correlazione avviene attraverso l'uso di distanze consistenti matematicamente, eventualmente normalizzate. Il calcolo del rank relativo alla query avviene valutando i documenti via via "vicini" ad un documento che presenta una buona similarità fino a fermarsi appena si scende sotto un valore di similarità fissato come esposto nella Figura 2-7:

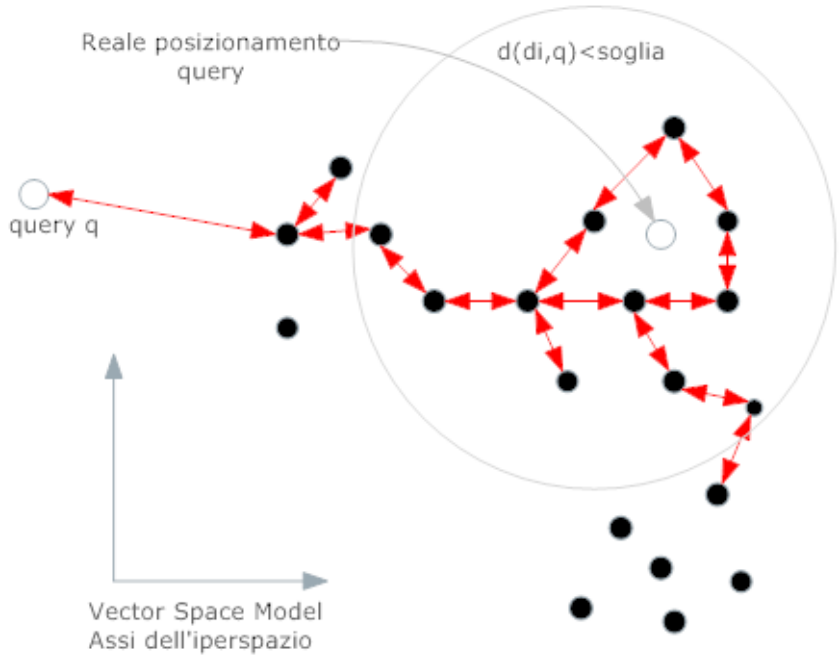

Figura 2-7 Modello Vettoriale

E' interessante introdurre il concetto di dipendenza tra termini considerando il fatto che gli assi in cui definiamo i nostri vettori possono non essere ortogonali tra loro, questo può essere un ottimo spunto per un lavoro futuro in cui viene considerato anche l'aspetto semantico.

Il modello vettoriale è quello che ci consente di trattare i dati tramite calcoli algebrici pertanto, in via teorica, è l'ambito dove si possono ottenere le prestazioni migliori.

3. **Modello probabilistico**: il modello probabilistico usa sostanzialmente la metrica di recall per fornire un ranking contenente i documenti rilevanti per una query.

   Quello che viene introdotto in questo modello è uno stimatore che stima appunto la probabilità relativa alla rilevanza. A posteriori poi è possibile valutare la sua efficienza attraverso dei dati di test su cui è possibile calcolare precisione e rilevanza.

   Pertanto si parte considerando i termini della query e dei documenti e tramite lo stimatore si valuta se la presenza di un termine è rilevante rispetto alla query espressa, cioè si stima:

   $$P(relevant|d_i, q) = F(d_i, q)$$

   Dove $F(d_i, q)$ è uno stimatore statisticamente corretto e consistente.

   Senza addentrarci nella costruzione di un tale stimatore è possibile valutare le assunzioni fatte in letteratura a riguardo.

   Innanzi tutto si assume che documenti e termini sono indipendenti tra loro, la rilevanza di un documento non implica la rilevanza di un documento "vicino" e questo è una limitazione rispetto ai modelli vettoriali.

   Infine i documenti vengono presentati classificati rispetto alla loro rilevanza stimata.

   A fronte di tutto questo in letteratura è ampiamente dimostrato come i modelli vettoriali forniscano prestazioni migliori rispetto a quelli probabilistici, vedi ad esempio (5).

2.1.7 Un esempio concreto: i moderni motori di ricerca

La ricerca sull'IR ha portato, nel corso degli anni 1990, a definire sistemi per l'accesso tramite query di documenti nella rete. Tali sistemi sono i moderni motori di ricerca che implementano la massima evoluzione delle tecniche appena esposte.

La prima distinzione che sorge è sul tipo delle query esprimibili. Come per i primi sistemi di IR la query può essere espressa in modo più o meno formale, in particolare con l'evoluzione dei motori di ricerca è possibile esprimere un proprio bisogno

informativo attraverso una query in forma "parlata" con in aggiunta operatori testuali e logici che possono aiutare a completare la richiesta.

Come aspetto laterale alla formalità delle query vi è la tipologia di bisogno espresso, in (6) vi è una chiara definizione:

1. **Query navigazionali**: esprimono l'intento immediato di raggiungere un sito particolare, esse vengono espresse specificando il brand o il nome del sito interessato.
2. **Query informative**: vengono espresse per raggiungere informazioni che si suppone presenti in forma di testo statico nella rete.
3. **Query transazionali**: specificano la necessità di raggiungere un dato servizio mediato tramite rete internet.

Come tutte le tecnologie informatiche anche i motori di ricerca hanno visto un'evoluzione: la prima generazione che ha operato dal 1994 fino al 1997 si basava in maniera molto diretta ed elementare su tecniche di IR appena viste, erano pertanto motori di ricerca che consideravano le pagine web solo dal punto di vista sintattico eventualmente supportate da parole chiave specificate dall'autore.

Nel 1998, con l'avvento di Google, sono stati introdotti elementi relativi alla reputazione che determina la Search Engine Result Page o SERP che viene mostrata dal motore di ricerca all'utente come ad esempio:

1. **Link Analysis**: dove attraverso l'algoritmo PageRank, derivato da HITS, vengono associati valori di reputazione diversi proporzionalmente ai link in ingresso e uscita alle singole pagine.
2. **Click Through Data**: la reputazione è tanto più alta quanto gli utenti scelgono dal ranking mostrato una fonte piuttosto che un'altra.
3. **Anchor Text**: assume importanza anche il testo che descrive i link tra le pagine.

Oltre a questi tre elementi basilari col tempo sono stati introdotti, attraverso vari brevetti, algoritmi che raffinano ulteriormente le ricerche o contrastano misure adottate per falsare la SERP.

Con l'avvento del web 2.0, a fine della decade 2000-2010 stanno vedendo la luce nuove proposte in senso semantico che pongono al centro dell'elaborazione l'utente e le informazioni che esso condivide con la rete. Proposte tuttavia per lo più ancora in fase di sviluppo.

2.2 La maturazione: Text Categorization

La nascita delle teorie e delle tecniche di TC si identifica con l'articolo del 1961 di Maron (7), parallelamente allo sviluppo di algoritmi di IR. Come descritto in (8) IR può essere visto come una sottoattività di TC: la classificazione può avvenire in base ad una query ossia ogni query esprime una classe di cui individuare i testi di interesse.

Sempre in (8) sono elencati i punti di contatto tra i due mondi. Quello che unisce le due tecniche è il fatto che entrambe dipendono dal contenuto dei documenti. In particolare nei metodi che sfruttano l'apprendimento viene attivato un processo induttivo, spesso di tipo greedy, per costruire un classificatore relativo ad una classe specifica. Questo processo viene dapprima istruito mediante "osservazione" di un insieme di testi appartenenti ad una data classe, i nuovi testi saranno quindi classificati "confrontando" le caratteristiche con quelle che identificano la classe.

IR condivide quindi alcune tecniche con TC: l'indicizzazione dei testi è effettuata per entrambi, inoltre molti algoritmi di TC usano tecniche come ad esempio il matching tra documenti o la logica che guida l'espansione delle query in IR. Infine la valutazione della qualità delle tecniche di IR o TC viene fatta condividendo le stesse metriche.

2.3 Metodi di Text Categorization

Viene qui proposta una rassegna di metodi di TC, si userà per quanto possibile una notazione omogenea tra i vari metodi. Essi sono stati suddivisi in base al background teorico che possiedono e alle caratteristiche specifiche di ogni tipologia.

Considerando un dominio di testi $D = \{d_1 \dots d_m\}$ e un codominio di classi $C = \{c_1 \dots c_p\} \cup \{c_q\}$ con in aggiunta una eventuale classe pozzo c_q, la relazione che in questo lavoro si è considerata come riferimento è una relazione suriettiva tra i due insiemi, dove ad un testo viene assegnata al più una classe. Questo consente di

semplificare le metriche per la valutazione dell'efficienza dell'algoritmo a fronte di un risultato tutto sommato in linea con la realtà. Non è escluso tuttavia che in alcune trattazioni la relazione tra testi e classi perda la proprietà di suriettività divenendo una relazione generica molti a molti, questo per completezza e per non precludere elaborazioni di maggior precisione.

2.3.1 Metodi caratteristici

Questi metodi sono così chiamati perché sfruttano caratteristiche proprie dei testi o delle classi. Sfruttano cioè informazioni in una certa misura intrinseche al dominio di analisi.

Alla base di questa esposizione si è voluto porre una notazione unica per i tre metodi qui esposti, pertanto si ha un insieme di vettori rappresentanti i testi:

$$\underline{d_i} \in D, i = 1 \dots m$$

Con D insieme dei testi, $m = |D|$, e un insieme di vettori rappresentanti le varie classi:

$$\underline{c_j} \in C, j = i \dots p$$

Con C insieme delle classi, $p = |C|$.

2.3.1.1 Rocchio

Questo algoritmo è storicamente il primo ad essere usato su larga scala in ambito TC. Esso consiste nel confronto sistematico tra vettori: quelli che rappresentano i testi e quelli che rappresentano genericamente le classi.

Per caratterizzare l'algoritmo bisogna scegliere tre elementi: il primo è il tipo di pesatura dei termini nel testo, la seconda scelta verte sul tipo di procedura usata per costruire i vettori rappresentanti le classi e infine bisogna scegliere una funzione di similarità per confrontare un vettore generico rappresentante un testo $\underline{d_i}$ e le varie classi $\underline{c_j}$.

In una prima fase di training pertanto si costruiscono i vettori prototipo che rappresentano le classi, avendo a disposizione dati di training già classificati, come

esposto in (9) si può ad esempio costruire i vettori prototipo banalmente tramite la somma dei vettori rappresentanti testi appartenenti ad una data classe:

$$\underline{c_j} = \sum_{\underline{d}\ t.c. classe(\underline{d})=\underline{c}} \underline{d}$$

Anche se ovviamente è sempre possibile scegliere funzioni diverse a riguardo.

Verrà fatto un confronto sui dati di test e successivamente su quelli da classificare tramite una funzione di similarità, in (9) viene ad esempio usata la funzione coseno:

$$classe(\underline{d}) = \underset{\underline{c}\in C}{\operatorname{argmax}}(\cos(\underline{d}, \underline{c}))$$

Riguardo la funzione coseno è riportato in Figura 2-8 un esempio di bontà nel confrontare due vettori di modulo diverso ma proporzionalmente afferenti alle stesse dimensioni.

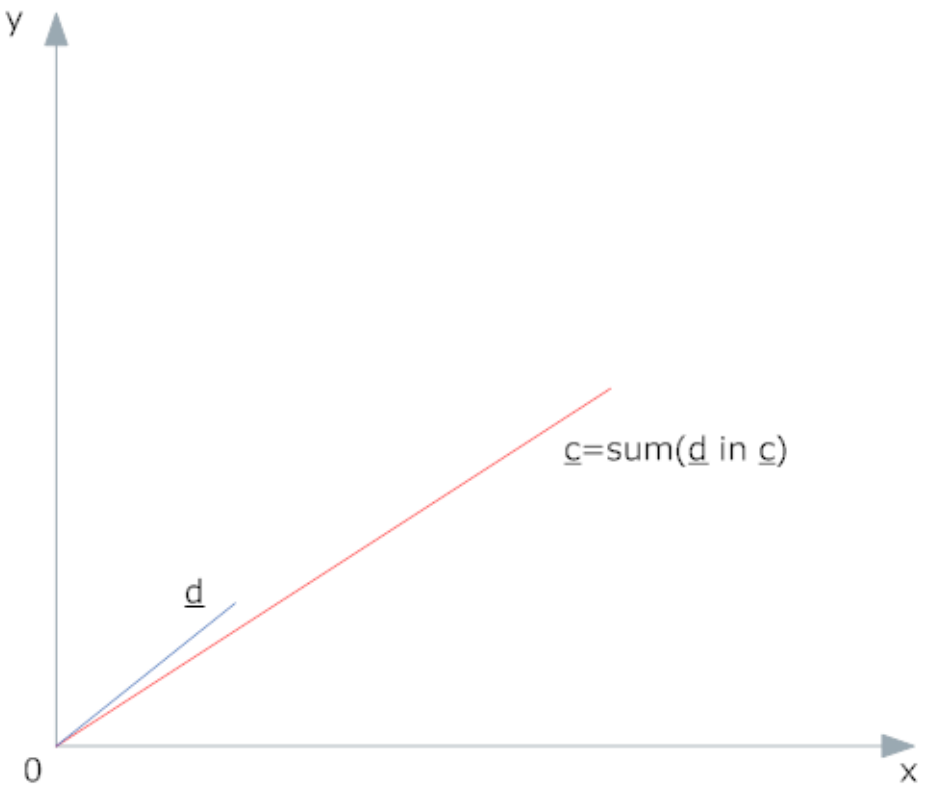

Figura 2-8 Cosine Similarity

In questo modo è possibile fare un confronto diretto tra il testo da classificare e tutte le classi considerate.

La brillante critica mossa in (8) mette bene in risalto i limiti di questo metodo, identificando un solo centroide, rappresentato da *c*, un testo verrà classificato sotto una data classe solo se ricade entro una certa "distanza" da *c*, si veda Figura 2-9. Questo può essere vero se lo spazio in cui si opera è lineare, essendo Rocchio un metodo lineare. Dal momento in cui questa ipotesi non sussiste bisogna usare una funzione kernel che permetta di linearizzare lo spazio con ovvie ricadute in termini prestazionali.

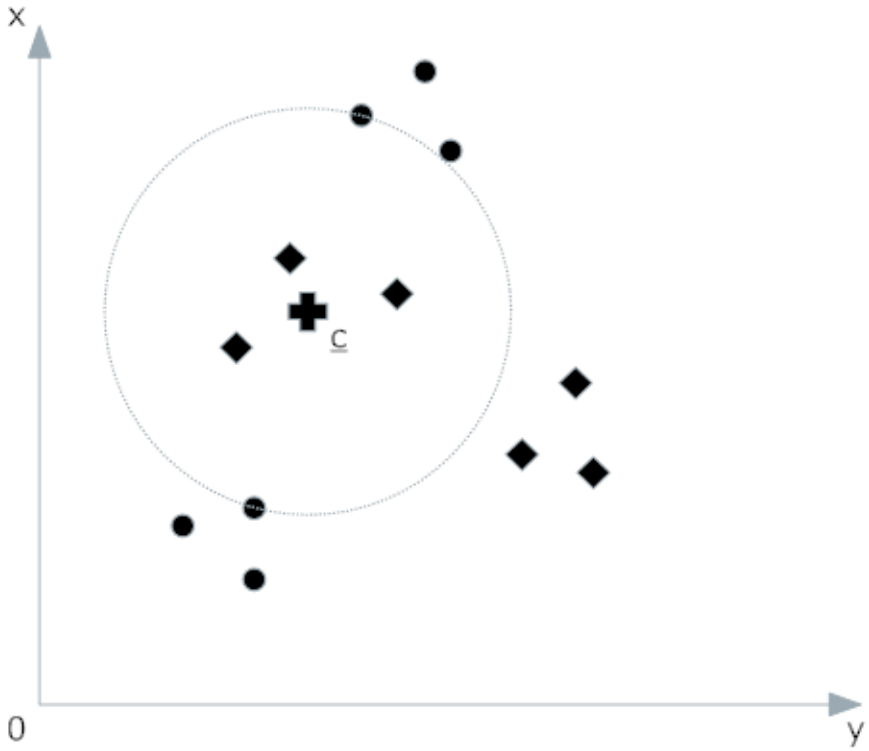

Figura 2-9 Limiti dell'Algoritmo Rocchio

In figura si può notare come il centroide *c* per la classe rappresentata dai testi "cerchietto" ricada in mezzo a testi di tipo "rombo", pertanto un testo simile a "rombo" verrà classificato come "cerchietto" dato che ricade vicino al centroide.

Questo effetto è però in parte mitigato dal fatto che le dimensioni dello spazio dipendono dai termini che se per assunzione semplificativa sono assunti come indipendenti tra loro in realtà non lo sono facendo diventare una situazione come quella di Figura 2-9 molto remota.

2.3.1.2 K-Nearest Neighbours: k-NN

Per superare i limiti dei classificatori lineari come Rocchio viene proposto k-NN. Esso è un classificatore "locale" nel senso che esula dall'uso di centroidi ma si basa solamente sulla situazione topologica.

k-NN innanzi necessita della scelta di un valore k, in (8) viene indicato $k = 30$ per ottenere buone prestazioni, una volta scelto questo valore si considerano i k testi più "vicini" a quello considerato come esposto in Figura 2-10:

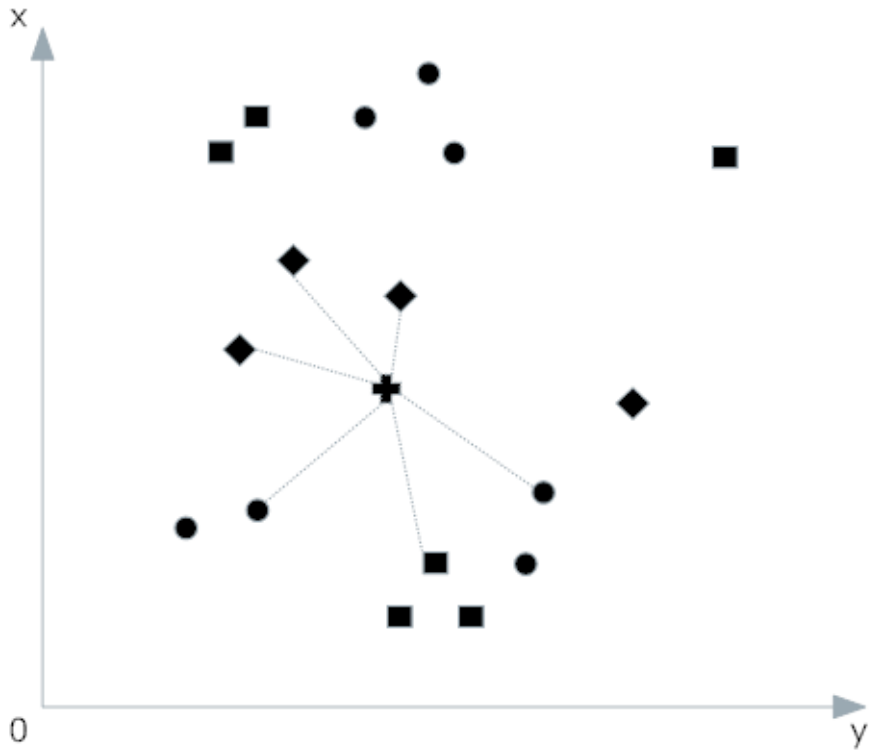

Figura 2-10 k-NN

Volendo esprimere in maniera rigorosa la definizione di una classe generica c_j sulla traccia della definizione di questo metodo:

$$c_j = \left\{\underline{d} \in D\ t.c. \forall \underline{d_i} \in D \wedge \forall \hat{\underline{d}} \in D \wedge classe\left(\underline{d_i}\right) = C_j \wedge classe\left(\hat{\underline{d}}\right) \neq C_j \wedge dist\left(\underline{d}, \underline{d_i}\right) < dist\left(\underline{d}, \hat{\underline{d}}\right)\right\}$$

$$c_1 \cap c_2 \cap \ldots \cap c_p = \{\quad\}$$

$$C = c_1 \cup c_2 \cup \ldots \cup c_p$$

Da cui si ricava una formula valida per questa classificazione che tiene conto della distanza tra i testi afferenti ad una data classe e $\underline{d}$:

$$classe(\underline{d}) = arg \min_{c_j \in C} \left(\sum_{\underline{d_j} \in D_k \; t.c. classe(\underline{d_j}) = C_j} dist\left(\underline{d}, \underline{d_j}\right) \right)$$

Dove D_k è l'insieme dei k testi più vicini a $\underline{d}$.

Come indicato in (10) k-NN non necessita di una fase di training vera e propria poiché può operare tranquillamente anche incrementalmente.

2.3.1.3 Naive-Bayes

Il classificatore Naive-Bayes è stato associato un po' impropriamente a Rocchio e a k-NN elencandolo nello stesso paragrafo, esso avrebbe dovuto essere annoverato tra i classificatori probabilistici ma viste le caratteristiche importanti ricercate ai fini di questo lavoro, il fatto di essere probabilistico è una caratteristica che viene considerata intrinseca al classificatore e nient'altro. Ossia le caratteristiche considerate importanti per questo classificatore coincidono con quelle del classificatore Rocchio e quindi i due classificatori sono stati associati.

L'assunzione di riferimento su cui si basa questo classificatore è esposta in (9) ossia che un testo può essere visto come un insieme di termini scelti da un dizionario tramite una distribuzione probabilistica. Ogni classe pertanto descrive proprio questa distribuzione di probabilità. Quello che fa questo classificatore è costruire degli stimatori per ogni classe per poi applicarli ad ogni testo.

Pertanto la quantità da stimare diventa:

$$P(c|\underline{d})$$

Ossia la probabilità che un testo $\underline{d}$ appartenga alla classe c. La scelta poi consiste nell'associare $\underline{d}$ alla classe con probabilità più alta:

$$classe(\underline{d}) = arg \max_{c \in C} \left(P(c|\underline{d}) \right)$$

Tramite il teorema di Bayes pertanto è possibile separare la stima:

$$P(c|\underline{d}) = \frac{P(\underline{d}|c) * P(c)}{\sum_{c \in C}\left(P(\underline{d}|c) * P(c)\right)}$$

Dove $P(c)$ è probabilità che un testo a priori sia di classe c, ossia in prima approssimazione la frequenza di testi appartenenti alla classe. Mentre $P(\underline{d}|c)$ è la probabilità di osservare un testo come $\underline{d}$ in c.

Lo stimatore per la prima probabilità è molto semplice ossia la mera frazione di testi appartenenti a c:

$$\hat{P}(c) = \frac{|\underline{d} \in c|}{|C|}$$

Mentre la seconda probabilità è leggermente più difficoltosa da stimare, lo stimatore si basa sull'assunzione che se un termine occorre in un testo questo dipende dalla classe di afferenza di quel testo, quindi in formule:

$$\hat{P}(\underline{d}|c) = \prod_{i=1}^{|\underline{d}|} P(w_i|c)$$

Ossia il prodotto tra le probabilità che il termine w_i appartenente a $\underline{d}$ compaia in testi di classe c. Questo è possibile tramite l'assunzione dell'indipendenza tra i termini che può essere fondata sulla scelta di una funzione di pesatura dei termini opportuna. Per stimare $P(w_i|c)$ si usa:

$$\hat{P}(w_i|c) = \frac{1 + |w_i \in \underline{d}\ t.c.\ \underline{d} \in c|}{|w \in F| + \sum_{w \in F}|w \in \underline{d}\ t.c.\ \underline{d} \in c|}$$

Dove F rappresenta il dizionario completo.

In (11) vengono esposte molto chiaramente le potenzialità e i difetti di questo metodo. Simulazioni Monte Carlo statistiche hanno dimostrato ottime prestazioni sotto due condizioni opposte, come atteso il metodo risponde bene quando vi è l'indipendenza tra i termini ma sorprendentemente esso è altrettanto efficiente anche quando i termini sono totalmente dipendenti.

Questo tuttavia rappresenta proprio il limite di questo metodo in quanto di norma all'interno di un testo vi è una dipendenza labile tra i termini.

2.3.2 Metodi algebrici

I metodi algebrici assumono particolare importanza dal momento che sono molto adatti all'esecuzione su un calcolatore, essi si basano sulla rappresentazione vettoriale di un testo e di conseguenza la possibilità di applicare trasformazioni algebriche e calcoli sistematici per poter classificare gli stessi.

2.3.2.1 Support Vector Machines, SVM

Tale metodo richiede basi matematiche e concettuali piuttosto avanzate, verrà esposto in una forma semplificata e maggiormente comprensibile, è sempre possibile fare riferimento alle fonti: (12), (13), (14), (15) e (16).

La prima assunzione che facciamo è relativa alla linearità delle dimensioni considerate: per noi le dimensioni che rappresentano il nostro vettore-testo sono indipendenti e lineari. Questo significa che a prescindere da quale sia il significato di una dimensione nel vettore relativo al testo (parola n-esima, pesatura contestuale, o altro) tutte le dimensioni dal punto di vista di SVM hanno un peso uguale.

La seconda assunzione è relativa alla linearità dei vettori tra loro: questo ci permette di affrontare il problema in modo separabile. Per quanto riguarda i testi questa assunzione impone che i testi siano sintatticamente indipendenti.

Entrambe queste assunzioni dal punto di vista del lavoro qui presentato sono ragionevoli in quanto considerando i testi a prescindere dal contesto e solo da un punto di vista meramente sintattico esse sussistono.

Iniziamo quindi a considerare un modello relativo ai dati a disposizione:

$\underline{x_i} \in \mathbb{R}^m$: vettore n-dimensionale relativo al testo i-esimo,

$\left(\underline{x_i}, y_i\right)$: vettore con una componente aggiuntiva y_i binaria $y_i \in \{-1,1\}$, questa componente rappresenta il fatto che il testo i-esimo appartiene o meno alla classe considerata.

Nella Figura 2-11 è presente una semplificazione bidimensionale del problema, si identifica una "linea" di separazione tra i vettori che hanno associato $y_i = 1$ o $y_i = -1$ rappresentando tale caratteristica in forma bidimensionale.

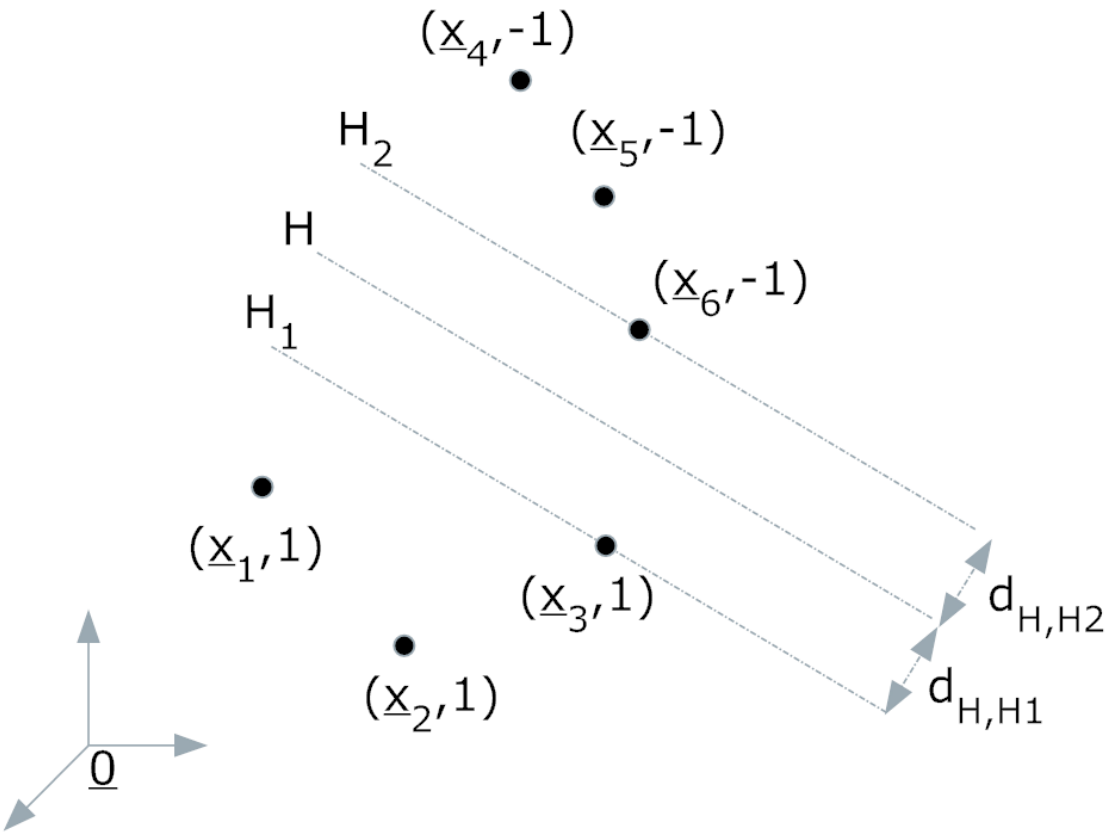

Figura 2-11 Rappresentazione Dello Spazio n-Dimensionale per SVM

SVM quindi ha il fine di determinare un iperpiano H, in Figura 2-11 rappresentato da una retta, che separi i testi con $y_i = 1$ e $y_i = -1$ sull'insieme di training per poi poter classificare nuovi testi in base alla posizione relativa assunta rispetto all'iperpiano H.

Formalmente pertanto si definisce l'iperpiano H come l'insieme dei punti $\underline{x}$ m-dimensionali:

$$\underline{x} \in H \Leftrightarrow \underline{x} * \underline{w} + b = 0$$

Dove $\underline{w}$ è il vettore direttore di H e b è lo scostamento dall'origine.

Ora, dati i punti $\underline{x_i}$, si può assumere che:

$$\underline{x_i} * \underline{w} + b \geq 1 \Rightarrow y_i = 1$$

$$\underline{x_i} * \underline{w} + b \leq -1 \Rightarrow y_i = -1$$

Dove i punti per cui:

$$\underline{x_i} * \underline{w} + b = 1$$

$$\underline{x_i} * \underline{w} + b = -1$$

Si impone siano i punti più vicini all'iperpiano H, che nella Figura 2-11 sono rispettivamente x_3 e x_6, pertanto nessun altro punto può avere una vicinanza ad H inferiore ad uno nelle equazioni sopra.

I punti per cui è verificata l'uguaglianza definiscono due nuovi iperpiani H_1 e H_2 paralleli ad H, cioè con lo stesso vettore direttore $\underline{w}$, che hanno scostamento dall'origine rispettivamente di $b - 1$ e $b + 1$.

Distanza tra H_1 e H_2

Dato un iperpiano K, la distanza che esso ha dall'origine sarà la distanza tra l'origine e il punto $\underline{x}$ sull'iperpiano dove cade la perpendicolare tra l'origine e l'iperpiano stesso come è possibile vedere in Figura 2-12:

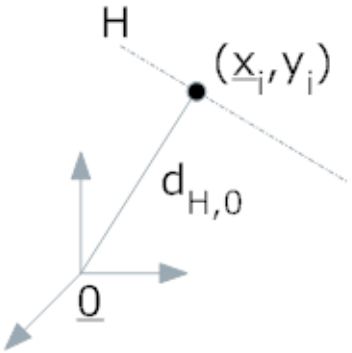

Figura 2-12 Distanza dall'Origine

In termini matematici sarà il punto che ha norma due minima e tale norma corrisponderà proprio alla distanza tra K e l'origine:

$$\underline{x} * \underline{w} + b = 0 \Rightarrow \underline{x} * \underline{w} = -b \Rightarrow \left\|\underline{x} * \underline{w}\right\| = |-b| \Rightarrow$$

$$\xrightarrow{\substack{disuguaglianza \\ triangolare}} \left\|\underline{x}\right\| * \left\|\underline{w}\right\| \geq |b| \Rightarrow \left\|\underline{x}\right\| \geq \frac{|b|}{\left\|\underline{w}\right\|} \xrightarrow{\substack{norma \\ minima}} d_{K,\underline{0}} = \frac{|b|}{\left\|\underline{w}\right\|}$$

Quindi genericamente la distanza di un iperpiano K dall'origine è pari al modulo dello scostamento dall'origine diviso la norma due del vettore direttore.

Per quanto riguarda la nostra dimostrazione abbiamo che:

$$d_{H,\underline{0}} = \frac{|b|}{\left\|\underline{w}\right\|}$$

$$d_{H1,\underline{0}} = \frac{|b-1|}{\left\|\underline{w}\right\|}$$

$$d_{H2,\underline{0}} = \frac{|b+1|}{\left\|\underline{w}\right\|}$$

Da cui si ricava che:

$$d_{H1,H2} = d_{H2,\underline{0}} - d_{H1,\underline{0}} = \frac{2}{\left\|\underline{w}\right\|}$$

Per avere una migliore qualità di classificazione è necessario quindi scegliere $\underline{w}$ in modo che il margine, ossia la fascia compresa tra H_1 e H_2 sia più estesa possibile:

$$\max_{(\underline{w},b)}\left(d_{H1,H2}\right) = \max_{(\underline{w},b)}\left(\frac{2}{\left\|\underline{w}\right\|}\right)$$

Ma questo deve valere a fronte dei vincoli visti relativi alla discriminazione:

$$\begin{cases} \underline{x_i} * \underline{w} + b \geq 1 \Rightarrow y_i = 1 \\ \underline{x_i} * \underline{w} + b \leq -1 \Rightarrow y_i = -1 \end{cases}$$

Da cui, moltiplicando entrambe le disequazioni ad entrambi i membri per y_i:

$$\begin{cases} y_i * \left(\underline{x_i} * \underline{w} + b\right) \geq y_i * 1 \wedge y_i = 1 \\ y_i * \left(\underline{x_i} * \underline{w} + b\right) \geq y_i * (-1) \wedge y_i = -1 \end{cases}$$

Questa trasformazione porta ad un'unica equazione di vincolo:

$$y_i * \left(\underline{x_i} * \underline{w} + b\right) \geq 1$$

La formulazione appena esposta ben si adatta al problema che porta alla definizione del moltiplicatore Lagrangiano.

Formulazione Lagrangiana basata sul moltiplicatore Lagrangiano

Viene ora fornita un'esposizione della formulazione Lagrangiana relativa a un problema di massimizzazione su due dimensioni, questa elaborazione la chiameremo nel seguito per brevità "problema Lagrangiano" sebbene il suo vero nome sia quello esposto in precedenza in grassetto.

Dato un problema di massimizzazione:

$$\max_{(x,y)\in A} \big(f(x,y)\big)$$

$$A = \{(x,y) \in R^2 \; t.c. \; g(x,y) = c \; \wedge c \in \mathbb{R}\}$$

Dobbiamo quindi massimizzare una funzione in tre dimensioni $f(x,y)$ muovendoci sulla curva di livello di un'altra funzione $g(x,y) = c$, cioè trovare punti di massimo su $f(x,y)$ stando nell'insieme A.

Questo significa che muovendoci lungo $g(x,y) = c$ identifico un punto di massimo o minimo quando i vettori tangenti sono paralleli, cioè quando:

$$\left(\frac{\partial f(x,y)}{\partial x} // \frac{\partial g(x,y)}{\partial x}\right) \wedge \left(\frac{\partial f(x,y)}{\partial y} // \frac{\partial g(x,y)}{\partial y}\right) \wedge (x,y) \in A$$

Ossia quando i gradienti:

$$\nabla_{x,y} f(x,y) = -\lambda * \nabla_{x,y} g(x,y), \lambda \in \mathbb{R}$$

Il parametro λ è richiesto perché se i gradienti sono paralleli il loro modulo può non essere uguale.

Definendo il moltiplicatore Lagrangiano:

$$\Lambda(x, y, \lambda) = f(x, y) + \lambda * (g(x, y) - c)$$

Si possono trovare soluzioni al problema iniziale risolvendo:

$$\nabla_{x,y,\lambda}\Lambda(x, y, \lambda) = 0$$

Questa digressione relativamente al problema Lagrangiano è particolarmente utile per due motivi: il primo è che il vincolo $y_i * \left(\underline{x_i} * \underline{w} + b\right) \geq 1$ è facilmente compatibile col vincolo del problema Lagrangiano, il secondo è che i dati di training compaiono in forma di prodotto scalare tra vettori.

Definiamo quindi il problema SVM compatibile col problema Lagrangiano: questo problema si trasforma in m problemi singoli da risolvere e da cui scegliere i parametri a posteriori tramite una semplice considerazione.

$$\left(\underline{x_i}, y_i\right)\ i = 1 \dots m$$

Sono i dati di training, mentre le variabili secondo cui risolvere il problema sono:

$$\underline{w}, b_i\ i = 1 \dots m$$

Quindi la formulazione diventa:

$$\max_{(\underline{w},b)\in A}\left(\frac{2}{\|\underline{w}\|}\right) = \min_{(\underline{w},b)\in A}\left(\|\underline{w}\|\right) = \min_{(\underline{w},b)\in A}\left(\frac{\underline{w}^2}{2}\right)$$

$$b = \min_{i=1\dots m}(|b_i|)$$

$$A_i = \left\{\left(\underline{w}, b_i\right) \in \mathbb{R}^{n+1}\ t.c.\ y_i * \left(\underline{x_i} * \underline{w} + b_i\right) = 1\right\}\ i = 1 \dots m$$

Il parametro b_i è stato inserito per ottenere l'uguaglianza nell'espressione del vincolo sull'insieme A, in seguito verrà scelto come scostamento dall'origine dell'iperpiano H il valore di b_i con modulo minimo. Questa scelta non inficia il risultato finale in quanto il vincolo espresso in realtà è una disuguaglianza che con valori inferiori di scostamento

sarà comunque verificata. Inoltre l'espressione $\underline{w}^2$ significa calcolare il prodotto scalare di $\underline{w}$ con se stesso.

In analogia col problema di Lagrange abbiamo che:

$$f(x,y) = \frac{\underline{w}^2}{2}$$

$$g(x,y) = y_i * \left(\underline{x_i} * \underline{w} + b_i\right)$$

$$c = 1$$

Costruisco quindi il moltiplicatore Lagrangiano che risolve tutti i problemi con vincolo A_i introducendo i parametri $\lambda_i \; i = 1 \dots m$:

$$\Lambda(\underline{w}, \underline{b}, \underline{\lambda}) = \frac{\underline{w}^2}{2} - \sum_{i=1}^{m} \left\{\lambda_i * \left[y_i * \left(\underline{x_i} * \underline{w} + b_i\right) - 1\right]\right\} =$$

$$= \frac{\underline{w}^2}{2} - \sum_{i=1}^{m} \left[\lambda_i * y_i * \left(\underline{x_i} * \underline{w} + b_i\right)\right] + \sum_{i=1}^{m} \lambda_i$$

Questo significa che calcolando il gradiente di $\Lambda(\underline{w}, \underline{b}, \underline{\lambda})$:

$$\langle \frac{\partial \Lambda(\underline{w}, \underline{b}, \underline{\lambda})}{\partial \underline{w}}, \frac{\partial \Lambda(\underline{w}, \underline{b}, \underline{\lambda})}{\partial \lambda_i}, \frac{\partial \Lambda(\underline{w}, \underline{b}, \underline{\lambda})}{\partial b_i} \rangle =$$

$$= \langle \underline{w} - \sum_{i=1}^{m} \left(\lambda_i * y_i * \underline{x_i}\right), -y_i * \left(\underline{x_i} * \underline{w} + b_i\right) + 1, \lambda_i * y_i \rangle$$

Imponendo che tale gradiente sia nullo si ottiene il sistema:

$$\begin{cases} \underline{w} = \sum_{i=1}^{m} \left(\lambda_i * y_i * \underline{x_i}\right) \\ y_i * \left(\underline{x_i} * \underline{w} + b_i\right) = 1, i = 1 \dots m \\ \lambda_i * y_i = 0, i = 1 \dots m \end{cases}$$

In particolare la seconda equazione esprime i vincoli A_i denotando quindi la piena compatibilità tra il problema Lagrangiano esposto e il problema SVM mentre la terza equazione è esprimibile in forma più estesa e meno restrittiva come:

$$\sum_{i=1}^{m} (\lambda_i * y_i) = 0$$

In quanto esprime un vincolo sui coefficienti b_i che verranno poi rilassati nella scelta di un unico b tramite una restrizione sul valore.

Il calcolo della funzione Lagrangiana si trasforma quindi in un problema duale:

$$\Lambda_D(\underline{w}, \underline{b}, \underline{\lambda}) = \sum_{i=1}^{m} \lambda_i - \frac{1}{2} * \sum_{i=1}^{m} \left[\sum_{j=1}^{m} \left(\lambda_i * \lambda_j * y_i * y_j * \underline{x_i} * \underline{x_j} \right) \right]$$

Una volta fissati quindi i valori di $\underline{w}$ e b è possibile discriminare nuovi testi $\underline{x}$ calcolando molto semplicemente:

$$\underline{x} * \underline{w} + b \lessgtr 0$$

Computazionalmente, una volta a conoscenza di $\underline{w}$ e b, la classificazione costa $O(n)$ ossia il solo costo del prodotto scalare tra vettori.

Tramite modifiche che prevedono l'uso di uno scostamento variabile $b \pm \xi$ è possibile rendere l'algoritmo valido anche per dati non separabili e in letteratura viene introdotto l'uso di una funzione Kernel per la trattazione di dati non lineari in quanto come si vede dalla formula del problema duale è presente il prodotto scalare $\underline{x_i} * \underline{x_j}$, questo è trattato in (14) e (15).

L'uso di Kernel e i calcoli proposti tuttavia portano ad una soluzione numerica degli stessi che quindi è necessariamente onerosa. Questo può essere tollerabile se vi sono due fasi di training e di test ben separate, inizia invece a diventare problematico quando la classificazione avviene su più classi e i testi continuano a fluire così come proposto nel presente lavoro.

Tuttavia costruendo classificatori per ogni classe obiettivo, come indicato in (14) SVM ottiene risultati qualitativi sensibilmente migliori rispetto agli alberi decisionali che ad oggi offrono i risultati qualitativamente migliori.

2.3.2.2 Latent Semantic Indexing, LSI

Questo metodo è un metodo di IR ma, con le dovute osservazioni, è adottabile come metodo TC. Vi è la dicitura "latente" perché tramite trasformazioni matematiche mira a cogliere la struttura latente nella matrice testi-termini, si annovera pertanto anche tra gli algoritmi di data mining.

I vettori che rappresentano i testi accostati per colonna nella matrice testi-termini, vedi Figura 3-4, possono essere proiettati in un nuovo spazio con dimensioni ridotte ottenuto tramite la decomposizione a valori singolari della matrice R. Questo spazio ha come basi gli auto-vettori di $R^T * R$ a cui corrispondono gli auto-valori più grandi associati alla matrice ottenuta dal prodotto $R^T * R$.

La decomposizione a valori singolari

Quella che qui vedremo è la decomposizione a valori singolari corrispondente a quella che si ottiene col comando:

```
svd(R,0)®
```

di Matlab®, ossia la versione "economy size" su valori reali, tale decomposizione esiste anche sul campo complesso $\mathbb{C}$. Per informazioni più dettagliate consultare la guida di Matlab®.

Presa una matrice $R \in \mathbb{M}_{\mathbb{R}}^{n\times m}$ di rango $r(R) = r \leq \min(\mathrm{n}, \mathrm{m})$

Essa può essere decomposta in:

$$R = U * \Lambda * V^T$$

Dimensionalmente:

$$(n \times m) = (n \times r) * (r \times r) * (r \times m)$$

Dove

$$U \in \mathbb{M}_{\mathbb{R}}^{n \times r}$$

È la matrice le cui colonne sono i vettori singolari sinistri, cioè gli auto-vettori di $R * R^T$, queste colonne sono ortonormali, cioè:

$$\underline{u_i} * \underline{u_j} = 0, \forall i \neq j$$

$$\underline{u_i} * \underline{u_i} = 1, i = 1 \dots r$$

Da cui:

$$U^T * U = I_r.$$

Mentre

$$V \in \mathbb{M}_{\mathbb{R}}^{m \times r}$$

È la matrice le cui colonne sono i vettori singolari destri, cioè gli auto-vettori di $R^T * R$, anche le colonne di V sono ortonormali.

Infine

$$\Lambda = \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{bmatrix}$$

È una matrice diagonale che contiene di valori singolari di R cioè gli auto-valori di $R * R^T$ con $\sigma_1 \geq \sigma_2 \geq \cdots \sigma_r$.

LSI nello specifico

Definita la decomposizione SVD è possibile scegliere un valore $s < r$ e calcolare una matrice approssimante:

$$R_s = U_s * \Lambda_s * V_s^T$$

Con

$$\Lambda_s = \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_s \end{bmatrix}$$

Matrice dei primi $s < r$ valori singolari di R più grandi e di conseguenza:

$$U_s = \left[\underline{u_1} \dots \underline{u_s}\right] \in \mathbb{M}_{\mathbb{R}}^{n \times s}$$

$$V_s = \left[\underline{v_1} \dots \underline{v_s}\right] \in \mathbb{M}_{\mathbb{R}}^{m \times s}$$

Pertanto R_s ha rango $s < r$.

Questa trasformazione la chiameremo s-LSI.

Teorema di Eckart Young (17)

La matrice R_s è particolare perché vale il teorema che afferma che tra tutte le matrici $m \times n$ di rango al più $s < r$ essa è quella che approssima meglio la matrice originale, ossia minimizza la norma:

$$\|R - A\|_F^2 = \sum_{i=1}^{m} \sum_{j=1}^{n} \left(R_{i,j} - A_{i,j}\right)$$

$$\forall A \in \mathbb{M}_{\mathbb{R}}^{m \times n}\ t.c.\ r(A) \leq s\ \wedge\ R_s = U_s * \Lambda_s * V_s^T \Rightarrow \|R - R_s\|_F^2 \leq \|R - A\|_F^2$$

Prima di addentrarci nella spiegazione del perché tale decomposizione porta a delineare in maniera naturale delle classi di afferenza dei testi che caratterizzano la matrice R è bene fare un paio di considerazioni.

Innanzi tutto bisogna considerare la complessità computazionale della decomposizione SVD cui, volendo dare una definizione rigorosa è:

$$\forall A \in \mathbb{M}_{\mathbb{R}}^{n \times m} \Rightarrow \left(m \geq n \wedge O(m * n^2)\right) \vee \left(m < n \wedge O(n * m^2)\right)$$

Inoltre come stabilito dal teorema Eckart Young LSI mantiene il più possibile la struttura latente della matrice originale a patto di scegliere ragionevolmente s.

Come rilevato in (18) se i testi trattano temi simili LSI permette di ottenere risultati molto precisi in fase di ricerca, questo dal punto di vista della TC può sembrare poco rilevante ma tale precisione è sintomo di una naturale classificazione dei testi.

Questa affermazione è giustificata dal fatto che un dato tema è una distribuzione di probabilità su tutti i termini di un dizionario. Pertanto vi è una correlazione tra temi e termini.

In questa direzione viene in aiuto il lemma esposto in (18):

Data $R \in \mathbb{M}_{\mathbb{R}}^{nxm}$ a rango $r(R) = r$ con decomposizione SVD:

$$R = U * \Lambda * V^T$$

Con

$$\Lambda = \begin{bmatrix} \sigma_1 & & & & & \\ & \ddots & & & & \\ & & \sigma_s & & & \\ & & & \sigma_{s+1} & & \\ & & & & \ddots & \\ & & & & & \sigma_r \end{bmatrix}$$

E $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r$ se si ha che: $\frac{\sigma_s}{\sigma_{s+1}} > c * \frac{\sigma_1}{\sigma_s}$, $c \gg 1$ cioè in breve: $\sigma_s \gg \sigma_{s+1}$ in proporzione ai valori degli altri auto-valori. Se definisco una matrice di perturbazione $F \in \mathbb{M}_{\mathbb{R}}^{nxm}$ cioè $\|F\|_2 < \varepsilon$ con $0 < \varepsilon \ll 1$ e prendo la matrice $R' = R + F$ con decomposizione SVD:

$$R' = U' * \Lambda' * V'^T$$

Le matrici U_s e U'_s sono le matrici formate dalle prime s colonne di U e U' allora $U'_s = U_s * P + Q$ con $P \in \mathbb{M}_{\mathbb{R}}^{kxk}$ matrice ortonormale di permutazione e $Q \in \mathbb{M}_{\mathbb{R}}^{nxk}$ con $\|Q\|_2 \leq o(\varepsilon)$.

Questo significa che, data l'omotetia tra la matrice R e $V_s * \Lambda_s$ e i relativi auto-valori dati dalle colonne di U_s, perturbando la matrice R si ottiene uno spazio quasi-omotetico all'originale generato dalle colonne di U'_s. Il "quasi" è dato da una variazione Q che è proporzionale alla perturbazione.

Si fa notare come la norma 2 (o norma di Frobenious) di matrice è data da:

$$A \in \mathbb{M}_{\mathbb{R}}^{mxn}, \|A\|_2 = \|A\|_F = \sqrt{tr(A * A^T)}$$

Dire pertanto che la norma 2 di una matrice è piccola, come sostenuto per la matrice F, significa dire che essa descrive una trasformazione che genera vettori quasi nulli e quindi sommata ad un'altra matrice $R + F$ lo spazio finale sarà relativo ad una trasformazione simile a quella descritta da R.

La validità di questo lemma nel nostro caso va letta "al contrario", ossia una trasformazione s-LSI porta a valutare indirettamente in modo stabile una matrice R a fronte di uno scostamento Q controllabile dovuto all'approssimazione di R con la trasformazione s-LSI.

Quindi data una matrice R e la sua s-LSI trasformazione e un insieme di classi C in cui classificare i testi si ottiene $s = |C|$ proprio perché il valore di s identifica il numero di auto vettori linearmente indipendenti che conservano maggiormente le caratteristiche di trasformazione geometrica effettuata dal prodotto della matrice R con un vettore. Pertanto tali s-"direttrici" sono un insieme di vettori che raccogliendo le caratteristiche fondamentali di R rappresentano anche i vettori "prototipo" a cui fanno riferimento i vettori di R.

Dato che il valore di prodotto scalare tra due vettori rappresentanti due testi è correlato al coseno dell'angolo compreso tra i due documenti si può ricavare:

$$\underline{v_1} * \underline{v_2} = \left|\underline{v_1}\right| * \left|\underline{v_2}\right| * \cos\varphi \Rightarrow \cos\varphi = \frac{\underline{v_1} * \underline{v_2}}{\left|\underline{v_1}\right| * \left|\underline{v_2}\right|}$$

Con $\underline{v_1}, \underline{v_2} \in V * \Lambda$ e φ angolo incidente tra $\underline{v_1}$ e $\underline{v_2}$, si ricava quindi:

$\cos\varphi \cong 1$ se sono paralleli e quindi rappresentanti due testi simili,

$\cos\varphi \cong 0$ se sono ortogonali e quindi rappresentanti due testi di temi diversi.

Questo vale, in forza delle osservazioni precedenti, anche per la trasformazione s-LSI di R dove il prodotto $\underline{v_1} * \underline{v_2}$ non costa più $O(r)$ ma $O(s)$ con $s < r$.

Come esposto dai dati presenti in (18) si ottengono dei valori più coerenti per quanto riguarda il prodotto scalare con quanto esposto precedentemente con la trasformazione s-LSI piuttosto che con la matrice originale. Diminuendone anche la deviazione standard e la complessità computazionale del prodotto.

Per quanto riguarda i sinonimi si ottiene che due termini che sono sinonimi compaiono grosso modo lo stesso numero di volte nei testi e con la stessa distribuzione determinando due righe simili nella matrice R. Questo porta ad ottenere un valore singolare molto piccolo relativo a tali termini e quindi ad escludere naturalmente l'effetto di termini sinonimi nella trasformazione s-LSI.

La classificazione naturale dei testi viene fuori tramite la scomposizione della matrice R in sotto-matrici che rispettano le osservazioni fatte in cui i testi che appartengono alla stessa classe tenderanno ad avere prodotto scalare vicino a uno sui vettori normalizzati questo anche a fronte di perturbazioni sulla matrice R, bisogna quindi effettuare un'analisi spettrale sotto perturbazione.

In pratica si tratta di calcolare il prodotto scalare reciproco tra tutti i vettori e considerare della stessa classe quei vettori che hanno prodotto pari circa a uno con costo per ogni prodotto di $O(r), r \leq min(m,n)$ senza riduzione LSI, oppure di $O(s), s \leq r$ con riduzione LSI.

2.3.3 Metodi contestuali

I metodi di classificazione contestuali sono sostanzialmente classificatori che fanno uso di regole la quale tengono conto del "contesto" relativo ad un termine. Per contesto si possono intendere ovviamente molte cose ma in prima approssimazione esso consiste nei termini "vicini" a quello considerato.

Sono altresì definibili come classificatori non lineari che quindi tengono conto della dipendenza del termine con i termini che co-occorrono nel testo. In (19) viene indicato che l'assunzione dell'indipendenza dei termini è poco realistica, questo può essere vero per grossi testi, ma per testi brevi e di indirizzo generale vale ancora?

2.3.3.1 RIPPER

Questo algoritmo considera un insieme di regole da applicare ad un dato testo per poterlo classificare.

Le regole sono semplici formule logiche di secondo ordine con co-implicazione, volendo esprimere in maniera generale possiamo scrivere:

$$\bigwedge_{i \in W} (w_i \in d_j) \bigwedge_{i \in W} (w_i \notin d_j) \Leftrightarrow d_j \in c_k \, , d_j \in D, c_k \in C$$

Dove:

w_i è il termine i presente nei testi dato che l'intero insieme di parole è W.

d_j è il testo j dato che l'intero insieme dei testi è D.

c_k è la classe k nell'intero insieme di classi C.

Quindi una regola è un insieme di disgiunzioni che co-implica l'appartenenza di un testo ad una data classe.

RIPPER si compone di due stadi, un primo stadio in cui si combinano le varie regole in modo da creare grossi insiemi di regole, RuleSet, con la quale classificare i testi. Questo perché se immaginiamo di avere molte regole "sparse" più di una di esse può essere applicata ad un testo o addirittura si possono ottenere regole in contrasto tra loro, attraverso i RuleSet si crea sostanzialmente una grande regola che permette di associare un testo dato ad una classe. Un RuleSet pertanto sarà una regola come le altre con la differenza di essere più corposa e rappresentante un riassunto relativo all'assegnamento del testo ad una classe particolare.

Il primo stadio è esposto in pseudo-codice:

```
Function IREP*(data)
{
        ruleset:={}
        while exist f.b.f. senza "not" in data
        {
                split(data, growdata, prunedata)
                rule:=growrule(growdata)
                rule:=prunerule(rule, prunedata)
                ruleset:=ruleset unito {rule}
                data = data – disgiunzioni(rule)
                if DL(ruleset) > DL(ruleset(o))+d
                {
                        ruleset:=compress(ruleset,data)
                        return ruleset
                }
        }
        ruleset:=compress(ruleset,data)
        return ruleset
}
```

Questa funzione è una versione modificata dell'algoritmo IREP (Incremental Reduced Error Pruning), un algoritmo di set-covering di tipo greedy.

Vengono fornite in ingresso un insieme di regole estratte tramite una procedura descritta in seguito, l'algoritmo continua a operare fin tanto che esistono regole in formato non-negato nell'insieme "data": avere solo regole di tipo negato non aggiunge alcuna informazione rilevante al ruleset.

Viene quindi eseguito uno split casuale delle regole esistenti, l'insieme "growdata" conterrà i due terzi delle regole dell'insieme "data", l'insieme "prunedata" conterrà le restanti.

La regola "rule" viene quindi fatta crescere (growrule) aggiungendo ad ogni ciclo una singola disgiunzione appartenente all'insieme "growdata", la disgiunzione scelta è tale che massimizza il guadagno informativo:

$$Gain(rule_{i+1}, rule_i) = P_{i+1} * (-\log_2\left(\frac{P_i}{P_i + N_i}\right) + \log_2\left(\frac{P_{i+1}}{P_{i+1} + N_{i+1}}\right)$$

Dove P_i sono il numero di disgiunzioni non negate presenti in "growdata" coperte dalla regola i, mentre N_i sono il numero di disgiunzioni negate presenti in "growdata" coperte dalla regola i.

Dopo averla fatta crescere viene semplificata, o potata (pruned), secondo un altro procedimento greedy che cancella ogni sequenza finale di disgiunzioni dalla regola e sceglie la cancellazione che massimizza la funzione:

$$Loss(rule_{i+1}) = \frac{P_{i+1} - N_{i+1}}{P_{i+1} + N_{i+1}}$$

Dove P_i sono il numero di disgiunzioni non negate nel pruning set coperte dalla regola i e N_i sono il numero di disgiunzioni negate nel pruning set coperte dalla regola i.

La nuova regola calcolata viene quindi unita al RuleSet e dall'insieme "data" vengono tolte le regole coperte dalla regola appena aggiunta.

Il requisito che $Gain(rule_{i+1}, rule_i)$ sia il più grande possibile implica che il guadagno in ogni evoluzione di una regola sia positivo ossia che ogni nuova regola copra disgiunzioni non negate e, valutando la condizione espressa nel while, che l'algoritmo termini.

Tuttavia in presenza di dati con errore è possibile che vengano costruiti ResultSet molto grandi usando un numero limitato di disgiunzioni e questo può risultare computazionalmente costoso.

Viene quindi introdotta un'euristica che determina se la regola corrente è grande abbastanza rispetto i dati di training, questa euristica è chiamata MDL (Minimum Description Length). MDL si basa sull'idea che il modello migliore costruito su un set di dati è quello che lo esprime in maniera più compatta. Questo perché nella codifica subentrano sia i dati relativi al modello che quelli relativi all'errore, pertanto più è compatta la codifica meno i dati relativi all'errore sono numerosi.

Si rende pertanto necessario avere una metrica di misura della codifica del modello relativo al RuleSet, tale metrica in (19) è banalmente il numero di bit usati.

Attraverso MDL è possibile creare una regola di arresto che ferma la creazione del RuleSet nel caso in cui il nuovo RuleSet differisca di d bit rispetto al RuleSet con la codifica più corta.

Dopo di che viene effettuata la compressione eliminando quelle regole che allungano la codifica del RuleSet.

RIPPER quindi passa al secondo stadio, ossia lo stadio di ottimizzazione.

Lo stadio di ottimizzazione si limita a rivedere le regole creando delle versioni più efficienti in base alla codifica e al numero di disgiunzioni presenti ed eventualmente sostituendole nel RuleSet.

Queste due fasi come indicato in (19) sono efficaci se alternate due volte nella loro esecuzione. Risultato provato sperimentalmente su grosse collezioni di dati.

2.3.3.2 Sleeping-Experts

Questo algoritmo può essere visto come un'evoluzione generica di altri algoritmi possibili. L'idea che sta alla base è quella di combinare la predizione di molteplici "esperti" riguardo all'assegnamento di un testo ad una classe. Teoricamente questi "esperti" possono essere algoritmi di classificazione di qualunque tipo, anche diversi tra loro i cui pareri poi vengono assemblati da un algoritmo principale tramite pesatura.

I pesi per la combinazione dei vari pareri vengono quindi aggiornati tramite algoritmi moltiplicativi che sono empiricamente migliori, si tratta quindi di un algoritmo con apprendimento.

Come per tutti gli algoritmi che prevedono apprendimento anche l'algoritmo sleeping-experts soffre del problema dell'overfitting, come riportato in (19) tanti "esperti" rendono l'algoritmo molto preciso ma per converso poco capace di fornire una buona generalizzazione. Chiaramente con pochi esperti la situazione si ribalta.

Entrando nel dettaglio dell'algoritmo esso si fonda su due pilastri fondamentali: il primo è un algoritmo per l'assegnamento dei pesi, chiamato Hedge, che viene scelto per aggiornare i pesi ad ogni iterazione di sleeping-experts. Il secondo è il modello "infinite attribute model" che impone l'attivazione di alcuni "esperti" utili alla classificazione dei

testi in ingresso mentre ignora il parere di altri che vengono considerati dormienti (sleeping).

Prima di vedere lo pseudocodice dell'algoritmo è bene definire alcuni valori:

1. $\beta\epsilon(0,1)$: penalizzazione che si assegna agli "esperti" che non predicono correttamente.
2. $\theta\epsilon(0,1)$: soglia che discrimina l'appartenenza di un testo ad una classe data.
3. $D = \{d_1, d_2 \dots d_n\}$: insieme dei testi.
4. $\underline{d_i} = (w_1, w_2 \dots w_m)$: singolo testo, visto come vettore di termini.
5. r: coefficiente di permutazione per la creazione di nuovi "esperti".
6. P: insieme di Pool.

Function sleeping-experts(β,θ,r,D)
{
P:={}
for i=1…n
{
W:={$\underline{w}$ t.c. $\underline{w}$=(w_{k1},w_{k2}…w_{kp}) ∧ k1<k2…<kp<m ∧ kp-k1<r}
E:={($\underline{w}$,k) t.c. $\underline{w}$∈W ∧ k∈{0,1}}
for $\underline{w}$∈w
for k=0,1
if(~(($\underline{w}$,k)∈P))
$p_{\underline{w},k}$=1

$$y = \frac{\sum_{\underline{w}\in W} p_{\underline{w},1}}{\sum_{\underline{w}\in W} p_{\underline{w},0} + \sum_{\underline{w}\in W} p_{\underline{w},1}}$$

if y>θ
{
classe(d)=C
for $\underline{w}$∈W
$p'_{\underline{w},0}$=β*$p_{\underline{w},0}$
} else {
for $\underline{w}$∈W
$p'_{\underline{w},1}$=β*$p_{\underline{w},1}$
}

$$p_{\underline{w},0} = \frac{\sum_{\underline{w}\in W} p_{\underline{w},0}}{\sum_{\underline{w}\in W} p'_{\underline{w},0}} * p'_{\underline{w},0}$$

$$p_{\underline{w},1} = \frac{\sum_{\underline{w}\in W} p_{\underline{w},1}}{\sum_{\underline{w}\in W} p'_{\underline{w},1}} * p'_{\underline{w},1}$$

```
                P:=P∪E
        }
}
```

Sleeping-Experts, come detto, classifica i testi combinando il parere di più "esperti".

Analizzando lo pseudo-codice si può vedere come l'insieme:

$$W = \{\underline{w}\ t.c.\ \underline{w} = (w_{k1}, w_{k2} \dots w_{kp}) \wedge k1 < k2 \dots kp < m \wedge kp - k1 < r\}$$

contiene tutti i vettori formati dalla scelta di termini consequenziali di distanza al più r che sono quindi le frasi attive. Se $r = 1$ le frasi attive coincidono con i termini che compaiono nel testo.

Poi viene creato l'insieme degli "esperti" E: dove ogni "esperto" è rappresentato dalla coppia $(\underline{w}, k)$ dove w è la frase che rappresenta l'esperto in questione e $k \in \{0,1\}$ è il parere riguardo all'appartenenza del testo alla classe. Quindi ogni frase $\underline{w}$ nell'insieme degli esperti inizialmente predirà sia l'appartenenza $k = 1$ che la non appartenenza $k = 0$ del testo alla classe.

In seguito vengono calcolati dei pesi $p_{\underline{w},k}$ che rappresentano la bontà della predizione. Se la predizione non appartiene al Pool allora viene valutata come veritiera con massimo grado $p_{\underline{w},k} = 1$ e in seguito sarà corretta in base alle predizioni future.

Si calcola quindi un valore y che, confrontato con Θ, identifica l'appartenenza o meno del testo alla classe in oggetto. Per quanto riguarda il calcolo di y esso viene fatto sull'insieme W a cui corrisponde l'insieme degli "esperti" che duplica ogni frase assumendo che predica l'appartenenza o la non appartenenza del testo alla classe.

Quindi, con riferimento alla Figura 2-13, si ha che alcuni elementi all'interno dell'insieme degli "esperti" facevano già parte del Pool relativo a testi precedenti e essi predicevano o meno l'appartenenza alla classe data. Questa relazione tra E e P può essere scomposta in base al valore di k associato ad ogni elemento. Si ha quindi che E_1 e P_1 sono gli insiemi relativi agli "esperti" con $k = 1$, E_0 e P_0 sono relativi a $k = 0$. Per cui identificare il peso $p_{\underline{w},k}$ relativo a W e al valore di k opportuno oppure il peso relativo all'insieme E_k è la medesima cosa.

Di conseguenza si ottiene la seguente relazione relativa alle cardinalità:

$$|E_1 - (E_1 \cap P_1)| = |E_0 - (E_0 \cap P_0)| = Q$$

La formula di y si svilupperà come:

$$y = \frac{\sum_{\underline{w} \in W} p_{\underline{w},1}}{\sum_{\underline{w} \in W} p_{\underline{w},1} + \sum_{\underline{w} \in W} p_{\underline{w},0}} =$$

$$= \frac{\sum_{\underline{w} \in E_1 - (E_1 \cap P_1)} p_{\underline{w},1} + \sum_{\underline{w} \in E_1 \cap P_1} p_{\underline{w},1}}{\sum_{\underline{w} \in E_1 - (E_1 \cap P_1)} p_{\underline{w},1} + \sum_{\underline{w} \in E_1 \cap P_1} p_{\underline{w},1} + \sum_{\underline{w} \in E_0 - (E_0 \cap P_0)} p_{\underline{w},0} + \sum_{\underline{w} \in E_0 \cap P_0} p_{\underline{w},0}}$$

Ma essendo $p_{\underline{w},1} = 1$ e $p_{\underline{w},0} = 1$

per $\underline{w} \in E_1 - (E_1 \cap P_1)$ e $\underline{w} \in E_0 - (E_0 \cap P_0)$ si ha che $\sum_{\underline{w} \in E_1 - (E_1 \cap P_1)} p_{\underline{w},1} = |E_1 - (E_1 \cap P_1)|$ e $\sum_{\underline{w} \in E_0 - (E_0 \cap P_0)} p_{\underline{w},0} = |E_0 - (E_0 \cap P_0)|$ allora si ha che:

$$y = \frac{Q + \sum_{w \in E_1 \cap P_1} p_{w,1}}{2Q + \sum_{w \in E_1 \cap P_1} p_{w,1} + \sum_{w \in E_0 \cap P_0} p_{w,0}} \leq \frac{1}{2}$$

Quindi si ha un intervallo di valori entro il quale far variare $\Theta \epsilon \left(0, \frac{1}{2}\right)$ e di conseguenza ottenere diversi risultati in termini di precision/recall.

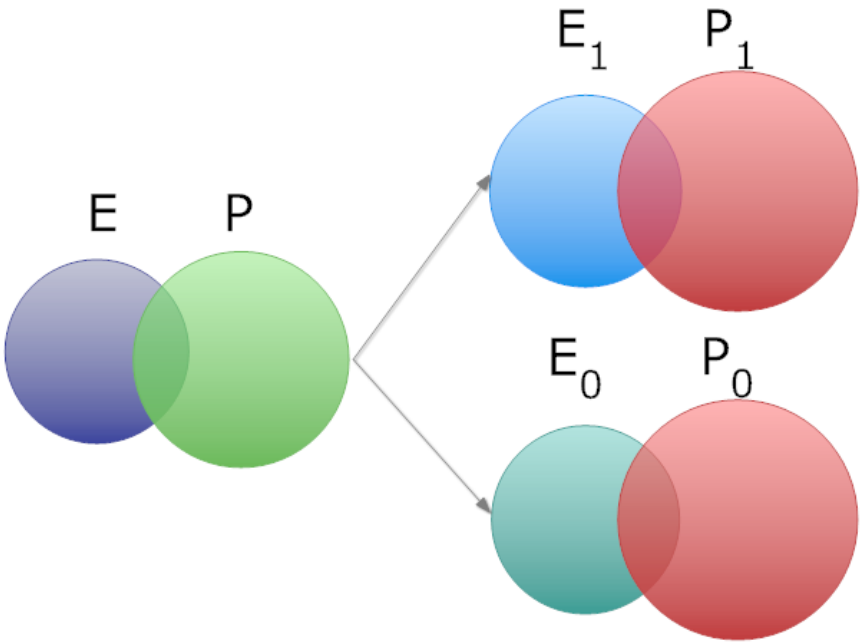

Figura 2-13 Insiemi Sleeping-Experts

In seguito vengono aggiornati i pesi delle predizioni errate secondo la penalizzazione β scelta e quindi normalizzati tra loro. Inoltre viene aggiunto al Pool l'insieme degli "esperti" relativi al testo corrente.

È da notare come l'uso delle frasi e quindi del contesto è limitato prettamente al riconoscimento del "pattern" in testi diversi in base all'appartenenza al Pool e di seguito vengono considerati e eventualmente aggiornati i pesi.

L'algoritmo è di tipo greedy ed è capace di autoapprendimento: greedy perché i vecchi testi non vengono riclassificati quando viene aggiornato il Pool e l'insieme di Pool stesso è artefice dell'autoapprendimento dell'algoritmo.

Una volta scelto il valore di permutazione r si ottengono diverse frasi nello stesso testo con alcune che possono comparire anche in altri testi. In particolare eventuali frasi poco rilevanti che compaiono in parecchi testi otterranno continue penalizzazioni portando il loro peso prossimo allo zero.

2.3.3.3 Considerazioni sui metodi contestuali

Entrambi i metodi presentano quindi delle limitazioni che li rendono poco adatti all'obiettivo finale di questo lavoro:

1. **Uso del contesto**: innanzi tutto entrambi tengono conto del contesto, questa caratteristica è insita negli algoritmi, che consiste in un insieme di termini "vicini" o che compaiono mutuamente nei testi. Nel caso di testi brevi, spesso con poche parole, è difficile quindi identificare con profitto un contesto su cui avviare le elaborazioni definite.
2. **Necessità di dati di training**: è necessario avere dei dati di training o comunque la disponibilità di regole iniziali con la quale istruire i sistemi. In entrambi i casi bisogna scansionare i testi per parametrizzare gli algoritmi.
3. **Poco estensibili**: entrambi gli algoritmi adottano una rappresentazione dei testi secondo i termini che compaiono e nient'altro. Come indicato ampiamente in letteratura questa modalità è poco efficiente in termini prestazionali e male si presta ad estensioni e affinamenti soprattutto semantici.

Come indicato in (19) il confronto di questi due metodi è stato fatto con l'algoritmo di categorizzazione Rocchio. Si sono ottenuti risultati numericamente migliori ma di ordine di grandezza confrontabile, questo indica un miglioramento non significativo in termini di precision/recall. Senza contare che computazionalmente queste elaborazioni risultano piuttosto costose in quanto necessitano del salvataggio e dell'elaborazione di un contesto.

A fronte di queste considerazioni si è preferito ignorare questi metodi come termini di confronto.

2.3.4 Metodi di Intelligenza Artificiale

Vengono qui esposti due metodi che basano la loro elaborazione su tecniche di intelligenza artificiale, in particolare il primo metodo fa uso di alberi decisionali e relativa potatura, mentre il secondo metodo si basa su tecniche di votazione.

2.3.4.1 CART

CART sta per Classification And Regression Tree Analysis.

La teoria che sotto intende questo metodo si basa su numerosi concetti: le catene di Markov, i metodi di stima Monte Carlo e le teorie probabilistiche Bayesiane. Come si può immaginare si tratta di un metodo che trova applicazione per qualsiasi problema di classificazione. Qui se ne vorrà dare un'esposizione centrata sulla classificazione sintattica di testi e in maniera semplificata, ad esempio considerando alberi binari con suddivisione vero-falso come vedremo, in modo da coglierne gli aspetti chiave per una valutazione analitica.

Il metodo qui esposto consiste nel generare un albero binario di decisione che generi una suddivisione in insiemi che rispecchi le classi del training set, si vuole quindi ottenere un albero costruito mediante regressione sui dati.

Gli elementi da definire per delineare questo metodo sono:

1. **Insieme dei testi**: consiste in un training set $D = \{d_1, \dots d_m\}$
2. **Insieme delle classi**: $C = \{c_1, \dots c_n\}$
3. **Insieme delle regole di split**: $S = \{s_1, \dots s_p\}$

   Queste regole sono usate per generare l'albero di classificazione, una regola di split nel caso di generazione di un albero binario deve fornire un risultato di tipo binario, sì/no vero/falso.

   La generazione di queste regole dipende chiaramente dal dominio, in (20) viene fatto accenno a domini numerici sia continui che discreti in maniera generale e viene anche fatto un rapido calcolo circa la cardinalità dell'insieme S. È chiaro che S avrà facilmente una cardinalità molto grossa in base al numero di variabili in gioco nella definizione del dominio, per questo in seguito verrà esposto un metodo per limitare la generazione di regole di split.

   Per quanto riguarda il dominio oggetto di questa tesi esso consiste in un insieme di testi formati da parole, un tipo di possibili regole possono essere quelle viste nell'algoritmo RIPPER paragrafo 2.3.3.1 ossia del tipo:

   $$\bigwedge_{i \in W} (w_i \in d_j) \bigwedge_{i \in W} (w_i \notin d_j) \Leftrightarrow d_j \in c_k \, , d_j \in D, c_k \in C$$

   Dove:

   w_i è il termine i presente nei testi dato che l'intero insieme di parole è W.

   d_j è il testo j dato che l'intero insieme di testi è D.

   c_k è la classe k nell'intero insieme di classi C.
4. **Nodo dell'albero**: D_i sia esso un nodo interno o una foglia consiste in un insieme di testi che è ovviamente sottoinsieme di D: $D_i \subseteq D$.

   $i \in \mathbb{N}$ rappresenta il numero del nodo, in (20) viene fornito un semplice metodo per etichettare i nodi ossia numerando il nodo figlio di sinistra come D_{2*i} e il nodo figlio di destra come D_{2*i+1}:

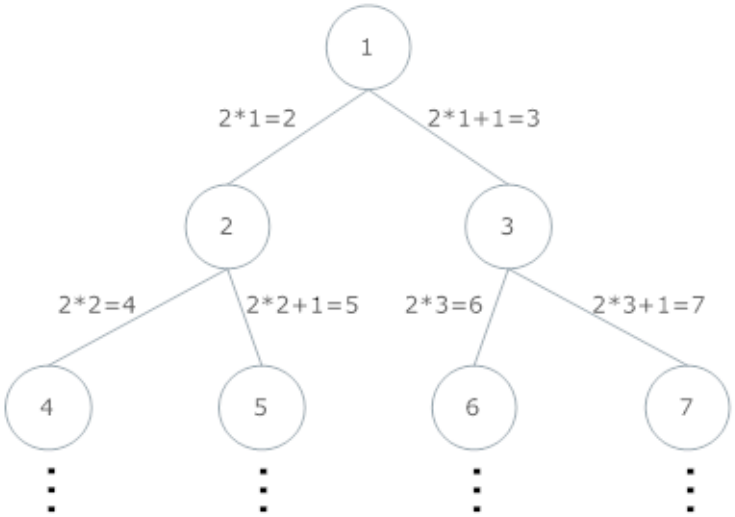

Figura 2-14 Numerazione Albero CART

5. **Funzione di diversità**: $\Delta(D_i)$ misura quanto sono eterogenei in relazione alle classi i testi appartenenti ad un dato nodo D_i. Essa è misurata con una funzione di entropia, un esempio di tale funzione può essere:

$$\Delta(D_i) = \sum_{c \in C} \left(P(c|D_i) * \ln\left(P(c|D_i)\right)\right)$$

La probabilità $P(c|D_i)$ misura la probabilità che un testo di classe c ricada nel nodo D_i, è stimabile come:

$$\hat{P}(c|D_i) = \frac{\left|d_j \in D_i \; t.c. \; classe(d_j) = c\right|}{\left|d_j \in D_i\right|}$$

Procedura regressiva di costruzione dell'albero binario

Definiti gli elementi precedenti si passa alla costruzione dell'albero binario di classificazione.

Ovviamente si parte dal nodo radice, esso conterrà tutti i testi presenti nel training set: $D_1 = D$.

L'algoritmo procede:

1. Si costruiscono quindi tutti i possibili split secondo le regole individuate, per una digressione esaustiva riguardo lo spazio composto da tutti i possibili split si consulti (20).

2. Si applicano sistematicamente tutti gli split al nodo corrente generando da un nodo padre due nodi figli, albero binario.

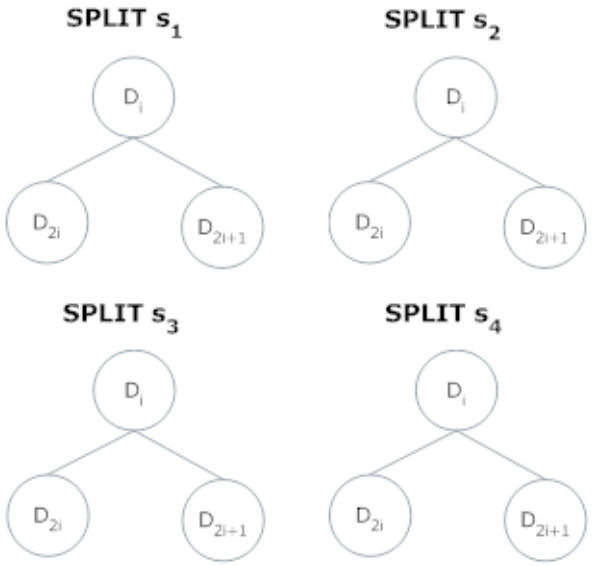

Figura 2-15 Costruzione Albero CART

Come riportato in figura sono stati generati quattro split $S = \{s_1, s_2, s_3, s_4\}$ per il nodo D_i, dei quattro scelgo lo split che:

$$s_{Di} = arg \max_{s \in S} \left(\Delta(D_i) - \left(\Delta(D_{2i}) + \Delta(D_{2i+1}) \right) \right)$$

Ossia quello che massimizza la diversità tra i testi contenuti nel nodo padre D_i e quelli contenuti nei nodi figli D_{2i} e D_{2i+1}. Questa diversità è massimizzata quando i valori di D_{2i} e D_{2i+1} sono minimi, ossia si ha un'omogeneità tra i testi selezionati all'interno di questi due nodi.

Vi è a questo punto un'ottimizzazione necessaria, come è prevedibile il numero di split applicabili ad un nodo è elevato, se non in alcuni casi infinito, in questo caso si generano, possibilmente seguendo una funzione di decadimento, gli split in successione e ci si ferma quando si ottiene un valore di diversità superiore ad una certa soglia t:

$$s_{Di} = \left\{ s \in S \; t.c. \max_{s \in S} \left(\Delta(D_i) - \left(\Delta(D_{2i}) + \Delta(D_{2i+1}) \right) \right) > t \; \wedge t \in \mathbb{R}^+ \right\}$$

3. Ci si ferma nella generazione dell'albero fissando un valore massimo di tagli generabile per ogni nodo o quando per ogni taglio si ottiene:

$$\Delta(D_i) - \left(\Delta(D_{2i}) + \Delta(D_{2i+1}) \right) < r, \forall s \in S$$

Con $r \in \mathbb{R}^+$ valore soglia.

Quando per un dato nodo è valida una di queste ipotesi tale nodo diventa un nodo foglia che chiaramente conterrà in maggioranza testi appartenenti ad una classe e testi appartenenti ad altre classi che sono stati classificati non correttamente.

Osservazioni e potatura

L'albero generato soffrirà di due tipi di singolarità:

1. Una classe può essere assegnata a più nodi foglia: questo è possibile ma non comporta alcun errore nella classificazione, significa semplicemente che una data classe è identificabile con due percorsi differenti, ossia applicando regole di split diverse.
2. Ho facilmente overfitting, l'albero generato classificherà perfettamente i dati di training. Per ovviare al problema dell'overfitting si può usare una tecnica di early-stopping esposta in (10) che consiste nella potatura dell'albero.
   Definisco quindi:
   m_{Di}: misclassified, rappresenta il numero di testi classificati di classe c in un dato nodo foglia D_i dato che non sono di classe c.
   l: leaves, numero di foglie totali.
   n: numero di testi presenti nelle foglie, notare che dal momento che è valido il punto precedente ho $n \geq |D|$.
   $E(T)$: funzione errore relativa all'albero T,

$$E(T) = \frac{\sum_{i=1}^{l} m_{Di}}{n}$$

   È calcolata come il rapporto tra la somma dei testi classificati non correttamente nei nodi foglia e il numero di testi totali presenti nei nodi foglia.
   La tecnica di early-stopping qui esposta consiste nell'eliminazione dei sottoalberi che aggiungono la minor predittività per ogni foglia.
   Se T_i è il sottoalbero i-esimo, $\alpha \in \mathbb{R}^+$ e $N(T_i)$ è il numero di foglie del sottoalbero T_i definisco la funzione:

$$E_\alpha(T_i) = E(T_i) + \alpha * N(T_i)$$

Supponendo quindi di avere due sottoalberi con rispettivamente due e tre nodi foglia posso distinguere i casi:

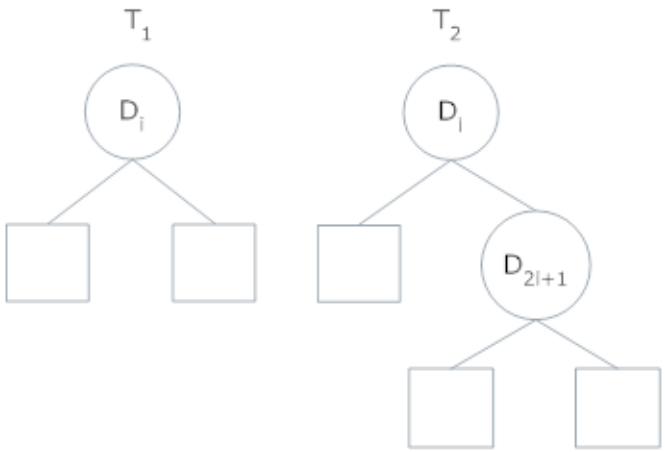

Figura 2-16 Early Stopping CART

$\alpha = 0 \Rightarrow E_\alpha(T_1) > E_\alpha(T_2)$: perché T_2 avendo più classi compie un errore di classificazione $E(T_2)$ inferiore.

$\alpha \gg \Rightarrow E_\alpha(T_1) < E_\alpha(T_2)$: perché T_2 avendo più foglie avrà più peso rispetto a T_1.

Quindi esisterà un valore di $\alpha = \alpha_*$ oltre al quale un sottoalbero ha sempre $E_\alpha(T_i)$ maggiore rispetto a tutti gli altri sottoalberi di uno stesso nodo. Questo valore rappresenterà una soglia entro cui far variare α per applicare diversi tagli sui sottoalberi che hanno $E_\alpha(T_i)$ maggiore rispetto agli altri.

Facendo variare quindi $\alpha \in (0, \alpha_*)$ genero molti alberi diversi dati dalle varie potature. Usando un test set diverso dal training set iniziale sceglierò tra tutti gli alberi quello che avrà:

$$T = arg\min_T \big(E(T)\big)$$

Concludendo, sebbene questo algoritmo presenti elevata precisione nella classificazione dei testi e una facilissima adattabilità in moltissimi ambiti esso presenta fondamentalmente due problemi: il primo è la scelta del tipo di split e la generazione che può essere un compito gravoso computazionalmente poiché vanno valutati per ogni split tutti i testi interni ad un nodo. Il secondo consiste nella potatura che necessita del mantenimento in memoria dei numerosi alberi generati e può rappresentare un grosso problema circa la complessità spaziale.

2.3.4.2 AdaBoost

I metodi di classificazione mediante votazione, in quanto metodi che sfruttano l'apprendimento, si dividono in due categorie: bagging e boosting. La differenza principale tra queste categorie è il metodo di creazione dei training set. In particolare i metodi di tipo boosting hanno una tecnica più evoluta rispetto ai primi, delucidazioni a riguardo possono essere consultate in (10).

Tra i metodi che effettuano la classificazione tramite votazione è stato scelto come riferimento AdaBoost.MH, di tipo boosting, che, come esposto in (21), offre la possibilità di fare una classificazione di un testo su più classi, fornendo una classifica. Esso è un'estensione dell'algoritmo AdaBoost.M1, esposto in (22), che si limita alla classificazione su una classe.

Qui esporremo due algoritmi: AdaBoost.M1 è il primo ad essere stato creato e contiene le basi per le variazioni seguenti, AdaBoost.MH che è invece la base di tutti gli algoritmi AdaBoost per effettuare classificazioni multi-classe.

1. **AdaBoost.M1**: questo algoritmo, come detto, presenta le caratteristiche e gli elementi essenziali per tutti gli altri. Necessita di un training set e gli elementi base sono sostanzialmente due: le ipotesi create ad ogni iterazione e una funzione di "apprendimento debole" così chiamata perché basa la creazione di una ipotesi sulla base dei valori di una distribuzione che rappresenta l'incertezza. Necessariamente quindi questa funzione non può fornire risultati precisi e ad ampio spettro su tutte le classi ed è per questo chiamata "debole". A valle di tale funzione c'è poi la valutazione di concerto di tutte le ipotesi create e la scelta della classe più adatta tramite un meccanismo di votazione. I classificatori creati quindi saranno più o meno esperti su una data classe e le varie valutazioni verranno poi combinate tra loro.
   L'algoritmo pertanto necessita come detto di un training set:
   $$Tset = (x_1, y_1), \dots (x_m, y_m)$$
   Con $x_i \in X, |X| = m$ insieme dei testi, $y_i \in Y, |Y| = n$ insieme delle possibili classi.

Di una funzione di "apprendimento debole" che genera una ipotesi di assegnamento globale su tutti i testi data una distribuzione di incertezza D:

$$H = WeakLearn(D)$$

Dove H è un'ipotesi di assegnamento cioè dato un testo x ne assegna una classe y:

$$H: X \to Y$$

$$H(x) = y$$

essa non consiste in un elenco di coppie testo-classe bensì di una funzione che, sfruttando le caratteristiche di un dato testo x, genera un'ipotesi circa l'appartenenza di x ad una classe y.

Infine viene fornito un numero di iterazioni $T \in \mathbb{N} - \{0\}$.

Function AdaBoost.M1(Tset, WeakLearn, T)
{
for i = 1...$|Tset|$
$D_{1,i} \coloneqq \frac{1}{|Tset|}$
for t = 1...T
{
$H_t \coloneqq WeakLearn(D_t)$
$E_t \coloneqq \sum_{i:H_t(x_i)\neq y_i} D_{t,i}$
if $E_t > \frac{1}{2}$
{
T:=t-1
break;
}
$\beta_t \coloneqq \frac{E_t}{1-E_t}$
$Z_t \coloneqq 2 * E_t$
for i = 1...m
if$(H_t(x_i) \neq y_i)$ $D_{t+1,i} \coloneqq \frac{D_{t.i}}{Z_t} * \beta_t$
else $D_{t+1,i} \coloneqq \frac{D_{t,i}}{Z_t}$
}
return $\underline{H}$,$\underline{\beta}$
}
Function assignclass(x,$\underline{H}$,$\underline{\beta}$)
{

```
        for y = 1...n
                sumclass_y = ∑_{t:H_t(x)=y} 1/β_t
        return arg max_{y∈Y}(sumclass)
}
```

Bisogna quindi fare alcune precisazioni circa i fattori di questo algoritmo:

E_t è l'errore associato alle ipotesi errate fatte dal classificatore generato all'epoca t, H_t. Esso è calcolato come la somma dei valori di incertezza dei testi nel training set non classificati correttamente. Conservativamente quindi si pone un limite di $\frac{1}{2}$ al valore massimo dell'errore, questo ci consente di far decadere l'errore dell'ipotesi di assegnamento nella funzione assignclass a zero con velocità esponenziale, così come esposto nel teorema 1 in (22).

D_t è la distribuzione relativa all'incertezza al passo t essa è una distribuzione rispetto ai testi da classificare. Viene quindi aggiornata ad ogni passo, costruendo la distribuzione per il passo successivo da cui verrà calcolata la nuova ipotesi di assegnamento.

β_t è una rielaborazione "comoda" dell'errore, la sua definizione è strettamente legata con Z_t che è uno scalare utile a normalizzare in modo da mantenere la distribuzione D_{t+1} valida: si parte innanzi tutto da un modello leggermente diverso, sapendo che $\beta_t < 1$ si suppone di ridurre con rinforzo negativo i valori della distribuzione $D_{t,i}$ associati ai testi x_i che sono stati classificati correttamente dall'ipotesi corrente H_t, e aumentare con rinforzo positivo i valori della distribuzione $D_{t,i}$ associati ai testi classificati non correttamente:

$$D_{t+1,i} = \beta_t * D_{t,i}, i: H_t(x_i) = y_i$$

$$D_{t+1,i} = \frac{D_{t,i}}{\beta_t}, i: H_t(x_i) \neq y_i$$

Per far sì che D_{t+1} sia anch'essa una distribuzione bisogna che sia verificata l'equazione:

$$\sum_{i:H_t(x_i)=y_i} D_{t+1} + \sum_{i:H_t(x_i)\neq y_i} D_{t+1} = 1 \Rightarrow$$

$$\Rightarrow \beta_t * \sum_{i:H_t(x_i)=y_i} D_t + \frac{1}{\beta_t} * \sum_{i:H_t(x_i)\neq y_i} D_t = 1$$

Ma

$$E_t = \sum_{i:H_t(x_i)\neq y_i} D_t$$

E

$$1 - E_t = \sum_{i:H_t(x_i)=y_i} D_t$$

Da cui

$$\beta_t * (1 - E_t) + \frac{E_t}{\beta_t} = 1 \Rightarrow \beta_t = \frac{E_t}{1 - E_t} \vee \beta_t = 1$$

Ignorando la soluzione banale e invariante $\beta_t = 1$, si sceglie di aggiornare:

$$\beta_t = \frac{E_t}{1 - E_t}$$

Da questo punto in poi si modifica un po' il modello fornendo solo rinforzo negativo per le ipotesi con predizione corretta:

$$D_{t+1,i} = \beta_t * \frac{D_{t,i}}{Z_t}, i: H_t(x_i) = y_i$$

$$D_{t+1,i} = \frac{D_{t,i}}{Z_t}, i: H_t(x_i) \neq y_i$$

questo però comporta l'introduzione di una costante di normalizzazione Z_t per far sì che la nuova distribuzione D_{t+1} sia definibile tale:

$$\sum_{i:H_t(x_i)=y_i} D_{t+1} + \sum_{i:H_t(x_i)\neq y_i} D_{t+1} = 1 \Rightarrow$$

$$\Rightarrow \frac{\beta_t}{Z_t} * \sum_{i:H_t(x_i)=y_i} D_{t,i} + \frac{1}{Z_t} * \sum_{i:H_t(x_i)\neq y_i} D_{t,i} = 1 \Rightarrow$$

$$\Rightarrow Z_t = \beta_t * \sum_{i:H_t(x_i)=y_i} D_{t,i} + \sum_{i:H_t(x_i)\neq y_i} D_{t,i} \Rightarrow$$

$$\Rightarrow Z_t = \beta_t * (1 - E_t) + E_t \Rightarrow$$

$$\Rightarrow Z_t = 2 * E_t$$

Il valore di β_t rappresenta una sorta di non-attendibilità e difatti la funzione assign-class userà il reciproco di tale valore per selezionare la classe più adatta ad un testo x in ingresso.

Purtroppo AdaBoost.M1 presenta alcune problematiche, su tutte vi è il fatto di partire iterando da una distribuzione uniforme il che implica che la prima

ipotesi costruita sarà necessariamente casuale. Inoltre dal teorema 1 di (22), come accennato, l'algoritmo non può gestire con precisione ipotesi con errore $E_t \geq \frac{1}{2}$. Questi e altri limiti sono superati dall'algoritmo AdaBoost.M2 che però qui non vedremo, piuttosto valuteremo l'algoritmo multi-classe AdaBoost.MH.

2. **AdaBoost.MH**: come esposto questo algoritmo permette di classificare su più classi in contemporanea, assegnando più classi di appartenenza ad un testo. L'estensione di questo algoritmo è AdaBoost.MR dove la R sta per "ranking" ossia fornisce una classifica ordinata in base ad un indice di appartenenza alle varie classi e supera inoltre alcuni limiti di AdaBoost.MH.
   Rispetto alla base concettuale fornita da AdaBoost.M1 bisogna fare alcune modifiche, fermi restando sulle stesse ipotesi e nomenclatura.
   La funzione ipotesi di predizione H sarà del tipo:

$$H: X \times Y \to \mathbb{R}$$

   Questa funzione fornisce un valore di confidenza circa l'appartenenza di un testo x ad una classe y. In questo modo si mantiene un insieme di pesi che descrivono l'appartenenza.
   In particolare:

$$-1 \leq H(x, y) \leq 1$$

   Dove:
   $H(x, y) = -1$ esprime il fatto che secondo l'ipotesi H il testo x non appartiene sicuramente alla classe y.
   $H(x, y) = 1$ esprime invece la sicurezza totale che il testo x sia di classe y.
   I valori intermedi valutano il grado di sicurezza dell'ipotesi H circa la predizione.
   Un'altra distinzione da fare è circa la reale appartenenza di un testo ad una data classe. Ogni testo x_i ha associato un insieme di classi a cui appartiene Y_i, formalmente si introduce una funzione binaria che esprime la reale appartenenza di un testo x ad una classe y:

$$R: X \times Y \to \{-1,1\}$$

$$y_j \notin Y_i \Leftrightarrow R(x_i, y_j) = -1$$

$$y_j \in Y_i \Leftrightarrow R(x_i, y_j) = 1$$

La distribuzione D sarà quindi definita sul prodotto cartesiano tra gli insiemi X e Y. In partenza, analogamente a AdaBoost.M1, sarà una uniforme e pertanto il suo valore sarà:

$$D_1(x * y) = \frac{1}{m * n}, \forall x \in X, \forall y \in Y$$

Function AdaBoost.MH(Tset, WeakLearn, T)
{
$D_1(x * y) = \frac{1}{m*n}, \forall x \in X, \forall y \in Y$
for t = 1...T
{
$H_t := WeakLearn(D_t)$
$D_{t+1}(x, y) := \frac{D_t(x,y) * e^{-\alpha_t * R(x,y) * H_t(x,y)}}{Z_t}, \forall x \in X, \forall y \in Y$
}
return $\underline{H}$,$\underline{\alpha}$
}
Function assignclass(x,y,$\underline{H}$,$\underline{\alpha}$)
{
return $\sum_{t=1}^{T}(\alpha_t * H_t(x, y))$
}

Riguardo a questo algoritmo bisogna fare alcune osservazioni: il valore di $\alpha_t > 0$ serve ad aggiornare D_t in modo da incrementare il peso delle coppie (x, y) che sono state classificate non correttamente da H_t cioè quando:

$$H_t(x, y) * R(x, y) = -H_t(x, y)$$

Ossia $H_t(x, y)$ differisce in segno da $R(x, y)$ fornendo una predizione in controtendenza rispetto alla realtà descritta nel training set.

Il calcolo di α_t consegue alle considerazioni sull'Hamming Loss (HL). Un parametro che è sintatticamente invisibile nell'algoritmo ma che concorre nella scelta di α_t e Z_t . Empiricamente vi è un limite superiore al valore di Hamming Loss dell'algoritmo:

$$HL < \prod_{t=1}^{T} Z_t$$

Questo limite superiore e α_t sono scelti in modo da minimizzare il valore di Z_t, costante di normalizzazione:

$$Z_t = \sum_{x \in X} \left[\sum_{y \in Y} \left(D_t(x,y) * e^{\alpha_t * R(x,y) * H_t(x,y)} \right) \right]$$

La dimostrazione di questa formula è analoga a quella fatta in AdaBoost.M1.

L'aspetto fino ad ora "misterioso" di questi algoritmi è rappresentato dalle operazioni effettuate dalle funzioni di "apprendimento debole", una chiara esposizione di alcune di esse è esposta in (21).

Il problema degli algoritmi AdaBoost è che hanno una fase di training dispendiosa come esposto in (21) a fronte di risultati in termini qualitativi solo leggermente migliori secondo le metriche esposte nella stessa fonte.

In accordo con il concetto di boosting questi algoritmi permettono di concentrare le risorse e le computazioni effettuate sui testi e sulle classi per cui è difficile fare predizioni veritiere.

2.4 Tabella riassuntiva delle caratteristiche

Viene qui esposta una tabella che elenca delle caratteristiche di interesse per il lavoro svolto, si cerca un algoritmo che possegga tali caratteristiche o la maggior parte di esse:

METODI	Contesto	Avoid overfitting	Bassa complessità	Non necessita di dati di training o test	Valutazione istantanea di più classi	Inserimento incrementale di testi
Rocchio		√	√		√	√
k-NN		√	√	√	√	√
Naive-Bayes		√	√		√	√
SVM		√				√
LSI		√	√	√	√	
RIPPER	√	√				√
Sleeping-Experts	√	√			√	√
CART					√	√
AdaBoost.MH					√	√

Tabella 1 Caratteristiche algoritmi di classificazione

1. **Contesto**: specifica se l'algoritmo è in grado di considerare dei dati relativi al contesto durante la classificazione. Il contesto è rappresentato fondamentalmente da due informazioni: dati relativi alle classi in base alla quale classificare e dati relativi agli altri testi nel momento in cui se ne classifica un secondo.
2. **Avoid overfitting, generalization**: dal momento che l'algoritmo necessita di dati di training e test esso è esposto al problema dell'overfitting. Questa caratteristica specifica se l'algoritmo ha funzionalità in grado di limitare l'overfitting.
3. **Bassa complessità**: questa caratteristica indica il fatto che l'algoritmo opera su dati adatti ad ottenere buone prestazioni in particolare la sua complessità è buona (logaritmica o polinomiale).

4. **Non necessita di dati di training e test**: questa proprietà è auto-esplicativa. In particolare non essendoci tali dati non vi saranno nemmeno le fasi di training e test ma si è analiticamente confidenti del buon risultato ottenuto. Tuttavia è sempre necessario effettuare dei test per avere un risultato empirico circa la bontà.
5. **Valutazione contemporanea di più classi**: l'algoritmo che gode di questa proprietà è in grado di assegnare il testo ad una classe piuttosto che ad un'altra in un'unica passata e non valutando l'appartenenza una classe per volta.
6. **Inserimento incrementale di testi**: tramite questa proprietà gli algoritmi sono in grado di classificare nuovi testi in arrivo in un secondo momento non operando quindi in modalità "batch". L'algoritmo poi può fare autoapprendimento se intrinsecamente non soffre di overfitting oppure limitarsi alla sola classificazione del nuovo testo.

Le tecniche esposte in questo corposo riassunto possono essere uno spunto per lavori futuri in questo ambito, le estensioni sono numerose e nulla vieta di adottare alcuni metodi qui esposti per:

1. Raffinare risultati ottenuti.
2. Implementare tecniche di riduzione delle dimensioni.
3. Velocizzare la computazione tramite l'integrazione di più tecniche.

Capitolo 3 **Modelli di Analisi**

Questo lavoro si pone come obiettivo il confronto di un algoritmo e una sua estensione, presentati nel resto di questo capitolo, con due algoritmi già presenti in letteratura che coprano la maggior parte delle funzionalità ritenute “interessanti”, si veda il paragrafo 2.4.

Verranno quindi esposte le altre tecniche implementate con un accenno teorico e informazioni circa la loro realizzazione pratica e le dovute motivazioni.

3.1 Obiettivi

Gli obiettivi di questo lavoro non consistono solamente in un semplice miglioramento in termini di precision/recall o altre metriche già ottimizzate dagli algoritmi presenti, piuttosto si vuole ottenere una precisione confrontabile a quella raggiunta anche nella classificazione di testi brevi e un miglioramento prestazionale significativo in quanto l'attività di crawling fornisce centinaia di migliaia di testi alla volta e l'attività di categorizzazione deve poter avvenire in tempo reale.

In particolare si vuole ottenere un algoritmo che non necessiti di dati di training e test, ma eventualmente che si basi sulla supervisione dell'utente almeno in una fase iniziale e eventuali "metadata" relativi al contesto. Per questo gli algoritmi di paragone scelti sono k-NN e LSI.

Il primo, k-NN, è quello che presenta lo "spettro" più ampio relativamente alle caratteristiche, si veda la Tabella 1. Inoltre come riportato in (10), dove viene fornito un confronto esaustivo su parecchi metodi, k-NN presenta le stesse prestazioni di algoritmi più complicati nonostante il suo approccio molto semplice e diretto.

Il secondo, LSI, perché ricopre le caratteristiche fondamentali: non soffre di overfitting, è centrato su una decomposizione che promette prestazioni elevatissime, non necessita di dati di training o test e consente di valutare più classi contemporaneamente. In particolare dal momento che l'attività in tempo reale è una caratteristica molto importante di questo lavoro è molto facile capire perché metodi di Intelligenza Artificiale o contestuali non siano molto indicati.

Rispetto alle riduzioni individuate in letteratura si vuole ottenere un algoritmo che non presenti variazioni significative nelle sue prestazioni e nella qualità della classificazione rispetto alle riduzioni applicate.

3.2 Assunzioni preliminari

Si è scelto di ignorare i formati internazionali e i social metadata in quanto non vi è nessuna evidente informazione utile all'analisi in essi, non rappresentando nulla di rilevante relativamente ad un dominio specifico. Non si è quindi presa in considerazione la struttura e le informazioni semantiche da questi addotte soprattutto per via dell'ambito sintattico in cui si vuole operare.

Lavori successivi possono stimare il peso dei contenuti semantici associati a questi elementi e in particolare quanto influisce la sorgente di provenienza dei testi rispetto all'analisi sintattica introducendo nella logica del software la loro concettualizzazione.

3.3 Schema concettuale di riferimento

Lo schema concettuale di riferimento considera le entità protagoniste nell'ambito in cui questo lavoro è collocato, esse sono: testi, termini e classi. In questo paragrafo si vuole fornire una enunciazione delle relazioni che possono intercorrere tra queste entità.

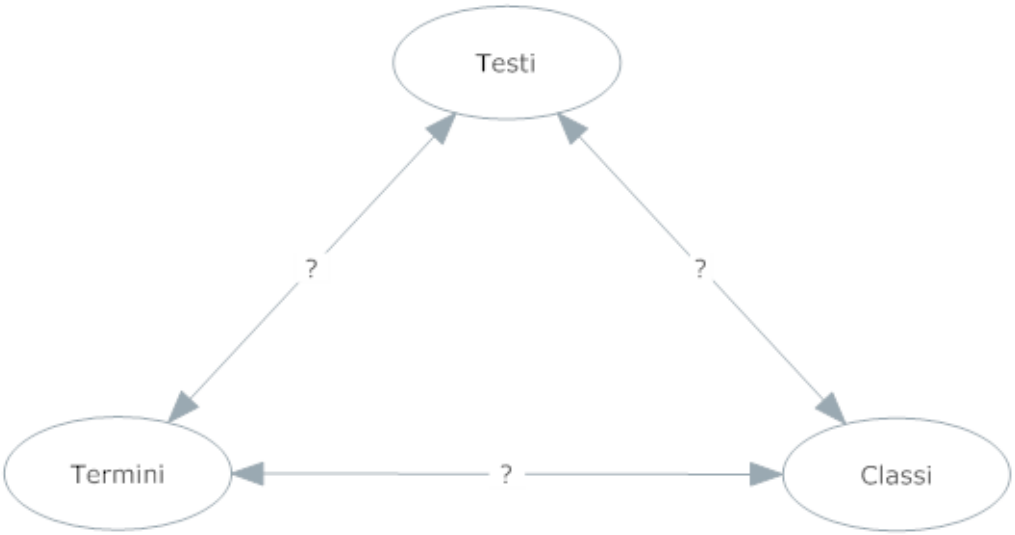

Figura 3-1 Relazioni Incognite tra Termini-Testi-Classi

1. **Relazione tra testi e termini**: essa viene espressa dalla matrice di rilevanza dove vengono calcolati dei valori che rappresentano l'importanza di un termine all'interno del testo. Tale calcolo si basa sul numero di comparse di un termine nei testi del dataset e sul numero di comparse nel singolo testo.

2. **Relazione tra testi e classi**: la costruzione di questa relazione è quanto gli algoritmi di classificazione si propongono di fare. A livello teorico la relazione preferita come riferimento è quella funzionale, quindi ad ogni testo sia associata al massimo una classe e possibilmente sia di tipo suriettivo ossia non vi siano classi a cui non sono assegnati testi, pertanto si avrà che:

   $$c \leq n$$

   Con c numero di classi e n numero di testi classificati.

   Anche se è doveroso ribadire come tale tipo di relazione può essere di vario tipo in base alle esigenze qualitative e algoritmiche.
3. **Relazione tra termini e classi**: questa relazione è definibile in due modi, tramite metadata forniti preventivamente circa i termini che identificano una classe oppure dopo la classificazione tramite i termini che appartengono a testi classificati.

3.4 Dati a disposizione

I dati a disposizione possono variare, la componente opzionale di maggiore importanza sono i dati relativi alle classi. Si possono avere quindi delle parole chiave, associate ad ogni classe, relative al dominio considerato che eventualmente possono essere espanse in numero mediante l'uso di statistiche o algoritmi di autoapprendimento. In presenza di tali dati la classificazione avviene senza affinamenti iterativi che identificano classi di divisione secondo tecniche di TC, vedi (10).

Di grande importanza sono i dati su cui avviare l'elaborazione, o Raw Data, come indicato in Figura 3-2 essi sono caratterizzati oltre che dal tipo di file in ingresso "csv" o "xml" anche dalla codifica dei caratteri e dalla lingua utilizzata per comporre il messaggio.

3.5 Schema generale dell'analisi

Le elaborazioni effettuate in questo lavoro consistono principalmente in clusterizzazioni, con un occhio di riguardo alle prestazioni, di testi relativamente brevi e informali. Per svolgere queste attività vengono presi in considerazione vari accorgimenti, che possano migliorare le prestazioni, applicati in ordine sequenziale e opzionale in modo da poter ottenere anche un confronto tra le tecniche applicate.

Il primo passo è estrarre gli alberi sintattici relativi ai testi da analizzare tramite un parser LL(1). Si effettua in seguito una prima fase di riduzione delle dimensioni concentrandosi nell'eliminazione di alcuni termini presenti nei testi in ingresso e nell'estrazione delle radici dei termini. Si passa successivamente alla fase di indicizzazione per ottenere la matrice di rilevanza che associa parole a testi con il relativo valore, ovviamente usando gli accorgimenti necessari a salvare matrici sparse. In seguito vi è una seconda fase di riduzione delle dimensioni che rimuove i valori relativi a parole poco significative e successivamente rielabora i dati a disposizione adottando una rappresentazione maggiormente ottimizzata.

É molto importante adottare tecniche di riduzione della complessità spaziale per due motivi: ridurre le risorse necessarie a rappresentare i dati e indirettamente ridurre le necessità computazionali per trattare tali dati, se vogliamo indirettamente si riduce anche la complessità temporale.

In seguito è riportato lo schema a blocchi concettuale dell'analisi con gli elementi più significativi che lo compongono:

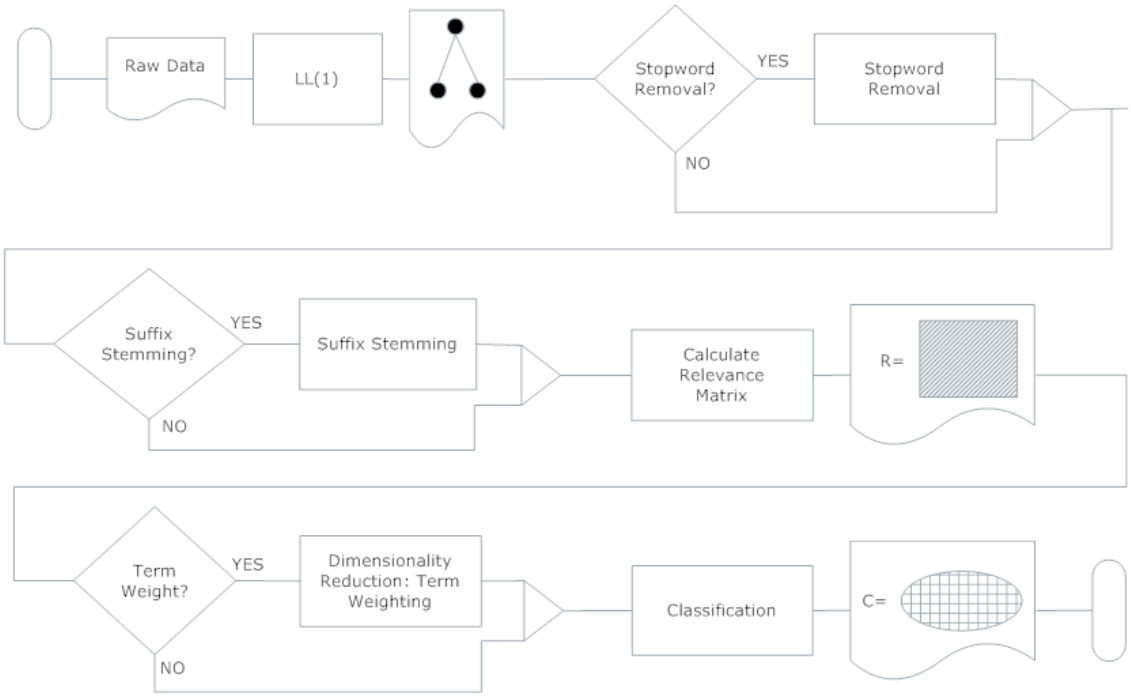

Figura 3-2 Diagramma di Flusso

Il resto del capitolo sarà dedicato alla spiegazione dei vari passaggi con i riferimenti teorici e le assunzioni adottate.

3.6 Parsing

É stato generato quindi tramite l'uso di JavaCC™ (23) un semplice parser LL(1), optando quindi per la massima velocità di analisi del testo in ingresso essendo sostanzialmente impossibile analizzare il linguaggio naturale mediante un formalismo matematico si è scritta una semplice grammatica in grado di identificare, seppur approssimativamente, parole, cifre, simboli e valutazioni monetarie.

Essa, anche se parzialmente data la complessità di un messaggio scritto in linguaggio naturale, è in grado anche di identificare le frasi che compongono il messaggio.

É di seguito riportata la grammatica nella formulazione secondo Chomsky estesa con espressioni regolari partendo dal non terminale <tokenizetext> che è stata implementata:

<tokenizetext> → <periods> (EOF)?
<periods> → <periodwithinitialterminals>* <period>+
<periodwithinitialterminals> → TERMINATORS
<period> → (CIPHER | SYMBOLS | VALUE | WORD | OTHER)+(TERMINATORS | EOF)

L'albero sintattico prodotto quindi conterrà una grammatica con attributi in cui l'unico attributo considerato è il numero della frase all'interno del messaggio. Vi sono pertanto delle problematiche da non trascurare riguardo l'albero sintattico prodotto: la già esposta ambiguità del testo sorgente, la scarsa potenza del parser LL(1) usato nel classificare e riconoscere i tokens e l'interpretazione di caratteri fondamentali quali ad esempio i terminatori.

Di seguito è fornito un esempio relativo al testo "*Eat the cake. Kitchen the chickens. Take a look to childrens.*":

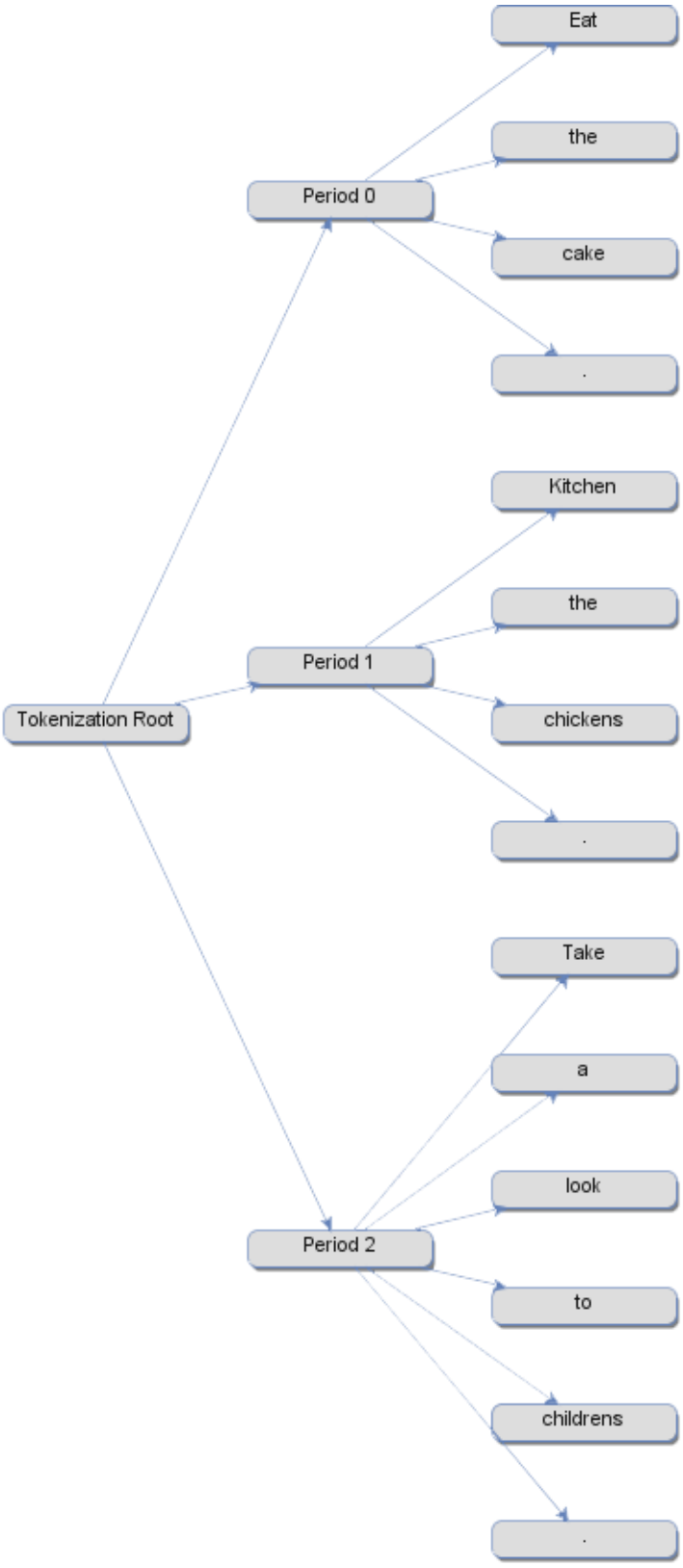

Figura 3-3 Albero Sintattico di Esempio

Come si può vedere ad ogni frase è associato un numero incrementale che rappresenta l'unico attributo considerato.

3.7 Riduzione delle dimensioni dello spazio di analisi

Dopo aver effettuato il parsing dei testi è possibile, opzionalmente, ridurre le dimensioni dello spazio di analisi tramite operazioni di stemming e stop-word removal. Queste attività riducono una dimensione della matrice di rilevanza migliorando in prima approssimazione la velocità di calcolo.

Come esposto in (8) la riduzione delle dimensioni dello spazio di analisi può comportare anche un effettivo miglioramento della qualità della classificazione dei testi riducendo in primis l'effetto dell'overfitting. Anche se questo non è vero in assoluto in quanto spesso dipende dal grado di riduzione stesso e dall'algoritmo di classificazione scelto.

Preferibilmente viene effettuata prima la rimozione delle stop-word e in seguito il troncamento dei suffissi per evitare che una parola troncata ricada in una stop-word e quindi venga eliminata quando invece rappresenta un contenuto sintatticamente rilevante.

1. **Stop-word Removal**: L'elenco delle stop-word in inglese da riconoscere è stato preso da (24).

   Il procedimento è immediato: presa ogni stringa facente parte del testo se questa è contenuta nell'elenco delle stop-word allora viene rimossa dall'albero.
2. **Suffix stemming**: per effettuare la rimozione dei suffissi nelle parole collassando parole con uguale significato nella stessa struttura sintattica si è usato il consolidato Suffix Stemmer di Porter (25).

   Come riportato nell'articolo la prima implementazione consisteva in 400 linee di codice BCPL e i test fatti all'epoca su un IBM 370/165 descritto in (26) hanno consentito di processare un vocabolario di 10000 differenti parole in 8.1 secondi. Questo è una garanzia per quanto riguarda la velocità computazionale, aspetto centrale in questo lavoro di tesi.

   È chiaro che il rischio è di far collassare due parole dalla sintassi simile ma dal significato diverso in un'unica parola nello spazio di analisi. A riguardo è possibile fare delle valutazioni immediate in quanto, come esposto nell'articolo di Porter tali casi sono estremamente rari e inoltre per quanto

riguarda questo lavoro di tesi bisogna considerare che nell'elaborazione dei testi non viene considerato il valore semantico delle parole, quindi eventuali trasformazioni non volute verranno "affossate" tanto più sono rare e tanto più è grande lo spazio di analisi.

L'algoritmo si basa su un riconoscitore LL(1) e un traduttore a regole non in linea. Vengono quindi definite vocali e consonanti, in (25):"*sono vocali le lettere 'A' 'E' 'I' 'O' 'U' e le 'Y' precedute da consonante*". Quindi chiamando le vocali 'v' e le consonanti 'c' si può tradurre una qualsiasi parola con l'espressione regolare:

$$c^*(v^+c^+)^m v^*, m \geq 0$$

Il valore di m assume particolare importanza in quanto conta il numero di sillabe contenute nella parola.

Si definiscono quindi delle regole di traduzione dove presa una parola se essa verifica le condizioni logiche della parte antecedente allora viene trasformata la parte finale della stessa con quanto riportato nella parte conseguente della regola, in termini generali:

$$(condizione\ su\ m) \wedge (condizioni\ su\ espressione\ regolare) \wedge (finale = \text{STRINGA}) \rightarrow finale \coloneqq STRINGA$$

Queste regole sono definite e applicate in 5 passi distinti su ogni parola, le regole di traduzione sono:

Step 1a:

*SSES → *SS

*IES → *I

*SS → *SS

*S → *ε

ε: stringa vuota

Step 1b:

(m>0) ∧ *EED → *EE , quindi procedi con lo Step 1c

(*v*) ∧ *ED → *ε , quindi procedi con lo Step 1b alternativo

(*v*) ∧ *ING → *ε , quindi procedi con lo Step 1b alternativo

Step 1b alternativo:

*AT → *ATE

$*BL \rightarrow *BLE$

$*IZ \rightarrow *IZE$

$(*vv) \wedge (\sim(*L \vee *S \vee *Z))) \rightarrow *v$

$(m=1) \wedge (*cv(c\text{-}\{W,X,Y\})) \rightarrow *E$

Step 1c:

$(*v*) \wedge *Y \rightarrow *I$

Lo step 1 opera sui plurali e i participi passati della lingua inglese.

Step 2:

$(m>0) \wedge *ATIONAL \rightarrow *ATE$

$(m>0) \wedge *TIONAL \rightarrow *TION$

$(m>0) \wedge *ENCI \rightarrow *ENCE$

$(m>0) \wedge *ANCI \rightarrow *ANCE$

$(m>0) \wedge *IZER \rightarrow *IZE$

$(m>0) \wedge *ABLI \rightarrow *ABLE$

$(m>0) \wedge *ALLI \rightarrow *AL$

$(m>0) \wedge *ENTLI \rightarrow *ENT$

$(m>0) \wedge *ELI \rightarrow *E$

$(m>0) \wedge *OUSLI \rightarrow *OUS$

$(m>0) \wedge *IZATION \rightarrow *IZE$

$(m>0) \wedge *ATION \rightarrow *ATE$

$(m>0) \wedge *ATOR \rightarrow *ATE$

$(m>0) \wedge *ALISM \rightarrow *AL$

$(m>0) \wedge *IVENESS \rightarrow *IVE$

$(m>0) \wedge *FULNESS \rightarrow *FUL$

$(m>0) \wedge *OUSNESS \rightarrow *OUS$

$(m>0) \wedge *ALITI \rightarrow *AL$

$(m>0) \wedge *IVITI \rightarrow *IVE$

$(m>0) \wedge *BILITI \rightarrow *BLE$

Un’ottimizzazione che qui vediamo chiaramente è stata quella di porre i suffissi da elaborare nell’antecedente in ordine alfabetico rispetto al penultimo carattere, in questo modo l’algoritmo può accedere velocemente alla regola corretta verificando tale carattere.

Step 3:

$(m>0) \wedge *ICATE \rightarrow *IC$

$(m>0) \wedge *ATIVE \rightarrow *\varepsilon$

$(m>0) \wedge *ALIZE \rightarrow *AL$

$(m>0) \wedge *ICITI \rightarrow *IC$

$(m>0) \wedge *ICAL \rightarrow *IC$

$(m>0) \wedge *FUL \rightarrow *\varepsilon$

$(m>0) \wedge *NESS \rightarrow *\varepsilon$

Step 4:

$(m>1) \wedge *AL \rightarrow *\varepsilon$

$(m>1) \wedge *ANCE \rightarrow *\varepsilon$

$(m>1) \wedge *ENCE \rightarrow *\varepsilon$

$(m>1) \wedge *ER \rightarrow *\varepsilon$

$(m>1) \wedge *IC \rightarrow *\varepsilon$

$(m>1) \wedge *ABLE \rightarrow *\varepsilon$

$(m>1) \wedge *IBLE \rightarrow *\varepsilon$

$(m>1) \wedge *ANT \rightarrow *\varepsilon$

$(m>1) \wedge *EMENT \rightarrow *\varepsilon$

$(m>1) \wedge *MENT \rightarrow *\varepsilon$

$(m>1) \wedge *ENT \rightarrow *\varepsilon$

$(m>1) \wedge (*S \vee *T)ION \rightarrow *\varepsilon$

$(m>1) \wedge *OU \rightarrow *\varepsilon$

$(m>1) \wedge *ISM \rightarrow *\varepsilon$

$(m>1) \wedge *ATE \rightarrow *\varepsilon$

$(m>1) \wedge *ITI \rightarrow *\varepsilon$

$(m>1) \wedge *OUS \rightarrow *\varepsilon$

$(m>1) \wedge *IVE \rightarrow *\varepsilon$

$(m>1) \wedge *IZE \rightarrow *\varepsilon$

Step 5a:

$(m>1) \wedge *E \rightarrow \varepsilon$

$(m=1 \wedge (\sim(*cv(c\text{-}\{W,X,Y\})))E \rightarrow \varepsilon$

Step 5b:

$(m>1 \wedge *vv \wedge *L) \rightarrow *v$

Come indicato nell'articolo (25) questa procedura riduce il dizionario di parole e quindi di conseguenza lo spazio di analisi di più di un terzo.

In particolare è riportato il risultato che su 10000 parole 3630 vengono ridotte ottenendo uno spazio di analisi di 6370 distinte parole.

3.7.1 Rappresentazioni frasali

Un sottoparagrafo a parte è stato riservato alle tecniche di rappresentazione che sfruttano entità più complesse dei singoli termini riuscendo eventualmente a ridurre ulteriormente le dimensioni.

In letteratura sono stati usati vari approcci, in (27) viene proposto un approccio "grammaticale" i testi sono scansionati in base ad una grammatica che ne coglie aspetti semantici, questa modalità presenta però buoni risultati solo attraverso l'appoggio di una clusterizzazione basata sui termini e inoltre si ottengono risultati spiccatamente migliori solo nel caso in cui si hanno fonti estese e numerose. Questo è poco in linea con l'obiettivo di questo lavoro che ha come fine l'analisi di testi brevi e poco strutturati provenienti da social network.

L'estensione del lavoro esposto in (27), ossia (28), approfondisce le motivazioni per cui l'uso di frasi sintattiche è poco efficiente nella classificazione di testi. Questo perché le frasi sono in accordo con la grammatica della lingua con cui sono scritte e questo, con riferimento al presente lavoro, è quantomeno deleterio quando si valutano testi provenienti da social network dove la correttezza grammaticale è spesso mancante.

Infine bisogna considerare il fatto che l'uso di frasi anziché termini seppur semanticamente più rilevante risulta statisticamente meno indicativo facendo venir meno eventuali tecniche di data mining e classificazione statistica in questo e in altri lavori futuri a riguardo.

3.8 Calcolo della rilevanza

Tutte le informazioni relative a questa fase sono riassunte in (10).

Concettualmente la matrice che rappresenta le coppie parola-testo, a fronte ovviamente dell'ottimizzazione relativa alle matrici sparse, è come quella riportata in Figura 3-4:

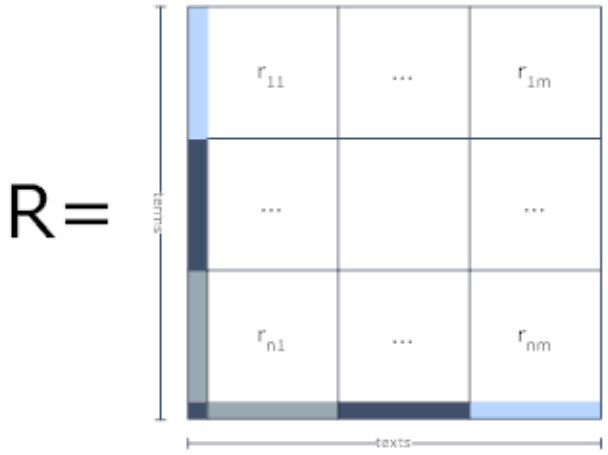

Figura 3-4 Matrice di Rilevanza

Usando una notazione matematica la matrice è: $R \in \mathbb{M}_{\mathbb{R}}^{n \times m}$.

Che contiene i valori di rilevanza: $r_{ij}, 1 \leq i \leq n, 1 \leq j \leq m$, n parole, m testi.

Le fonti consultate in letteratura identificano sei tecniche per calcolare il valore di rilevanza r_{ij}:

1. **Boolean weighting**: non consiste in una effettiva pesatura, si limita ad assegnare i valori uno o zero in base alla presenza del termine nel testo.
2. **Frequenza**: anche questa è una tecnica molto immediata e rudimentale, essa si limita ad assegnare come peso il numero di volte che compare la parola nel testo.
3. **TFxIDF weighting**: acronimo di: Term Frequency Inverse Document Frequency, questa formula si basa sulla frequenza con cui un termine compare in un testo e dal numero di testi che contengono la parola, da qui il riferimento all'indice inverso:

$$r_{ij} = f_{ij} * \ln\left(\frac{m}{m_i}\right)$$

Dove f_{ij} è la frequenza del singolo termine nel testo, m è il numero di testi e m_i è il numero di testi contenenti il termine i.

Ragionando sulla formula si ottiene che, essendo $m \geq m_i$, ho che $\ln\left(\frac{m}{m_i}\right) \geq 1$ ed è tanto più grande quanto la parola è specifica per pochi testi, e quindi tutto viene ponderato con la frequenza che determina il fatto che la rilevanza di un termine in un testo è tanto maggiore quanto quel termine compare più volte nel testo ed è raro negli altri testi.

4. **TFC weighting**: dato che la precedente formula non tiene in conto della lunghezza di ogni testo, per normalizzare tale aspetto si usa la formula:

$$r_{ij} = \frac{f_{ij} * \ln\left(\frac{m}{m_i}\right)}{\sqrt{\sum_{k=1}^{n}\left[\left(f_{kj} * \ln\left(\frac{m}{m_k}\right)\right)^2\right]}}$$

Al numeratore si ha la stessa formula della pesatura TFxIDF mentre al denominatore si tiene conto della lunghezza dei testi perché viene considerato il valore di rilevanza calcolato con il metodo TFxIDF su tutti i testi, effettuando pertanto una normalizzazione.

É da notare come al denominatore si considerano tutti i termini *k* del testo *j* pesati in tutti i testi $\ln\left(\frac{m}{m_k}\right)$, quindi più il testo considerato è lungo più il denominatore ha valore elevato. In particolare l'uso diffuso nel testo *j* di termini molto specifici non presenti in altri testi fa aumentare il valore del denominatore normalizzando quindi i pesi di tali termini.

5. **LTC weighting**: questo tipo di pesatura considera le ultime due formule usando il logaritmo della frequenza anziché il termine lineare:

$$r_{ij} = \ln(f_{ij} + 1) * \ln\left(\frac{m}{m_i}\right), \text{ TFxIDF logaritmico.}$$

$$r_{ij} = \frac{\ln(f_{ik}+1)*\ln\left(\frac{m}{m_i}\right)}{\sqrt{\sum_{k=1}^{n}\left[\left(\ln(f_{kj}+1)*\ln\left(\frac{m}{m_k}\right)\right)^2\right]}}, \text{ TFC logaritmico.}$$

Usando il logaritmo al posto della funzione lineare si ottiene il gradito effetto che per valori elevati dell'argomento la funzione considerata non tende ad "esplodere" mantenendo confrontabili le quantità in gioco.

6. **Entropy weighting**: usando questa formula il valore di rilevanza dipenderà dalla frequenza e dall'entropia che il termine ha nei vari testi:

$$\mathrm{r_{ij}} = \ln(f_{ij} + 1) * \left[1 + \frac{\sum_{k=1}^{m}\left(\frac{f_{ik}}{m_i} * \ln\left(\frac{f_{ik}}{m_i}\right)\right)}{\ln(m+1)}\right]$$

Dove:

$$\frac{\sum_{k=1}^{m}\left(\frac{f_{ik}}{m_i} * \ln\left(\frac{f_{ik}}{m_i}\right)\right)}{\ln(m+1)}$$

è il valore di entropia associato al termine i si nota come:

$f_{ik} \geq m_i \ \forall i,k \in N^+$ perché se un termine compare una sola volta significa che compare una volta in un testo, intuitivo.

Di conseguenza $\frac{f_{ik}}{m_i} * \ln\left(\frac{f_{ik}}{m_i}\right) \geq 0$ ed è zero quando $f_{ik} = 0 \vee f_{ik} = m_i$ cioè quando intuitivamente l'entropia dei termini tra i testi è nulla.

Rispetto alla versione presente in letteratura si è modificato il denominatore dentro la parentesi $\ln(m+1)$ rispetto all'originale $\ln(m)$ in modo da gestire senza errori il caso degenere in cui si ha un solo testo e quindi $m = 1 \Rightarrow \ln(m) = 0$.

3.8.1 Considerazioni relative alle tecniche di calcolo della rilevanza

Di queste formule si sono ignorate le prime due in quanto poco informative rispetto a valutazioni maggiormente sofisticate: per quanto riguarda la prima tecnica essa tende ad appiattire i valori associati a tutti i termini dando uguale rilevanza a parole significative e ad esempio a congiunzioni molto frequenti con poca rilevanza. Mentre per la seconda tecnica basta considerare che dal parallelo con gli algoritmi di indicizzazione dei motori di ricerca il mero conteggio della frequenza può portare ad un problema analogo al "Real keyword stuffing" (29) ossia la presenza numerosa di una parola nel testo tende a falsare la reale rilevanza del termine. Ovviamente un testo che contiene numerose ripetizioni di una parola relativamente alla frequenza delle altre è da ritenersi ambiguo. É chiaro quindi come è fondamentale "pesare" le frequenze delle varie parole tra loro. A fronte di queste considerazioni è quindi analiticamente dimostrato che è possibile ignorare queste due tecniche.

In tal senso si vuol far notare come in questa fase preliminare è fondamentale mantenere il numero più elevato di informazioni relativamente ai dati. In fasi successive quindi è possibile scremare o ridurre in maniera mirata la quantità di informazioni in gioco. Pertanto il fatto di mantenere informazioni relative all'entropia di un termine è sicuramente un vantaggio.

Visto inoltre il target di testi analizzato si è preferito tralasciare la tecnica TFC in quanto non apporta alcuna informazione aggiuntiva utile rispetto a TFxIDF ma necessita, calcolando a parte $\ln\left(\frac{m}{m_k}\right), k = 1 \dots n$ di $O(n)$ operazioni aggiuntive.

Dei tre metodi rimanenti: TFxIDF normale e logaritmico e Entropy la scelta è basata su due tipi di considerazioni. La prima è relativa alla solidità del calcolo effettuato che consiste nell'ottenere dei valori che tendono a mantenersi confrontabili in modulo, per questo le versioni logaritmiche sono preferibili. La seconda riguarda la "diffusione" del calcolo preferendo quindi formule che considerano maggiori informazioni disponibili durante il calcolo di un singolo valore di rilevanza.

A fronte di queste considerazioni si preferisce quindi TFxIDF logaritmico e Entropy a TFxIDF. Entropy inoltre garantisce una miglior "diffusione" rispetto a TFxIDF logaritmico in quanto per ogni valore di rilevanza vi è la pesatura con:

$$\frac{\sum_{k=1}^{m}\left(\frac{f_{ik}}{m_i} * \ln\left(\frac{f_{ik}}{m_i}\right)\right)}{\ln(m+1)}$$

Tuttavia Entropy con un piccolo accorgimento è possibile ridurlo ad una complessità computazionale di $O(n * m)$ o più specificatamente ad $O(n * m + n * m)$ pre-calcolando per ogni termine $i = 1 \dots n$ la quantità:

$$\left[1 + \frac{\sum_{k=1}^{m}\left(\frac{f_{ik}}{m_i} * \ln\left(\frac{f_{ik}}{m_i}\right)\right)}{\ln(m+1)}\right]$$

Che costa quindi, asintoticamente: $O(n * m)$.

A fronte di queste considerazioni si è scelto l'uso di Entropy per il calcolo della rilevanza.

3.9 Riduzione delle dimensioni della matrice di rilevanza

É ora chiaro come una matrice costruita con i metodi visti al precedente paragrafo risulti onerosa da trattare sotto ogni punto di vista e potrebbe contenere informazioni superflue che oltre a pesare in termini prestazionali possono risultare anche deleterie dal punto di vista della classificazione.

Innanzi tutto è quindi necessario rimuovere informazioni relative a parole poco significative per l'analisi. La letteratura ci viene incontro proponendoci i seguenti metodi:

1. **Soglia di frequenza**: si parte dal presupposto che una parola che compare raramente tra i testi sia poco importante.
   Questa sembra un'ipotesi fin troppo sbrigativa e banale e difatti lo è, basta considerare che per determinare una soglia realistica bisogna avere un set di dati di training da cui desumere un valore ragionevole. A questo punto, dato un set di dati classificato su cui fare training è prevedibile farne un uso più approfondito e intelligente per avere analisi più dettagliate. Inoltre bisogna considerare che una parola che compare poche volte può essere talmente specifica per una classe da comparire in tutti i testi che trattano un certo argomento che quindi eliminando tale parola non verrebbero classificati correttamente.
2. **Latent Semantic Index**: una interessante elaborazione per quanto riguarda la riduzione della matrice di rilevanza viene fornita dal Latent Semantic Index (LSI) che permette, tramite scomposizione a valori singolari (SVD), di estrarre le informazioni vettoriali più significative di una matrice, a riguardo conviene tornare al paragrafo 2.3 dove viene esposto questo metodo.
   Viene quindi considerata come matrice di rilevanza il prodotto:

$$V_s * \Lambda_s = \begin{bmatrix} \underline{s_1} \\ \vdots \\ \underline{s_n} \end{bmatrix}$$

Le cui righe rappresentano i testi, mantenendo le distanze relative tra loro a meno di un errore rispetto alla rappresentazione originale di R dato dall'approssimazione che si ha scegliendo i primi s autovalori di Λ:

$$E = \max_{i,j}\left(\left\|\left(\underline{s_i} - \underline{s_j}\right) - \left(\underline{r_i} - \underline{r_j}\right)\right\|_{\infty}\right) \leq C * (r - s)$$

Con r rango della matrice R, $s < r$ valore scelto per ridurre le dimensioni di R e $C \in \mathbb{R}^{+} - \{0\}$ costante a piacere.

In questo modo è inoltre possibile calcolare la similarità tra due vettori con un costo $O(s)$ anziché $O(m)$ come avveniva sulla matrice originale R. Da notare come, a fronte di un costo del calcolo della decomposizione SVD, è possibile quindi calcolare la distanza tra due vettori senza errori, $E = 0$, con costo $O(r)$ anziché $O(m)$ $r \leq m$.

Possiamo quindi dire che R e $V_s * \Lambda_s$ rappresentano due trasformazioni lineari omotetiche a meno di un errore E controllabile tramite la scelta di s.

La validità di queste osservazioni è empiricamente dimostrata dal sorgente matlab seguente che valuta sia la congruenza nel caso in cui non si riduca lo spazio con la scelta di $s = r$ sia la stima dell'errore compiuto in caso di scelta di un valore $s < r$:

```
clear
% generates sizes of matrix and subspace randomly
m=floor(rand()*10^(rand()*2)+10);
n=floor(rand()*10^(rand()*2)+10);
% ensure that m>n
if(m<n)
   temp = m;
   m = n;
   n = temp;
end
% generate R with real values in [-5, 5]
R = rand(m,n)*10-5;
% calculate economy size svd of R
[U D V]=svd(R,0);
% calculate omomorphic matrix V*D
VD = V*D;
% check that every line of VD has same distance to each other respect to
% every line of R
```

```
for i=1:size(VD)-1
   for j=i+1:size(VD)
      distr(i,j)=norm(R(:,i)-R(:,j));
      distvd=norm(VD(i,:)-VD(j,:));
      ratio = distr(i,j)/distvd;
      if(ratio > 1+1e3*eps || ratio < 1-1e3*eps)
         disp(sprintf('ratio is %d 1 for indexes: %d %d', ratio, i, j));
      end
   end
end
% find s where eigenvalue is half of biggest eigenvalue
s=size(D);

for i=1:size(D)
   if(D(i,i)/D(1,1) < 0.5)
      s = i;
      break;
   end
end
% calculate reduced matrix
Ds=D(1:s,1:s);
Vs=V(:,1:s);
VsDs = Vs*Ds;
% check changes of "shape" of vectors in R and VsDs
ratiomax = 0;

for i=1:size(VsDs)-1
   for j=i+1:size(VsDs)
      distvsds=norm(VsDs(i,:)-VsDs(j,:));
      ratio = distr(i,j)/distvsds;
      if(ratiomax < ratio)
         ratiomax = ratio;
         im = i;
         jm = j;
      end
   end
end

ri = R(:,im)
```

```
rj = R(:,jm)
si = VsDs(im,:)'
sj = VsDs(jm,:)'
ratiomax
norm(ri-rj)/norm(si-sj)
```

3. **Information Gain**: questo metodo predice la quantità di informazioni che un dato termine adduce alla classificazione dei testi:

$$IG(t) = -\sum_{i=1}^{p}\big(P(c_i) * \ln(\mathrm{P(c_i)})\big) + P(t) * \sum_{i=1}^{p}\big(P(c_i|t) * \ln\big(P(c_i|t)\big)\big) + P(\bar{t}) * \sum_{i=1}^{p}\big(P(c_i|\bar{t}) * \ln\big(P(c_i|\bar{t})\big)\big)$$

Dove: $\{c_1 \dots c_p\}$ è l'insieme delle classi, t è il termine considerato, $P(c_i)$ è la probabilità che un generico testo sia classificato in c_i, $P(t)$ è la probabilità che un testo generico contenga il termine t, $P(c_i|t)$ è la probabilità che un testo di classe c_i contenga il termine t e infine $P(c_i|\bar{t})$ è la probabilità che un testo di classe c_i non contenga il termine t.

Quindi per stimare i vari valori di probabilità è possibile semplicemente contare e fare il rapporto. Il vero problema però è come stimare tali valori se non si sono già classificati i testi. Questa formula pertanto è utile quando si ha a disposizione un insieme di dati già classificati, training set.

A fronte di queste ipotesi si può stimare:

$P(t) = \frac{m_t}{m}$ con m_t numero di testi che contengono il termine t e m numero di testi totali.

$P(c_i) = \frac{m_{ci}}{m}$ con m_{ci} numero di testi di classe c_i.

$P(c_i|t) = \frac{m_{cit}}{m}$ con m_{cit} numero di testi di classe c_i che contengono il termine t, questa probabilità è anche approssimabile come

$$P(c_i|t) \cong P(t) * P(c_i) = \frac{m_t * m_{ci}}{m^2}$$

anche se tale approssimazione è poco accurata per termini non distribuiti uniformemente nei testi appartenenti alle varie classi, si viene a creare pertanto un bias elevato tra il valore stimato con la formula e quello reale per termini specifici.

Pensando tuttavia a come computare il termine m_{cit} avendo un insieme di dati di test questo lo si può fare durante il conteggio di m_t tenendo traccia di un dato aggiuntivo relativo alla classe di appartenenza del testo.

Infine, ovviamente, $P(c_i|\bar{t}) = 1 - P(c_i|t)$ da calcolarsi immediatamente.

4. **Statistica χ^2**: dovendo trattare il disaccoppiamento tra le varie classi c_i e i termini generici t quello che è possibile costruire è una statistica chiquadro. Ma il parametrizzare un intero tool statistico avrebbe un costo computazionale insostenibile, quello che si fa è costruire una formula che fornirà una stima del disaccoppiamento tra una classe e un termine:

$$\chi^2(t, c_i) = \frac{m * (A * D - B * C)^2}{(A + C) * (B + D) * (A + B) * (C + D)}$$

Questa formula fornisce un valore di dipendenza tra un termine t e una classe c_i da cui è possibile poi scremare i vari termini che hanno una dipendenza tra le varie classi molto bassa.

Il significato dei simboli è:

A: numero di testi della classe c_i che contengono il termine t.

B: numero di testi che contengono t ma che non appartengono a c_i.

C: numero di testi appartenenti a c_i ma che non contengono t.

D: numero di testi che non appartengono a c_i e non contengono t.

Contando quindi, analogamente alla formula di Information Gain: m, m_{ci}, m_t, m_{cit} è possibile trovare il valore di questi parametri e quindi il calcolo del disaccoppiamento non è più oneroso computazionalmente rispetto all'Information Gain.

In particolare:

$$A = m_{cit}$$
$$B = m_t - m_{cit}$$
$$C = m_{ci} - m_{cit}$$
$$D = 1 - (m_{cit} + m_t - m_{cit})$$

Il significato di queste formule è facilmente verificabile guardando la figura:

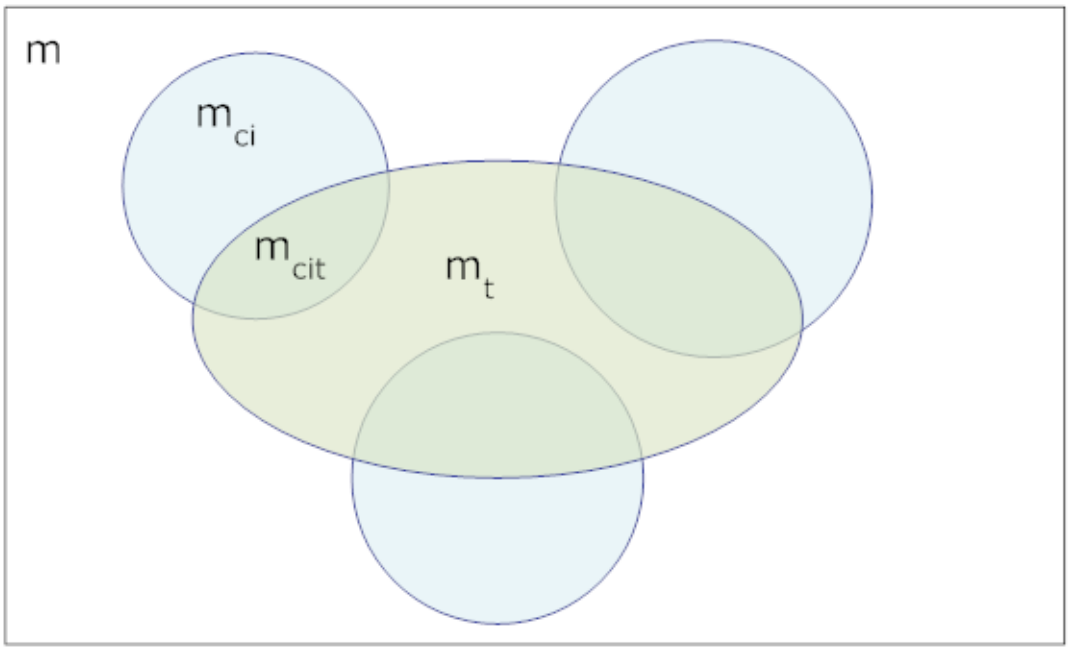

Figura 3-5 Diagramma di Venn Relativo al Conteggio delle Appartenenze

Successivamente, calcolato il valore $\chi^2(t, c_i)$ relativo al termine t per ogni classe c_i è possibile stabilire il valore di accoppiamento tra il termine t e tutte le classi normando secondo una formula valida come norma, ad esempio:

$$\chi^2(t) = \sum_{i=1}^{p} \left(\frac{m_{ci}}{m} * \chi^2(t, c_i) \right)$$

Oppure usando la più generale norma infinito:

$$\chi^2(t) = \|\chi^2(t, c_i)\|_\infty \, , i = 1 \dots p$$

Trovato pertanto un valore è possibile escludere quei termini che hanno un accoppiamento con le classi inferiore di un certo valore soglia parametrizzato con un insieme di test o tramite raffinamenti successivi.

5. **Term weighting**: data la matrice di rilevanza R, Figura 3-4, si eliminano le righe corrispondenti ai termini che hanno una rilevanza inferiore ad una data soglia t, in formule:

$$R_{new} = \left\{ \begin{bmatrix} \underline{r_1} \\ \vdots \\ \underline{r_p} \end{bmatrix} \; t.c. \, 1 \leq i \leq p \; \wedge \; \underline{r_i} \in R \; \wedge \; \left\| \underline{r_i} \right\|_\infty \geq t \right\}$$

$$t = \frac{max\left(\underline{r_i}\right) - min\left(\underline{r_i}\right)}{10} + min\left(\underline{r_i}\right), \forall \underline{r_i} \in R$$

3.9.1 Considerazioni sulla riduzione della matrice di rilevanza

La scelta dei metodi di riduzione delle dimensionalità da applicare tra quelli qui esposti si basa fondamentalmente sull'applicabilità del metodo. Il metodo relativo alla soglia di frequenza è stato escluso come visto per via delle sue ipotesi semplicistiche che possono inficiare l'ottenimento di buoni risultati.

Per quanto riguarda gli altri metodi bisogna escludere i metodi di Information Gain e Statistica χ^2 in quanto necessitano della conoscenza relativa alle classi da considerare, tali classi possono non essere note a priori per quanto riguarda questo lavoro, quindi sono inapplicabili. LSI, per converso, è un'interessante elaborazione ma è utile solo nel caso in cui si consideri la distanza tra i vettori rappresentanti i testi. Come esposto di seguito l'algoritmo proposto necessita invece delle informazioni relative ai termini e quindi questo metodo non è applicabile. Pertanto non è stato applicato nemmeno agli algoritmi di confronto in quanto si vuole avere una misurazione dell'efficienza dei metodi partendo da una base comune che è rappresentata dalla matrice di rilevanza R.

Nulla vieta in lavori futuri di provare a valutare l'apporto in termini prestazionali di una riduzione mediante il metodo LSI.

L'unico metodo applicabile pertanto è term weighting.

3.10 Algoritmo di classificazione: ExhaustiveSets

Avendo quindi a disposizione la matrice di rilevanza R e osservandone la struttura in alcune simulazioni si nota che essa è fortemente sparsa e ha una struttura regolare simil-diagonale:

$$R = \begin{bmatrix} \# & & & & & & & \\ \# & & & & & & & \\ & \# & & & & & & \# \\ & & \# & & & \# & & \\ & & \# & & & & & \\ & & & \# & & & \# & \\ & & & \# & & \# & & \\ & & & & \# & & & \\ & & & & & \# & & \# \end{bmatrix}$$

Dove le righe rappresentano i termini, le colonne i testi e # sono numeri in $\mathbb{R}$ diversi da zero, essa per via della brevità dei messaggi, è fortemente sparsa e molto allungata rispetto alle colonne vista la numerosità dei testi.

Lo schema concettuale di riferimento a cui afferisce questo algoritmo è quello riportato.

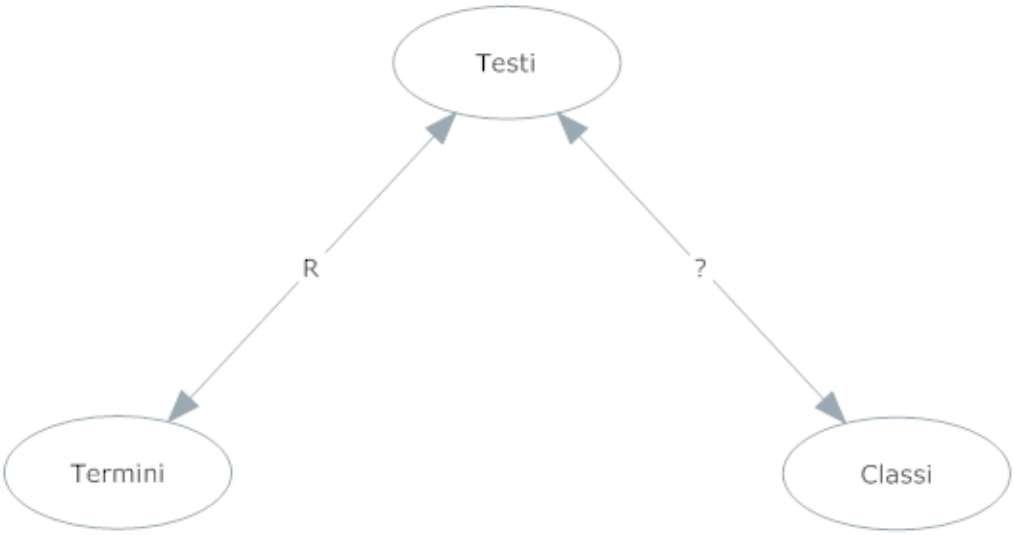

Figura 3-6 Relazione Termini-Testi-Classi ExhaustiveSets

Non vengono forniti dati circa i termini che afferiscono ad una data classe, viene fornita solo la matrice di rilevanza che esprime la relazione tra termini e testi. Tramite l'algoritmo si vuole classificare i testi in classi da costruire opportunamente.

Queste osservazioni sono utili alla definizione dell'algoritmo che dovrà far leva su una serie di aspetti utili:

1. **Dimensione di riferimento**: la dimensione di riferimento devono essere i termini in quanto in numero minore rispetto ai testi e più invariante rispetto all'inserimento di nuovi testi.
2. **Natura della matrice**: ovviamente l'algoritmo dovrà sfruttare la natura sparsa della matrice. In particolare dovrà essere in grado di sfruttare la rappresentazione per matrici sparse adottata.
3. **Approccio completo**: l'algoritmo dovrà avere un approccio il più completo possibile rispetto alla classificazione, cioè non trascurare possibili computazioni che possono dar luogo a classificazioni corrette qualitativamente.

A fronte di una permutazione arbitraria delle colonne è possibile individuare, seppur sporadici, blocchi di parole appartenenti a testi diversi. In particolare se i pochi termini di un testo compaiono in altri testi con rilevanza ponderante è facilmente intuibile l'appartenenza di tali testi ad una classe comune.

R=

	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}
t_1	#								#	
t_2	#							#		
t_3	#		#				#			
t_4		#	#	#					#	
t_5			#	#			#		#	
t_6				#				#		#
t_7					#	#				#

Figura 3-7 Blocchi Matrice di Rilevanza

Con riferimento alla Figura 3-7 il rettangolo in linea continua ci fa notare come ad esempio i testi d_1 e d_9 condividono il termine t_1 questo può suggerirci che d_1 e d_9 possano avere un contenuto attinente e quindi appartenere alla stessa classe c_1 con un peso di appartenenza esprimibile dalla rilevanza $r_{1,1}$ e $r_{1,9}$ rispettivamente, in formule:

$$classet(c_1) = \{d_1, d_9\}$$

$$rel(d_1, c_1) = r_{1,1}$$

$$rel(d_9, c_1) = r_{1,9}$$

$$termset(\{d_1, d_9\}) = \{t_1\}$$

$$docset(t_1) = \{d_1, d_9\}$$

Ovviamente d_1 può allo stesso modo appartenere anche ad un'altra classe c_2 con ad esempio d_8 e l'attinenza di questo assegnamento sarà la rilevanza $r_{2,1}$ e $r_{2,8}$:

$$classet(c_2) = \{d_1, d_8\}$$

È quindi possibile costruire per ogni testo una classifica delle classi di appartenenza ordinata secondo l'attinenza dell'assegnamento tra testo e classe.

Osservando invece il rettangolo tratteggiato in figura Figura 3-7 si nota come i testi d_3 e d_4 possiedono i termini t_4 e t_5, la loro classificazione sarà:

$$classet(c_3) = \{d_3, d_4\}$$

$$rel(d_3, c_3) = r_{4,3} + r_{5,3}$$

$$rel(d_4, c_3) = r_{4,4} + r_{5,4}$$

$$termset(\{d_3, d_4\}) = \{t_4, t_5\}$$

$$docset(\{t_4, t_5\}) = \{d_3, d_4\}$$

Formalizzando quanto esposto fino ad ora abbiamo:

1. **Insieme dei termini**: $t_i \in T, i = 1 \dots n$
2. **Insieme dei testi**: $d_j \in D, j = 1 \dots m$
3. **Insieme delle classi**: $c_k \in C, k = 1 \dots c$

4. **Matrice di rilevanza**: $R \in \mathbb{M}_{\mathbb{R}}^{p \times m}$ matrice di rilevanza ottenuta dopo le fasi di riduzione delle dimensionalità dove quindi $p \leq n$ in accordo con la fase di term weighting, cambierà quindi anche l'insieme dei termini.
5. **Insieme dei termini ridotto**: $t_i \in T, i = 1 \dots p$
6. **Funzione tra classi e testi**: $classet: C \to D^m$ associa ad una classe i testi che afferiscono ad essa.
7. **Funzione di attinenza**: $rel: D \times C \to \mathbb{R}$ esprime l'attinenza di un testo rispetto ad una classe.

$$rel(d_j, c_k) = \sum_{t_i \in termset(classet(c_k))} r_{i,j}$$

8. **Relazione tra testi e termini**: $termset: D^m \to T^p$ funzione che associa ad ogni insieme di testi un insieme di termini in comune tra i vari testi.
9. **Relazione tra termini e testi**: $docset: T^p \to D^m$ associa un insieme di termini identificato da un $termset$ ai testi che li contengono, è la relazione inversa alla precedente.

È quindi chiaro che ogni classe è rappresentata da una combinazione di termini e per una ricerca esaustiva delle possibili appartenenze dei testi alle classi è necessario costruire tutte le combinazioni tra i termini $termset$.

Volendo essere più precisi è necessario valutare, in via teorica, tutte le combinazioni di termini t_i che sono:

1. Senza ripetizioni.
2. Non dipendenti dalla posizione: cioè ad esempio $(t_1, t_2) = (t_2, t_1)$
3. Lunghe da 1 a p.

Ogni insieme così fatto andrà a comporre un $termset$ associato ad una classe a cui di conseguenza apparterranno dei testi con una relativa $attinenza$.

3.10.1 Complessità teorica

La complessità di questo metodo consisterà quindi nella costruzione di tutte le combinazioni possibili di termini. Per effettuare questo calcolo ci viene in aiuto la teoria del calcolo combinatorio: dato l'insieme dei termini $T = \{t_1 \dots t_p\}$ con $|T| = p$ si vuole valutare il numero di combinazioni ottenibili valutando tutte le combinazioni di classe che va da $r = 1 \dots p$. La teoria del calcolo combinatorio ci indica che il numero di combinazioni di classe r su p elementi è:

$$C(p,r) = \binom{p}{r}$$

Se vogliamo valutare ogni classe i di combinazioni con $i = 1 \dots p$ si ottiene un numero di combinazioni totali pari a:

$$C = \sum_{i=1}^{p} C(p,r) = \sum_{i=1}^{p} \binom{p}{i}$$

Che è dimostrato fornire come risultato:

$$C = 2^p - 1$$

Questo risultato ci dice che in via teorica la complessità dell'algoritmo è esponenziale $O(2^p)$, una caratteristica poco allettante e performante.

Tuttavia con le dovute osservazioni e ottimizzazioni è possibile ridurre drasticamente tale complessità e come vedremo pervenire ad un algoritmo piuttosto lineare.

3.10.2 Ottimizzazioni

La costruzione sistematica di tutte le possibili combinazioni di termini di classe $r = 1 \dots p$ come visto porta ad una complessità esponenziale che computazionalmente è insostenibile.

E' quindi necessario imporre alcune regole che permettano di ridurre il numero di calcoli necessari alla classificazione dei testi:

1. **Docset a contenuto singolo**: sapendo che parecchi termini appartengono ad un solo testo, quando ottengo un $docset$, con relativo $termset$, che contiene un solo testo allora escludo il $termset$ relativo dalle computazioni

successive. In sostanza si usano anche le informazioni relative all'appartenenza dei testi per non generare ciecamente tutte le combinazioni tra termini. Il fatto di escludere i $docset$ con un singolo testo dalle computazioni successive è arbitrario, si sarebbero potuto includere anche tali $docset$ aumentando il carico computazionale o escludere $docset$ più ampi, con due o più testi. Tuttavia osservando empiricamente la matrice di rilevanza si ha in percentuale un numero di termini che appartengono ad un singolo testo pari all'80% e il numero di $docset$ che contengono più di un testo hanno un andamento di tipo logaritmico rispetto all'aumento della cardinalità dei $termset$ che si ottiene nelle varie computazioni.

2. **Intersezioni vuote tra $termset$**: in accordo con quanto detto in precedenza le combinazioni devono essere generate controllando che non vi siano termini in comune tra i $termset$ in quanto si avrebbe che il $docset$ ottenuto comprenderà per forza di cose testi che contengono i termini in comune tra i $termset$:

$$termset(\{d_1, d_2, d_3\}) = \{t_1, t_2\} \Leftrightarrow docset(\{t_1, t_2\}) = \{d_1, d_2, d_3\}$$

$$termset(\{d_3, d_4\}) = \{t_2, t_3\} \Leftrightarrow docset(\{t_2, t_3\}) = \{d_3, d_4\}$$

$$termset(\{d_1, d_2, d_3\}) \cap termset(\{d_3, d_4\}) = \{t_1, t_2\} \cap \{t_2, t_3\} = \{t_2\}$$

$$docset(\{t_2\}) = docset(\{t_1, t_2\}) \cap docset(\{t_2, t_3\}) = \{d_1, d_2, d_3\} \cap \{d_3, d_4\} = \{d_3\}$$

Che è una computazione inutile in quanto sicuramente nella prima passata si è già individuata la classe relativa a:

$$docset(\{t_2\}) = \{d_3\}$$

3. **Esclusione delle combinazioni già esplorate**: tengo traccia nell'algoritmo delle combinazioni di $termset$ già esplorate per evitare computazioni ripetute.
4. **Procedura incrementale**: la generazione dei $termset$ e la computazione procede incrementalmente con opportune condizioni di stop che limitano il numero di computazioni allo stretto necessario.
5. **Ricombinazione intelligente**: ricombino solo i nuovi $termset$ generati nel passaggio precedente in quanto quelli già generati saranno già stati ricombinati in passaggi precedenti tra loro.

3.10.3 Pseudocodice ExhaustiveSets

Lo pseudocodice dell'algoritmo è esposto di seguito. Esso fa uso di matrici associative, chiamate anche mappe e dell'algebra insiemistica. Nei paragrafi seguenti verranno esposte considerazioni relative alla complessità.

Essendo un algoritmo di "copertura" delle varie combinazioni esso opera a passate successive, imponendo una condizione di terminazione sicura quando il numero di passate è pari a $p = |T|$ ossia quando i $termset$ generati hanno classe massima $r = p$, in accordo con la teoria combinatoria che sta alla base della complessità teorica dell'algoritmo.

Chiaramente vi è una condizione di terminazione anticipata che avviene quando i $termset$ generati non danno luogo a nuovi $docset$ da analizzare che, vista la sparsità della matrice R, è una condizione che si verifica in pratica nella totalità dei casi.

```
Function ExhaustiveSets(R, T, D)
{
        docset:={}
        termset:={}
        lasttermset:={}
        prevtermset:={}
        gentermset:={}

        for t_i∈T
        {
                docset({t_i}):={d_j∈D t.c. r_ij≠0}
                termset(docset({t_i})):={t_i}
                if |docset({t_i})|≥1
                {
                        generateClass(docset({t_i}),{t_i})
                        if |docset({t_i})|≥2
                        {
                                lasttermset:=lasttermset∪{t_i}
                                prevtermset:=prevtermset∪{t_i}
                        }
                }
        }
        sweep:=1
```

```
        do
        {
                sweep:=sweep+1
                gentermset:={}
                for tsl∈lasttermset
                        for tsp∈prevtermset
                        {
                                if tsl∩tsp=={} ∧ tsl∪tsp∉termset
                                {
                                        dsgen:=docset(tsl)∩docset(tsp)
                                        tsgen:=tsl∪tsp
                                        if |dsgen|≥1
                                        {
                                                generateClass(dsgen,tsgen)
                                                if |dsgen|≥2
                                                {
                                                        gentermset:=gentermset∪tsgen
                                                        docset(tsgen):=dsgen
                                                }
                                        }
                                        termset(dsgen):=tsgen
                                }
                        }
                prevtermset:=prevtermset∪(lasttermset-prevtermset)
                lasttermset:=gentermset
        }
        while gentermset≠{} ∧ sweep≤|T|
}
```

L'algoritmo si basa sull'intersezione tra i due insiemi generati ad ogni passaggio: $lasttermset$ e $prevtermset$ tramite i cicli for annidati:

```
for tsl∈lasttermset
        for tsp∈prevtermset
```

La complessità quindi dipenderà dalle dimensioni di tali insiemi ad ogni iterazione. In particolare $lasttermset$ contiene le combinazioni di termini appena generate mentre $prevtermset$ contiene le combinazioni di termini generate in passate precedenti.

Inoltre bisogna notare come in $docset$ vengono salvate solo le informazioni relative a classi che vengono poi effettivamente create, non vi è quindi una corrispondenza biunivoca tra il contenuto di $docset$ e $termset$ e questo ovviamente consente un risparmio di spazio. È invece necessario salvare tutte le informazioni di $termset$ ossia tutte le combinazioni di termini valutate in modo da poter evitare la valutazione doppia di alcune combinazioni, si veda il controllo nel costrutto if:

```
tsl∪tsp∉termset
```

L'inizializzazione dell'algoritmo consiste nella generazione di due insiemi di $termset$ uguali:

```
lasttermset:=lasttermset∪{tᵢ}
prevtermset:=prevtermset∪{tᵢ}
```

Questo perché con la seconda passata sarà possibile costruire le combinazioni di classe $r = 2$ e procedere con le passate successive.

Infine bisogna specificare un particolare, l'istruzione:

```
prevtermset:=prevtermset∪(lasttermset-prevtermset)
```

che aggiorna $prevtermset$ si premura di scremare $prevtermset$ da $lasttermset$, questo ci è utile alla fine della seconda passata in quanto la prima passata aveva impostato, come visto:

$$prevtermset = lasttermset$$

E facendone l'unione si otterrebbe la ripetizione degli insiemi.

La funzione $generateClass$ si limita a generare la classe relativa alla corrispondenza $docset\text{-}termset$ passata come argomento e appena computata.

3.10.4 Esempio di computazione

Si propone ora un esempio di risoluzione relativo alla matrice di rilevanza:

R=

	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8
t_1	#							
t_2	#	#	#				#	#
t_3	#			#				#
t_4		#	#		#		#	#
t_5			#		#	#		#

Figura 3-8 Matrice Rilevanza Esempio ExhaustiveSets

Viene quindi fornita una tabella che mostra l'evoluzione dell'algoritmo, in particolare le colonne relative a *docset* e *termset* mostrano l'evoluzione della popolazione di tali strutture dati che ricordiamo essere matrici associative. Le altre colonne invece mostrano lo stato delle strutture dati a fine passata per quanto riguarda *lasttermset*, *prevtermset* e *gentermset* e lo storico degli assegnamenti per quanto riguarda *flag*, *dsgen* e *tsgen*. Sono omesse le informazioni relative alle classi create.

DOCSET		**TERMSET**		**LASTTERMSET**	**PREVTERMSET**	**GENTERMSET**	**FLAG**	**DSGEN**	**TSGEN**	**FINE PASSO SWEEP**
TERMSET	DOCSET	DOCSET	TERMSET							
t_1	d_1	d_1	t_1	t_2	t_2	//		//	//	
t_2	$d_1\ d_2\ d_3\ d_7\ d_8$	$d_1\ d_2\ d_3\ d_7\ d_8$	t_2	t_3	t_3	//		//	//	
t_3	$d_1\ d_4\ d_8$	$d_1\ d_4\ d_8$	t_3	t_4	t_4	//	//	//	//	**1**
t_4	$d_2\ d_3\ d_5\ d_7\ d_8$	$d_2\ d_3\ d_5\ d_7\ d_8$	t_4	t_5	t_5	//		//	//	
t_5	$d_3\ d_5\ d_6\ d_8$	$d_3\ d_5\ d_6\ d_8$	t_5			//		//	//	
$t_2\ t_3$	$d_1\ d_8$	$d_1\ d_8$	$t_2\ t_3$	$t_2\ t_3$	t_2	$t_2\ t_3$		$\{d_1\ d_8\}$	$\{t_2\ t_3\}$	
$t_2\ t_4$	$d_2\ d_3\ d_7\ d_8$	$d_2\ d_3\ d_7\ d_8$	$t_2\ t_4$	$t_2\ t_4$	t_3	$t_2\ t_4$	T	$\{d_2\ d_3\ d_7\ d_8\}$	$\{t_2\ t_4\}$	**2**
$t_2\ t_5$	$d_3\ d_8$	$d_3\ d_8$	$t_2\ t_5$	$t_2\ t_5$	t_4	$t_2\ t_5$		$\{d_3\ d_8\}$	$\{t_2\ t_5\}$	

//	//	d_8	$t_3\ t_4$	$t_4\ t_5$	t_5	$t_4\ t_5$		{d_8}	{$t_3\ t_4$}	
//	//	d_8	$t_3\ t_5$			Vuoto a fine passo		{d_8}	{$t_3\ t_5$}	
$t_4\ t_5$	$d_3\ d_5\ d_8$	$d_3\ d_5\ d_8$	$t_4\ t_5$					{$d_3\ d_5\ d_8$}	{$t_4\ t_5$}	
//	//	d_8	$t_2\ t_3\ t_4$	$t_2\ t_4\ t_5$	t_2	$t_2\ t_4\ t_5$	T	{d_8}	{$t_2\ t_3\ t_4$}	**3**
//	//	d_8	$t_2\ t_3\ t_5$		t_3	Vuoto a fine passo		{d_8}	{$t_2\ t_3\ t_5$}	
$t_2\ t_4\ t_5$	$d_3\ d_8$	$d_3\ d_8$	$t_2\ t_4\ t_5$		t_4			{$d_3\ d_8$}	{$t_2\ t_4\ t_5$}	
//	//	d_8	$t_3\ t_4\ t_5$		t_5			{d_8}	{$t_3\ t_4\ t_5$}	
					$t_2\ t_3$					
					$t_2\ t_4$					
					$t_2\ t_5$					
					$t_4\ t_5$					
//	//	d_8	$t_2\ t_3\ t_4\ t_5$	{}	t_2	{}	F	{d_8}	{$t_2\ t_3\ t_4\ t_5$}	**4**
					t_3					
					t_4					
					t_5					
					$t_2\ t_3$					
					$t_2\ t_4$					
					$t_2\ t_5$					
					$t_4\ t_5$					

Tabella 2 Evoluzione ExhaustiveSets

É interessante notare come la maggior parte delle computazioni riguardano il testo d_8 che contiene quasi tutti i termini disponibili. Questa, per via della sparsità della matrice, è una condizione molto remota pertanto una forte riduzione della complessità è facilmente prevedibile. Inoltre si sono compiuti 4 passi anziché 5 come invece avrebbe previsto la costruzione di tutte le combinazioni di classe $r = 1 \dots 5$, questo risultato

verrà ulteriormente migliorato tanto più la matrice sarà sparsa, quella usata nell'esempio non lo è tuttavia un miglioramento c'è stato.

3.10.5 Creazione delle classi

Nello pseudocodice l'aggiunta di una classe avveniva tramite la chiamata:

```
generateClass(dsgen,tsgen)
```

Dove quindi venivano specificati i testi appartenenti alla classe e i termini a cui faceva riferimento la classe stessa. Inoltre, in accordo con quanto scritto nel paragrafo 3.10 viene anche salvato il valore di attinenza tra testo e classe.

Pertanto la struttura dati che memorizza una classe conterrà:

1. Un elenco di termini associati alla classe: $termset(classet(\dots))$
2. Un elenco di testi di tale classe: $classet(\dots)$
3. Una struttura dati associativa, o mappa, che associa ad un dato testo il relativo valore di attinenza con la classe: $rel(\dots,\dots)$

Lo pseudocodice che crea una nuova classe sarà:

```
Function generateClass(docset,termset)
{
        c.termset:=termset
        c.classet:=docset
        tempRel:=0

        for d∈docset
                for t∈termset
                        tempRel:=tempRel+R(t,d)

        c.rel[d]:=tempRel
        classes.add(c)
}
```

La complessità di questa funzione sarà pari al prodotto della grandezza di $docset$ e $termset$ passati in ingresso data la presenza dei due cicli for annidati:

$$O(|docset| * |termset|)$$

3.10.6 Scrematura delle classi

Quanto ottenuto fino ad ora però contrasta con le ipotesi fatte nel paragrafo 3.2 poiché ad un testo possono essere assegnate più classi.

Questo impone una fase di scrematura che consideri per ogni testo la classe con l'attinenza maggiore tra quelle a cui è assegnato.

Lo pseudocodice relativo è esposto di seguito:

```
for c∈C
        if |c.classet| == 1
                classes.remove(c)

for d∈D
{
        classmax(d):=max_{c∈C}(rel(d,c))
        for c∈classes-{classmax(d)}
        {
                c.remove(d)
                if c=={}
                        classes.remove(c)
        }
}
```

Innanzi tutto è necessario eliminare le classi con un solo testo contenuto, queste classi sono facilmente identificate con un insieme di termini $termset$ che pressoché coincide con i termini presenti nell'unico testo. Queste operazioni hanno un costo computazionale di $O(c)$.

In seguito l'algoritmo cerca la classe che ha attinenza massima rispetto al testo tramite l'istruzione:

```
classmax(d):=max_{c∈C}(rel(d,c))
```

anche questo ha un costo di $O(c)$.

In seguito scandisce le restanti classi rimuovendo il testo in modo da garantire che la relazione sia una funzione, quindi se la classe è vuota la rimuove dall'insieme delle classi:

```
if c=={}
        classes.remove(c)
```

in modo da garantire la suriettività. Questo passaggio costerà $O(c-1)$.

La complessità totale di questa fase sarà: $O\big(c + m * (2 * c - 1)\big)$.

3.10.7 Complessità reale

Viene qui proposta un'analisi relativa alla complessità, vengono considerate le prime iterazioni dell'algoritmo. Come intuibile l'algoritmo ha una complessità non facilmente rappresentabile e fortemente dipendente dalla distribuzione degli elementi diversi da zero in R. Tuttavia vi sono semplificazioni utili a ricondurre la complessità ad una complessità nota e poter avere un confronto esplicito seppur a fronte di maggiorazioni e assunzioni conservative.

Prima passata

La prima passata è effettuata dal ciclo for iniziale con costo $O(p)$:

```
for t_i∈T
{
        docset({t_i}):={d_j∈D t.c. r_ij≠0}
        termset(docset({t_i})):={t_i}
        if |docset({t_i})|≥1
        {
                generateClass(docset({t_i}),{t_i})
                if |docset({t_i})|≥2
                {
                        lasttermset:=lasttermset∪{t_i}
                        prevtermset:=prevtermset∪{t_i}
                }
        }
}
```

Ed esplora tutte le righe di R, costruisce gli insiemi $lasttermset$ e $prevtermset$ inserendo tutti i termini che stanno in almeno due testi, a questo punto bisogna tener conto di alcune considerazioni:

1. Se una matrice è sparsa si può assumere che il numero degli elementi diversi da 0 sia il 10% del totale, quindi:

   $$R \in \mathbb{M}_{\mathbb{R}}^{p \times m}$$

   Numero massimo di elementi diversi da zero:

   $$q = \frac{p * m}{10}$$

2. La matrice è riempita massimo al 10% questo significa che se essa ha almeno dieci colonne e una di queste è composta da numeri diversi da zero allora ho un riempimento del 10%:

   $$R = \begin{bmatrix} \# & & & & & & & & & \\ \# & & & & & & & & & \\ \# & & & & & & & & & \end{bmatrix}$$

3. Per costruzione però ogni testo ha almeno un termine quindi ogni colonna avrà almeno un valore diverso da zero.

   $$R = \begin{bmatrix} \# & & & & & & & & & \\ & \# & & & & & & & & \\ & \# & & & & & & & & \\ & & \# & & & & \# & & & \\ & & & \# & & & & & & \\ & & & \# & & & & \# & & \\ & & & & \# & & & & & \# \\ & & & & & \# & & & \# & \\ & & & & & & \# & \# & & \end{bmatrix}$$

4. Quindi se R ha più di dieci colonne è possibile che qualche riga abbia più di un elemento diverso da zero mantenendo la matrice di tipo sparso.

A fronte di queste considerazioni si può dire che dopo il primo ciclo for gli insiemi $prevtermset$ e $lasttermset$ conterranno un numero di elementi pari al numero di righe con almeno due valori diversi da zero, cioè una frazione di q. Tali termini saranno parole molto frequenti e di uso comune che la fase di Term Weight screma facilmente, riducendo ancora di più tale frazione.

Supponendo di avere una matrice molto allungata rispetto alle colonne $m \gg p$ e con un numero di elementi al limite della soglia del 10%: $\left|\{r_{ij} \neq 0\}\right| = q$ volendo fare un'ipotesi conservativa si assume che:

$$|lasttermset| = \frac{|T|}{5} = \frac{p}{5} \wedge |prevtermset| = \frac{|T|}{5} = \frac{p}{5}$$

Tenendo conto che statisticamente l'80% dei termini compaiono in un solo testo. Tuttavia questa è una statistica empirica e non si hanno evidenze della sua validità assoluta pertanto si assumerà l'unico risultato sicuramente corretto e conservativo:

$$|lasttermset| = p \wedge |prevtermset| = p$$

Seconda passata

La seconda passata si svolge all'interno del ciclo while e itera con due for annidati su $prevtermset$ e $lasttermset$ che sono grandi entrambi p quindi costa $O(p^2)$:

```
for tsl∈lasttermset
        for tsp∈prevtermset
        {
                if tsl∩tsp=={} ∧ tsl∪tsp∉termset
                {
                        dsgen:=docset(tsl)∩docset(tsp)
                        tsgen:=tsl∪tsp
                        if |dsgen|≥1
                        {
                                generateClass(dsgen,tsgen)
                                if |dsgen|≥2
                                {
                                        gentermset:=gentermset∪tsgen
                                        docset(tsgen):=dsgen
                                        flag:=true
                                }
                        }
                        termset(dsgen):=tsgen
                }
        }
prevtermset:=prevtermset∪(lasttermset-prevtermset)
lasttermset:=gentermset
```

```
gentermset:={}
```

Bisogna qui far notare che essa compie p^2 iterazioni ma con il controllo if evita le iterazioni dove:

```
tsl∩tsp≠{} ∧ tsl∪tsp∈termset
```

Che nel caso in cui $prevtermset = lasttermset$ consiste nella combinazione di p elementi su due posizioni, quindi entrerà nel costrutto if:

$$\binom{p}{2} = \frac{p * (p-1)}{2}$$

volte, che è assimilabile ad una complessità $O\left(\frac{p^2}{2}\right)$.

Quindi aggiorna gli insiemi $prevtermset$ e $lasttermset$ per la passata successiva, questo aggiornamento si basa sulla costruzione di un terzo insieme $gentermset$ che contiene le coppie di termini che compaiono in almeno due testi, chiaramente questo valore in una matrice sparsa è $s \ll p$, approssimabile a:

$$s = \ln p$$

Da cui, se:

```
prevtermset:=prevtermset∪(lasttermset-prevtermset)
```

allora

$$|prevtermset| = 2 * p$$

E inoltre:

```
lasttermset:=gentermset
```

allora

$$|lasttermset| = \ln p$$

Terza passata

Con i valori di cardinalità appena calcolati è possibile ricavare la complessità della terza passata $O(2 * p * \ln p)$.

Quindi analogamente alla passata precedente avremo che:

$$|prevtermset| = 2 * p + \ln p$$

$$|lasttermset| = \ln(\ln p)$$

La complessità delle prime quattro passate sarà:

$$O(p + p^2 + 2 * p * \ln p + (2 * p + \ln p) * \ln(\ln p))$$

Come è facile intuire non è immediatamente riconducibile ad una delle classi di complessità note, ma con un calcolo più specifico relativo alla ricorsione delle iterazioni è possibile ricavare una forma confrontabile con la complessità teorica al fine di stimare il miglioramento.

Calcolo ricorsivo della complessità

Nominando i due insiemi $lasttermset = LT$ e $prevtermset = PT$ e considerando il numero massimo di passate possibile ossia p, che è un'assunzione fortemente difensiva in quanto l'algoritmo con elevatissima probabilità si arresterà molto prima, si ottiene il modello ricorsivo seguente:

$$\begin{cases} i = 2 \dots p \\ |LT_1| = p \\ |PT_1| = p \\ |LT_i| = \ln LT_{i-1} \\ |PT_i| = |PT_{i-1}| + |LT_{i-1}| \\ O_i(|LT_i| * |PT_i|) \end{cases}$$

Dove $O_i(|LT_i| * |PT_i|)$ è la complessità al passo i-esimo.

I passi dal terzo in poi avranno complessità:

$$O_i(|LT_i| * |PT_i|) = O_i(\ln(|LT_{i-2}|) * (|PT_{i-2}| + |LT_{i-2}|)) =$$

$$= O_i\big(\ln(\ln(|LT_{i-3}|)) * (|PT_{i-3}| + |LT_{i-3}| + \ln(|LT_{i-3}|))\big)$$

È evidente che i due fattori hanno delle maggiorazioni relative alla loro grandezza:

$\ln(\ln(|LT_{i-3}|))$ tenderà a decrescere in grandezza e nel caso in cui $LT_{i-3} = LT_1$ si ottiene facilmente il fattore $\ln(\ln p)$ che è maggiorato da $\ln p$, quindi:

$$\ln(\ln(|LT_{i-3}|)) < \ln p$$

Il secondo fattore invece è leggermente più complicato perché rappresenta $prevtermset$ che ad ogni iterazione tenderà a crescere seppur di una quantità sempre più piccola, sapendo che $|PT_1| = p$ e che al massimo si hanno p iterazioni è chiaro che ad ogni iterazione al massimo può aggiungersi una quantità grande p, anche se già dalla terza iterazione in poi viene aggiunto $prevtermset$ che è grande $\ln p$. Conservativamente però si maggiora la quantità con:

$$|PT_{i-3}| + |PT_{i-3}| + \ln(|LT_{i-3}|) < p^2$$

La complessità globale su p passate pertanto sarà:

$$O\big(p + p^2 + (p-2) * (\mathrm{p}^2 * \ln p)\big) = O(p + p^2 + p^3 * \ln p - 2 * p^2 * \ln p)$$

Che è asintoticamente uguale a:

$$O(p^3 * \ln p)$$

Questa stima della complessità è fortemente pessimistica per via di tutte le considerazioni conservative fatte, tuttavia è di complessità polinomiale-logaritmica che rispetto ad una complessità esponenziale è un ordine di complessità inferiore e rispetto all'esecuzione concreta dell'algoritmo che genera un dizionario di decine di migliaia di termini: $|T| = p \approx k * 10^5$ si può immaginare quanto sia il guadagno tra una complessità:

$$O(2^p) \approx O\big(2^{k*10^5}\big)$$

E un'altra:

$$O(p^3 * \ln p) \approx O((k * 10^5)^3 * \ln(k * 10^5))$$

Complessità che nel primo caso coincide con quella reale di un ipotetico algoritmo che esplora tutte le possibili combinazioni, nel secondo caso è parecchio sovrastimata

soprattutto perché si è quasi certi che l'algoritmo non impieghi tutte le p passate assunte come ipotesi per tale stima.

Va aggiunta quindi la complessità della fase di scrematura pervenendo ad una complessità finale:

$$O\left(p^3 * \ln p + c + m * (2 * c - 1)\right)$$

3.11 Estensione dell'algoritmo base: ExhaustiveSets+Meta

L'algoritmo appena visto si basa sulla costruzione degli insiemi di termini per effettuare una ricerca dei possibili insiemi di testi da raggruppare in classi.

Molti degli algoritmi visti nel Capitolo 2 contemplano la presenza preventiva di dati relativi alla classificazione ossia, come indicato nella Tabella 1, dati relativi al contesto. Rispetto all'algoritmo base vengono forniti dei metadata relativi al contesto ossia per ogni classe entro cui classificare sono forniti i termini di riferimento.

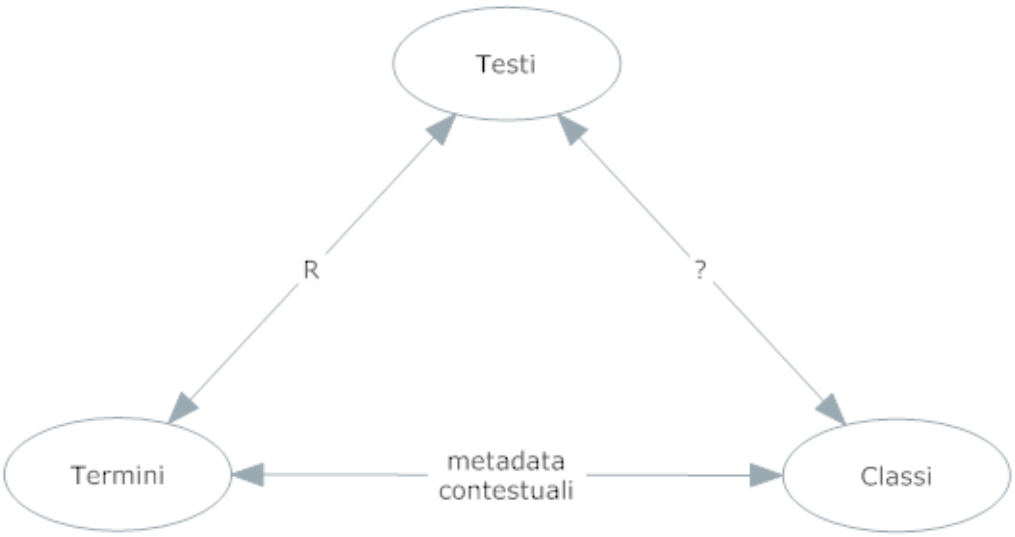

Figura 3-9 Relazione tra testi-termini-classi

Le tre relazioni che in Figura 3-9 sono rappresentate dalle frecce con doppia punta, nell'algoritmo sono delle matrici, in particolare: la matrice di rilevanza R tra termini e testi esposta in Figura 3-4, la matrice di attinenza A tra testi e classi e la matrice dei pesi P tra termini e classi esposte di seguito.

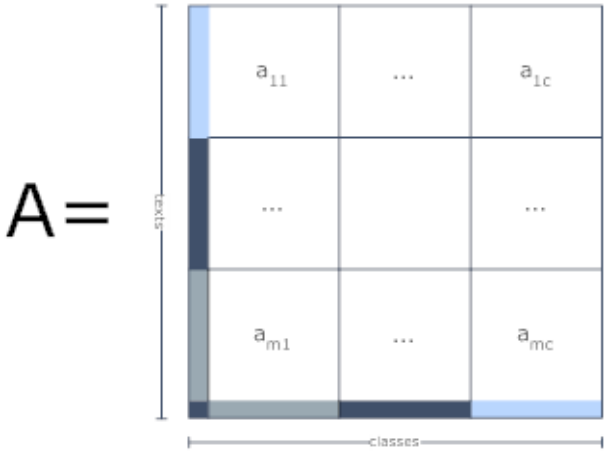

Figura 3-10 Matrice di Attinenza

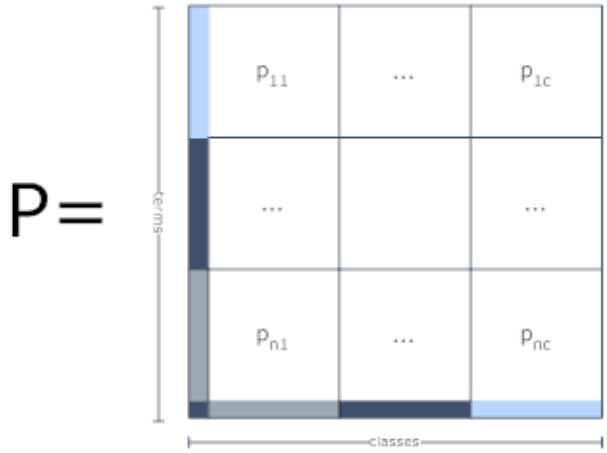

Figura 3-11 Matrice dei Pesi

3.11.1 Pseudocodice ExhaustiveSets+Meta

L'algoritmo implementato è di tipo greedy, esso realizza apprendimento affinando i termini associati alle classi e utilizzando le informazioni apprese per la classificazione di nuovi testi.

```
Function ExhaustiveSets+Meta(R,T,D,C,INCLUSIONPERCENTAGE)
{
        P:=zeros(n,c)
        A:=zeros(m,c)
        EXCLUSIONPERCENTAGE:=INCLUSIONPERCENTAGE*0.9

        for t(i) in T
        {
                p(i):=max(R(i,:))
                delta(i):=p(i)/m
```

```
        h(i):=INCLUSIONPERCENTAGE*p(i)
        e(i):=EXCLUSIONPERCENTAGE*p(i)
}

for c(j) in C
{
        T(c(j)):=terms(c(j))

        for t(i) in T(c(j))
                P(i,j):=p(i)
}

for c(j) in C
{
        D(c(j)):={d(k) in D t.c. R(i,k)>0 & t(i) in T(c(j))}

        for d(k) in D(c(j))
        {
                for t(i) in terms(d(k)) intersect T(c(j))
                {
                        A(k,j):=A(k,j)+R(i,k)
                        P(i,j):=P(i,j)+delta(i)
                }

                for t(i) in terms(d(k)) - T(c(j))
                {
                        P(i,j):=P(i,j)+delta(i)

                        if P(i,j)>h(i)
                        {
                                T(c(j)):=T(c(j)) union {t(i)}
                                D(c(j)):=D(c(j)) union docset(t(i))
                        }
                }

                for t(i) in T(c(j)) - terms(d(k))
                {
                        P(i,j):=P(i,j)-delta(i)
```

```
                                    if P(i,j)<e(i)
                                    {
                                            T(c(j)):=T(c(j)) - {t(i)}
                                            D(c(j)):={d(k) in D t.c. R(i,k)>0 & t(i) in T(c(j))}
                                    }
                            }
                    }
            }
}
```

All'interno del codice si usano le variabili e le strutture dati:

1. **Soglia di inclusione**: espressa dal valore $INCLUSIONPERCENTAGE$, è un parametro che determinerà la soglia sopra la quale un termine assume rilevanza nella classificazione di testi per una data classe.
2. **Soglia di esclusione**: consiste in un valore che è proporzionale a
   $$EXCLUSIONPERCENTAGE = INCLUSIONPERCENTAGE * 0.9$$
   Sotto tale soglia un termine perderà rilevanza per la classificazione in una data classe e verrà rimosso.
3. **Matrice dei pesi**: P inizializzata alla matrice nulla.
4. **Matrice di attinenza**: A inizializzata alla matrice nulla.
5. **Vettore dei pesi iniziali**: $\underline{p}$ rappresenta il peso iniziale di ogni termine all'interno della matrice dei pesi, in particolare $\underline{p_i}$ è il peso iniziale relativo al termine t_i ed è il suo valore di rilevanza massimo all'interno della matrice R:
   $$\underline{p_i} = \left\| \underline{r_{i,:}} \right\|_\infty$$
6. **Vettore dei rinforzi**: $\underline{\Delta}$ contiene i rinforzi positivi o negativi relativi ai termini. Dal momento che un termine verrà valutato su m testi per ogni classe si è deciso di inizializzare questo vettore con quantità pari a:
   $$\underline{\Delta_i} = \frac{p_i}{m}$$
   In modo tale che il peso di un termine all'interno di una classe varierà nel dominio:
   $$P_{i,j} = [0, 2 * p_i]$$

7. **Vettore delle soglie**: $\underline{h}$ contiene le soglie che determinano l'inclusione di un termine nell'insieme di termini rilevanti per una data classe, i valori contenuti dipendono dalla percentuale scelta.
8. **Funzione di estrazione dei termini**: la funzione $terms(\dots)$ dato un testo o una classe estrae i termini contenuti o di riferimento.
9. **Funzione di estrazione di testi**: la funzione $docset(\dots)$ dato un termine estrae i testi che lo contengono.

L'algoritmo quindi procede innanzi tutto inizializzando la matrice dei pesi P associando il valore iniziale a tutti i termini afferenti ad una data classe passati come metadata:

```
for c(j) in C
{
        T(c(j)):=terms(c(j))

        for t(i) in T(c(j))
                P(i,j):=p(i)
}
```

Poi per ogni classe estrae l'elenco dei documenti che contengono almeno un termine associato ad essa e aggiorna la matrice di attinenza A secondo la rilevanza di un termine all'interno della classe:

```
A(k,j):=A(k,j)+R(i,k)
```

Mentre l'aggiornamento della matrice dei pesi avviene secondo tre modalità:

1. **Un termine rilevante per una classe è contenuto nel testo**: rinforzo positivo poiché il termine effettivamente opera positivamente per quella classe. Tale rinforzo viene aggiunto ad un valore iniziale diverso da zero in quanto il termine è rilevante per la classe.
2. **Un termine è contenuto nel testo ma non è indicato come rilevante per la classe**: rinforzo positivo, il termine può essere utile per la classe ma non è specificato nei metadata relativi ad essa. Il valore iniziale è zero, significa che raggiungerà il valore soglia solo se compare con buona frequenza in testi in cui compaiono altri termini specificati nei metadata relativi alla classe.

3. **Un termine specificato come rilevante non è contenuto nel testo**: rinforzo negativo poiché è facile che se non compare in molti testi dove sono contenuti gli altri termini avrà una bassa rilevanza per quella classe e dovrà essere escluso.

In seguito in base al superamento del valore soglia di inclusione si includono nuovi termini e i relativi testi:

```
if P(i,j)>h(i)
{
        T(c(j)):=T(c(j)) union {t(i)}
        D(c(j)):=D(c(j)) union docset(t(i))
}
```

o si escludono i termini e i testi che contengono solo i termini esclusi che quindi hanno una rilevanza per la classe inferiore alla soglia di esclusione:

```
if P(i,j)<e(i)
{
        T(c(j)):=T(c(j)) - {t(i)}
        D(c(j)):={d(k) in D t.c. R(i,k)>0 & t(i) in T(c(j))}
}
```

3.11.2 Caratteristiche generali

Per quanto riguarda le caratteristiche generali dell'algoritmo si nota che:

1. **Utilizzo dei metadata**: uso le informazioni aggiuntive a disposizione relative ai metadata (classi e termini di riferimento) per migliorare la puntualità della classificazione.
2. **Disponibilità di strutture dati efficienti**: ho a disposizione strutture dati che permettono, con costo costante $O(1)$ sia l'estrazione dei termini contenuti in un testo che l'estrazione dei testi che contengono un dato termine.
3. **Assegnamento dei testi alle classi**: anziché generare nuove classi assegno direttamente i testi alle classi che ho già a disposizione con il relativo valore di attinenza tra un testo e una classe.

4. **Uso di formule logiche**: è possibile in futuri lavori introdurre l'uso di formule logiche in formato disgiunto che esprimono l'esistenza o meno di un termine all'interno di un testo:

$$t_i \vee (\sim t_{i+1}) \vee t_{i+2}$$

5. **Feedback**: il feedback consiste nella modifica della matrice dei pesi tra termini e classi e nel successivo confronto con un valore soglia. I termini che sono contenuti nel testo otterranno un rinforzo positivo, i termini di riferimento della classe ma che non sono presenti nel testo corrente otterranno anche loro un rinforzo positivo. Se un termine supera la soglia di inclusione allora verrà incluso tra i termini di riferimento per quella classe e quindi inclusi i testi che lo contengono. Se un termine scende sotto alla soglia di esclusione allora verrà rimosso dai termini di riferimento per quella classe e i testi che contengono solo quel termine verranno esclusi dalla computazione.

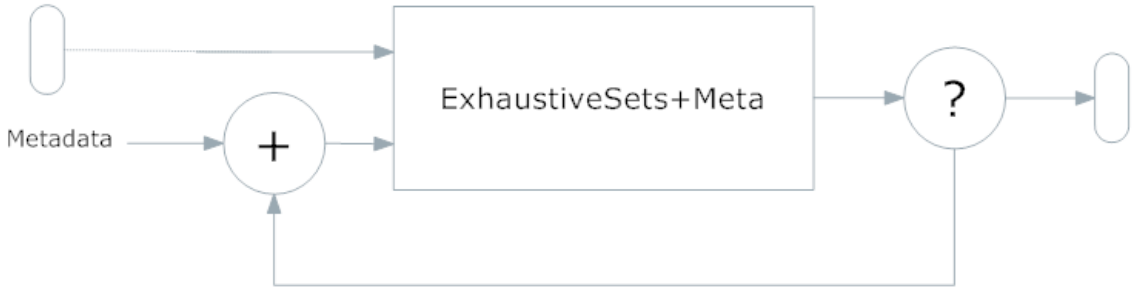

Figura 3-12 ExhaustiveSets+Meta con feedback

6. **Uso della doppia soglia**: per evitare un fenomeno di "sfarfallio" ossia di inclusione ed esclusione repentina di termini che hanno rilevanza per una data classe vicino ad una eventuale soglia singola si fa uso della doppia soglia.
   Le due soglie individuate sono proporzionali alla rilevanza massima di un termine e in particolare sono:

$$h_i = INCLUSIONPERCENTAGE * \left\| \underline{r_{i,:}} \right\|_\infty$$

$$e_i = INCLUSIONPERCENTAGE * 0.9 * \left\| \underline{r_{i,:}} \right\|_\infty$$

   Si attiva pertanto un effetto stabilizzante simile al modello di isteresi.

7. **Velocità di analisi**: non è più necessario generare tutti i possibili $termset$ al fine di individuare classi più rilevanti per i testi. Fornendo preventivamente i metadati per classificazione ci si pone l'obiettivo di ottenere velocità di analisi elevatissime.

8. **Necessità di scremare termini frequenti**: dal momento che l'algoritmo apprende sulla base dell'esistenza o meno di un termine è fondamentale scremare le stop-word che compaiono frequentemente nei testi forniti. In caso contrario è analiticamente evidente come l'algoritmo a lungo termine tenda a classificare in base alle stop-word che vengono apprese più facilmente di altri termini di interesse.

3.11.3 Scrematura delle classi

Dopo aver eseguito l'algoritmo si ottengono come risultati le associazioni tra testi e classi. Si ottiene quindi un insieme di valori di attinenza tra queste entità che è possibile esprimere in una matrice dove sulle righe vi sono i testi e sulle colonne le classi.

Da un'analisi relativa alla matrice di attinenza in Figura 3-10 è possibile identificare tre modi per scremare le associazioni tra testi e classi:

1. **Nessuna scrematura**: si mantengono tutte le informazioni circa l'appartenenza dei testi alle classi.
2. **Scrematura parziale**: si identifica una metrica per definire un valore soglia sotto al quale eliminare l'associazione tra testo e classe.
3. **Scrematura completa**: si mantiene solo l'associazione con attinenza massima.

In questo lavoro si è scelto di effettuare una scrematura parziale in modo da eliminare i testi che hanno una attinenza "bassa" con la classe di riferimento.

Si calcola quindi un valore di soglia per la scrematura in base alle attinenze:

$$SKIMMINGTHRESHOLD = \min A + \frac{\left(\|A\|_\infty - \min A\right)}{10}$$

Quindi i testi che afferiscono ad una data classe con attinenza inferiore a tale soglia vengono rimossi dalla classe, in pseudocodice:

```
SKIMMINGTHRESHOLD:=min(A)+(max(A)-min(A))*0.1

for d(k) in D
        for c(j) in C
                if A(k,j)<SKIMMINGTHRESHOLD
                        A(k,j):=0
```

Questa fase consente di migliorare la metrica di precision ma c'è il rischio che peggiori leggermente la metrica di recall.

3.11.4 Complessità ExhaustiveSets+Meta

Per quanto riguarda la complessità bisogna fare prima di tutto due considerazioni:

1. **Estrazioni a costo costante**: il recupero di termini e testi avviene con costo costante in quanto il software contiene strutture dati create e aggiornate appositamente che permettono queste operazioni, quindi le istruzioni:

   - D(c(j)):={d(k) in D t.c. R(i,k)>0 & t(i) in T(c(j))}
   - terms(d(k))
   - docset(t(i))

   Costano tutte $O(1)$.
2. **Grandezze logaritmiche**: in merito alle dimensioni degli insiemi bisogna considerare il fatto che partendo dalla matrice di rilevanza R che è sparsa si usano le maggiorazioni viste per l'algoritmo base ExhaustiveSets. Quindi $\ln(n)$ per quanto riguarda la grandezza degli insiemi di termini di un testo o ad una classe, con $n = |T|$. E $\ln(m)$ per quanto riguarda i testi che contengono i termini di una data classe.

Analizzando il codice quindi si ha che il ciclo iniziale:

```
for t(i) in T
{
        p(i):=max(R(i,:))
        delta(i):=p(i)/m
        h(i):=INCLUSIONPERCENTAGE*p(i)
        e(i):=EXCLUSIONPERCENTAGE*0.9*p(i)
}
```

Ha un costo di n.

Il ciclo che assegna i pesi nella relativa matrice:

```
for c(j) in C
{
        T(c(j)):=terms(c(j))

        for t(i) in T(c(j))
                P(i,j):=p(i)
}
```

Costerà $O(c * \ln(n))$ con $c = |C|$.

Mentre la fase di classificazione e apprendimento contiene i tre cicli di aggiornamento dei pesi e calcolo dell'attinenza tra testi e classi:

```
for t(i) in terms(d(k)) intersect T(c(j))
{
        A(k,j):=A(k,j)+R(i,k)
        P(i,j):=P(i,j)+delta(i)
}

for t(i) in terms(d(k)) - T(c(j))
{
        P(i,j):=P(i,j)+delta(i)

        if P(i,j)>h(i)
        {
                T(c(j)):=T(c(j)) union {t(i)}
                D(c(j)):=D(c(j)) union docset(t(i))
        }
```

```
}

for t(i) in T(c(j)) - terms(d(k))
{
        P(i,j):=P(i,j)-delta(i)

        if P(i,j)<e(i)
        {
                T(c(j)):=T(c(j)) - {t(i)}
                D(c(j)):={d(k) in D t.c. R(i,k)>0 & t(i) in T(c(j))}
        }
}
```

Costeranno ciascuno $O(\ln(n))$ quindi in totale $O(3 * \ln(n))$ mentre le istruzioni:

```
D(c(j)):={d(k) in D t.c. R(i,k)>0 & t(i) in T(c(j))}

for d(k) in D(c(j))
        ...
```

Determineranno una complessità di $O(\ln(m))$.

Si suppone inoltre che le inclusioni/esclusioni di testi dall'elenco di testi da analizzare per effetto dell'apprendimento di nuovi termini tendano numericamente ad annullarsi le une con le altre. Questa assunzione è valida dal momento che l'insieme di termini contenuti nei testi T è finito. Inoltre è supportata da valutazioni empiriche. Pertanto nel calcolo della complessità si può ignorare la crescita/decrescita dei testi da analizzare.

E infine vi è un ciclo che opera su tutte le classi che costerà $O(c)$.

La formula della complessità totale sarà quindi:

$$O(n + c * \ln(n) + c * \ln(m) * 3 * \ln(n))$$

Che asintoticamente tenderà a:

$$O(c * \ln(m) * \ln(n))$$

Supponendo che $m \gg n \gg c$ che è un risultato di tutto rispetto considerando che si hanno a disposizione i metadati rispetto alle classi e si è adottato un approccio greedy.

3.11.5 Scelta delle soglie di apprendimento

L'algoritmo esposto fa uso di due soglie per l'apprendimento di termini di riferimento per una data classe. Le soglie sono dipendenti tra loro quindi sono identificabili da un singolo parametro. Non è possibile sapere quale sia il valore che garantisce prestazioni migliori.

Questo dipende da due fattori:

1. **Dimensione dell'insieme di testi in ingresso**: per valori bassi della soglia $INCUSIONPERCENTAGE \cong 0$ un insieme di testi molto numeroso potrebbe portare ad una esclusione o inclusione continua di termini in quando dopo l'analisi di pochi testi un termine potrebbe venire incluso e escluso portando l'algoritmo ad essere molto instabile.
   Per valori alti della soglia $INCLUSIONPERCENTAGE \cong 1$ l'algoritmo tenderà ad apprendere pochi termini nuovi quindi non riuscire a migliorare le prestazioni su insiemi di testi piccoli. In particolare per $INCLUSIONPERCENTAGE = 1$ l'algoritmo non apprenderà alcun termine nuovo.
2. **Numero di termini di riferimento per le classi**: una classe a cui all'interno dei metadata sono associati pochi termini tenderà a modificare poco i pesi associati tra termini e classi degli altri termini, rendendo difficile l'apprendimento di eventuali termini. Questo farà preferire una soglia bassa $INCLUSIONPERCENTAGE \cong 0$.
   Viceversa una classe con tantissimi termini tenderà a modificare molto i pesi relativi agli altri termini rendendo l'apprendimento fin troppo invasivo.

Quello che da questa analisi è chiaro è che non è possibile definire un valore di soglia analiticamente. I test relativi alla qualità della classificazione pertanto vanno ripetuti per valori di soglia diversi. Per quanto riguarda le prestazioni esse non varieranno a prescindere dal valore della soglia.

Si effettueranno pertanto dei test per valori di soglia:

$$INCLUSIONPERCENTAGE \in \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$$

Facendo poi a posteriori considerazioni circa la qualità della classificazione unificando i risultati.

3.12 Modello di misura delle metriche e valutazione qualitativa

Le metriche da definire per il confronto e l'analisi dei risultati ottenuti devono essere in grado di dare una valutazione esaustiva e completa della problematica. Si vogliono quindi definire delle metriche per la valutazione degli obiettivi esposti nel paragrafo 3.1.

Nei seguenti sottoparagrafi verranno quindi esposte due tipi di metriche: le prime con finalità relative alla valutazione della qualità degli algoritmi le seconde in grado di valutare le prestazioni degli algoritmi, che è una questione centrale.

3.12.1 Misura dell'efficienza: qualità

Le tecniche usate in letteratura per la misura dell'efficienza degli algoritmi di TC sono mutuate dai lavori riguardo IR. Con i dovuti aggiustamenti le misure di qualità degli algoritmi di TC ha alla base le misure relative agli algoritmo di IR.

In (30) viene data una prima precisa definizione riguardo alla differenza tra la valutazione di un sistema di IR rispetto ad un sistema di TC, in particolare le misure di bontà di IR si basano sul paradigma query-insieme di risposta, si misura quindi quanto i testi forniti come risposta dal sistema rispondano in maniera corretta alla query espressa. In TC invece si hanno delle classi e l'obiettivo è misurare quanto gli assegnamenti testo-classe siano corretti.

Bisogna tenere conto, come ampiamente esposto in (30), anche del contesto in cui opera l'algoritmo di TC oggetto di misurazione, ossia quali siano gli effetti indiretti di una buona o cattiva classificazione sul sistema esterno che fa uso di tali dati.

3.12.1.1 Contingency Table

In (30) viene fornita la cosiddetta tabella di contingenza:

	Risposta corretta: appartiene	**Risposta corretta: non appartiene**
Decisione presa: appartiene	a	b (errore seconda specie)
Decisione presa: non appartiene	c (errore prima specie)	d

Tabella 3 Contingency table

Data una specifica classe e un test-set relativo è possibile contare il numero di testi classificati correttamente o non correttamente circa l'appartenenza o meno ad una classe. In questo senso è possibile ottenere quattro valori numerici:

$$a, b, c, d \in \mathbb{N}^+$$

In particolare il valore di b rappresenta un falso positivo che tenderà a peggiorare la metrica di precision, mentre il valore di c rappresenterà un falso negativo che tenderà a peggiorare la metrica di recall.

3.12.1.2 Metriche disaggregate

Da questi quindi sono calcolabili le seguenti metriche per ogni classe $c \in C$, con C insieme delle classi:

1. **Precision**: è il numero di testi classificati correttamente tra quelli identificati come appartenenti
$$P_c = \frac{a}{a+b} \in [0,1]$$
2. **Recall**: è il numero di testi classificati correttamente tra quelli che effettivamente appartengono alla classe
$$R_c = \frac{a}{a+c} \in [0,1]$$

3. **Fallout**: è il numero di testi non classificati correttamente dati come appartenenti ad una classe mentre in realtà non lo sono, è il duale della Recall

$$N_c = \frac{b}{b+d} \in [0,1]$$

4. **Overlap**: viene usata come confronto diretto tra due algoritmi

$$O_c = \frac{a}{a+b+c} \in [0,1]$$

5. **Accuracy**: rappresenta le decisioni corrette prese per una data classe

$$A_c = \frac{a+d}{a+b+c+d} \in [0,1]$$

6. **Error**: è duale rispetto l'Accuracy, sono cioè le decisioni non corrette

$$E_c = \frac{b+c}{a+b+c+d} = 1 - A_c \in [0,1]$$

7. **F-α-measure**: fornisce una misura bilanciata da un parametro $\alpha \in [0,1]$ tra Precision e Recall

$$F_{\alpha c} = \frac{P_c * R_c}{\alpha * R_c + (1-\alpha) * P_c} \in [0,1]$$

Dove $\alpha = 0$ fa coincidere questa misura con la misura di Recall, $\alpha = 1$ fa coincidere questa misura con la misura di Precision. In questo lavoro si è considerato F-α-measure con $\alpha = \frac{1}{2}$ ossia un ugual peso tra Precision e Recall.

8. **F-β-measure**: è una formula simile a F-α-measure e del tutto uguale come finalità, solo che considera valori di $\beta \in [0, +\infty)$

$$F_{\beta c} = \frac{(\beta^2 + 1) * P_c * R_c}{\beta^2 * P_c + R_c} \in [0,1]$$

L'equilibrio tra le due misure si ottiene per $\beta = 1$. Questa misura presenta delle non-linearità in quanto un valore di $\beta = +\infty$ non è trattabile computazionalmente. Per questo motivo si preferisce F-α-measure.

Bisogna quindi aggregare le misure fatte per ogni singola classe, in tutti i testi che trattano questa problematica viene fatta la distinzione tra macroaveraging e microaveraging: la prima misura media tutte le decisioni prese globalmente (a, b, c e d) calcolando i singoli indici sull'intero spazio classi-testi, in (30) viene indicato come questo metodo meglio si adatta a sistemi di IR. La seconda misura media i parametri

calcolati ottenuti per ogni singola classe a posteriori. In sostanza come indicato in (10) microaveraging fornisce un ugual peso a tutti i testi mentre macroaveraging fornisce ugual peso a tutte le classi.

3.12.1.3 Metriche aggregate

In questo lavoro si è preferito un approccio microaveraging poiché oltre a fornire risultati specifici per ogni singola classe pesa meglio le difficoltà dei vari algoritmi sulle singole classi aggregando subito i dati ad un livello basso. In formule per ogni algoritmo verranno calcolati del valori globali mediando i singoli valori calcolati con la formula relativa su tutte le classi:

1. **Precision globale**:

$$P = \frac{\sum_{c \in C} P_c}{|C|} \in [0,1]$$

2. **Recall globale**:

$$R = \frac{\sum_{c \in C} R_c}{|C|} \in [0,1]$$

3. **Fallout globale**:

$$N = \frac{\sum_{c \in C} F_c}{|C|} \in [0,1]$$

4. **Overlap globale**:

$$O = \frac{\sum_{c} O_c}{|C|} \in [0,1]$$

5. **Error globale**:

$$E = \frac{\sum_{c \in C} E_c}{|C|} \in [0,1]$$

6. **F-α-measure globale**:

$$F_\alpha = \frac{\sum_{c \in C} F_{\alpha c}}{|C|} \in [0,1]$$

Riguardo a queste misure va considerato che la bontà di un algoritmo sta nel fornire buoni risultati su tutte le misure perché come indicato in (10) un algoritmo che assegna tutti i testi a tutte le classi avrà una Recall massima ma una Precisione minima, il che non è indice di bontà, per questo si considerano con più attendibilità misure combinate.

3.12.1.4 Considerazioni sulla generalità delle metriche scelte

A lato di queste misure vi sono delle considerazioni da fare: prima di tutto quanto pesa un errore di prima o seconda specie? Questione non affrontata in questo lavoro ma spunto di una nuova rivisitazione soprattutto in un'ottica di valutazione del contesto in cui va ad operare l'algoritmo di TC proposto.

In quegli algoritmi parametrizzabili tramite un valore reale è possibile variare in maniera più o meno libera i valori qualitativi. È quindi possibile definire un break-even point ossia il punto in cui un algoritmo presenta valori di precision e recall equivalenti. Sperimentalmente quindi è possibile ottenere la configurazione migliore relativamente all'efficienza di questi algoritmi.

Infine, estremizzando, non è a priori valido considerare la classificazione di riferimento (test-set) fatta manualmente da persone assolutamente corretta, a tal fine risulta interessante valutare una classificazione ottenuta incrociando più algoritmi soprattutto nei casi di forti discrepanze.

3.12.2 Misura delle prestazioni: profiling

Il modello costruito per la misura delle prestazioni si basa su due aspetti ortogonali tra loro: la misura delle prestazioni (profiling) e i task di interesse.

Volendo fare un breve riassunto sull'attività di profiling è possibile partire da (31). L'attività di profiling è chiamata anche "dynamic program analysis", si tratta cioè della misura delle prestazioni di un software a runtime, ossia basandosi sulle attività che questo svolge durante la sua esecuzione. Al contrario dell'analisi statica che si occupa dell'ottimizzazione del codice sorgente prima della compilazione o in fasi intermedie di essa.

Il profiling misura principalmente tre caratteristiche di un dato software:

1. **Uso della memoria**: quantità di memoria principale usata ed eventuale grandezza di spazio di swap. Vengono di solito forniti dei dati medi e uno storico con la relativa occupazione di memoria.
2. **Uso di istruzioni particolari**: vengono misurate le prestazioni relative a particolari istruzioni, la loro frequenza, le tempistiche e quali unità operative sono coinvolte nell'esecuzione.

3. **Frequenza e durata di chiamate a funzioni**: vengono misurate le prestazioni di determinati compiti implementati tramite specifiche funzioni.

Per ottenere tali misure si usano software o librerie che rientrano nella categoria dei code profiler, essi possono fornire: sommari statistici, sequenza di eventi registrati o statistiche ottenute dal supervisore del sistema operativo (SO).

I code profiler possono adottare vari metodi per raccogliere le informazioni riguardo al software analizzato: in base ad eventi sollevati nel SO all'esecuzione di una data istruzione o a intervalli regolari circa lo stato del program counter in modo da ottenere dati parzialmente accurati ma statisticamente validi. Tramite "instrumentation" ossia con l'utilizzo di librerie che forniscono API in grado di effettuare misure a runtime anche se questa tecnica è chiaramente invasiva rispetto al codice sorgente e "falsa" i tempi di esecuzione reali del software ma fornisce dati molto accurati. Oppure tramite interprete, eseguendo cioè il software all'interno di una macchina virtuale che rileva i dati senza "invadere" il codice sorgente.

3.12.2.1 Caratteristiche del sistema di test

Quando si svolge un'attività di profiling è fondamentale specificare le caratteristiche del sistema di test, tali caratteristiche includono le specifiche riguardo a:

1. **Hardware**: vengono fornite informazioni relative alla CPU, la RAM e al disco che compongono il sistema. Queste informazioni sono fondamentali in quanto forniscono un metro di paragone per l'esecuzione del software su sistemi più o meno performanti.
2. **Sistema operativo**: è utile conoscere anche le caratteristiche del SO in quanto nelle prestazioni dell'esecuzione di un programma influiscono lo scheduling adottato e la relativa priorità del processo, o dei processi, associato al software e l'allocazione delle risorse che nei moderni SO è eseguita da un modulo del SO stesso.
3. **Java® Virtual Machine**: il software di questo lavoro di tesi è implementato in linguaggio Java, pertanto è utile conoscere le caratteristiche della JVM che esegue tale codice.

4. **Processi concorrenti in esecuzione**: i processi concorrenti all'esecuzione del software oggetto di profiling influiscono soprattutto per quanto riguarda le tempistiche per via dell'accesso alle risorse. Così come per l'hardware questo dato può essere utile per ottenere un metro di confronto per l'esecuzione del software su sistemi più o meno carichi.
5. **Codice da eseguire**: conoscere il codice da eseguire è fondamentale in quanto si può avere un'idea di come la complessità dell'algoritmo influisca nei tempi e nelle risorse. Eventualmente è possibile ottimizzare l'esecuzione agendo sulle parti maggiormente onerose.
6. **Codice di ispezione**: infine dal momento che per eseguire un'attività di profiling è necessario avvalersi di un code profiler, come indicato nel paragrafo 3.12.2, bisogna specificarne le caratteristiche. In questo lavoro ci si è basati su un software di profiling esterno InspectIT® e si è creato un package che racchiude le classi necessarie al profiling "instrumentation" ottenute usando la libreria Java nativa: Java Virtual Machine Tools Interface, JVMTI.

3.12.2.2 Metriche prestazionali

La concentrazione è stata rivolta ai tempi di esecuzione della varie fasi che interessano la classificazione.

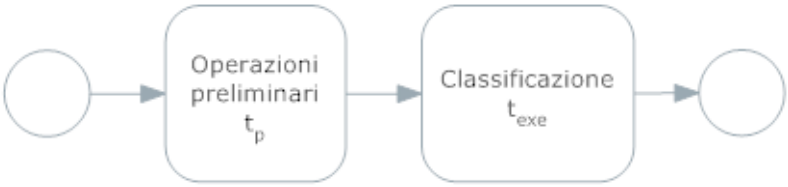

Figura 3-13 Macrostruttura Algoritmo

Le metriche misurate in questo lavoro sono esposte di seguito:

1. **Tempo speso per operazioni preliminari**: con riferimento alla Figura 3-13 è il tempo necessario ad eseguire le operazioni di parsing e riduzione delle dimensionalità, quindi fino a valle delle operazioni di riduzione tramite term weighting, vedi Figura 3-2. Ci si aspetta che questo parametro sia uguale per tutti gli algoritmi entro un certo ordine di grandezza visto l'inevitabile overhead dato dallo scheduling del SO:

$$t_p$$

Questa quantità sarà data dalla somma dei tempi della fase parsing (t_{par}) dei testi con le relative riduzioni e della fase di calcolo della rilevanza con eventuale riduzione delle dimensioni in base a pesatura (t_{rel}):

$$t_p = t_{par} + t_{rel}$$

2. **Tempo di esecuzione**: è la misura del tempo di esecuzione degli algoritmi di classificazione, dato che essi presentano una base comune misurata con la metrica precedente è stato misurato solamente il tempo necessario alla classificazione vera e propria:

$$t_{exe}$$

Si può quindi valutare l'impatto delle riduzioni di dimensioni sul tempo di esecuzione degli algoritmi di classificazione.

3.12.3 Insiemi di dati

Gli insiemi di dati presenti che sono stati considerati per le misurazioni esposte sopra sono:

1. **20Newgroup**: consiste in approssimativamente 20000 testi provenienti da newsgroup divisi in training e test set. I dati relativi alle classi di appartenenza sono l'oggetto della e-mail, è chiaro però che le classi vanno ulteriormente aggregate in base al significato dei vari oggetti.
2. **Cranfield**: questo dataset è composto da 1400 estratti di circa 20 righe ciascuno di testi scientifici e tecnici dell'università Cranfield. Esso fornisce anche informazioni relative ai risultati corretti forniti da questo dataset in base a delle query.
3. **Ohsumed**: è un dataset di tipo medico che consiste in 348566 testi medici estratti da 270 giornali medici tra il 1987 e il 1991. Anche questo insieme di dati, al pari di Cranfield, si basa sull'interazione tramite query, pertanto sono disponibili informazioni circa i risultati corretti a fronte di query fornite.
4. **Reuters-21578**: questo insieme di dati è composto da 21578 estratti di articoli Reuters del 1987, molti di essi sono stati organizzati in categorie da personale Reuters. Inoltre l'intero dataset è formattato in XML e viene fornito anche il doctype di riferimento.

5. **Torcia**: questo dataset consiste in un insieme di tweet in italiano e classificati relativi alle emergenze usati all'interno del progetto Torcia. Questo insieme di dati è composto da due tipi di files. Il primo contiene dati relativi all'associazione di termini alle classi, il secondo contiene dei tweet estratti con le relative classi di appartenenza.

A valle di questa rassegna circa i dataset più comuni disponibili si è deciso di fare riferimento al dataset Reuters-21578 e Torcia.

Il primo oltre a fornire dati formattati esprime chiaramente le categorie di appartenenza con dei topic piuttosto che delle query che non permettono una buona aggregazione. Questo dataset è stato usato per il confronto tra gli algoritmi k-NN e LSI con ExhaustiveSets, per poter validare quindi quest'ultimo algoritmo su testi brevi.

Mentre il secondo esprimendo anche i termini afferenti ad una classe è utile per validare l'estensione dell'algoritmo base ExhaustiveSets+Meta.

Capitolo 4 **Analisi e Risultati**

In questo capitolo viene esposto il lavoro effettuato per la misura e il confronto delle funzionalità. Dapprima viene illustrato come sono stati implementati gli algoritmi di confronto individuati nel paragrafo 3.1 con un occhio ai paragrafi 2.3.1.2 e 2.3.2.2 dove sono presenti i concetti teorici.

In seguito viene spiegato come sono state implementate le misure delle metriche esposte nel paragrafo 3.12 e quindi esposti i risultati ottenuti. Infine vengono analizzati i risultati.

4.1 Implementazione degli algoritmi di confronto

Gli algoritmi da confrontare con ExhaustiveSets sono k-NN e LSI, di seguito vengono esposti i concetti pratici della loro implementazione, per i riferimenti teorici è possibile consultare i paragrafi 2.3.1.2 per k-NN e 2.3.2.2 per LSI.

4.1.1 k-NN

La prima problematica nell'implementazione dell'algoritmo k-NN è la costruzione delle classi iniziali dato un set di testi. Si tratta sostanzialmente del calcolo della matrice delle distanze e nella clusterizzazione seguente.

Partendo dalla definizione di classe riportata nel paragrafo 2.3.1.2:

$$c_j = \left\{\underline{d} \in D\ t.c. \forall \underline{d_j} \in D \wedge \forall \hat{\underline{d}} \in D \wedge classe\left(\underline{d_j}\right) = c_j \wedge classe\left(\hat{\underline{d}}\right) \neq c_j \wedge dist\left(\underline{d}, \underline{d_j}\right) < dist\left(\underline{d}, \hat{\underline{d}}\right)\right\}$$

$$c_1 \cap c_2 \cap \ldots \cap c_n = \{\quad\}$$

$$C = c_1 \cup c_2 \cup \ldots \cup c_n$$

Un testo appartenente ad una classe c_j ha una distanza rispetto agli altri testi in c_j inferiore a testi di classe diversa da c_j. La stima della classe di un generico testo $\underline{d}$ avviene valutando la classe di appartenenza dei $k \in \mathbb{N} - \{0\}$ testi più vicini a $\underline{d}$ pesata con la distanza tra $\underline{d}$ e tali testi.

Si vuole porre attenzione particolare all'approccio usato da questo metodo: esso è focalizzato sui testi piuttosto che sui termini, dato che l'insieme dei testi tenderà ad essere più grande rispetto a quello dei termini questo metodo tenderà ad avere prestazioni meno performanti rispetto a quello di confronto, ExhaustiveSets, che invece opera sui termini.

Assume quindi un'importanza centrale la metrica usata per il calcolo della distanza, in (32) viene riportata un'esaustiva trattazione, in particolare riportando quanto esposto per vettori in $\mathbb{R}^n$, una distanza tra due vettori è una funzione che associa:

$$d\left(\underline{x}, \underline{y}\right) = d, \underline{x}, \underline{y} \in \mathbb{R}^n$$

$$\mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$$

Che soddisfa le seguenti proprietà:

1. $d\left(\underline{x},\underline{y}\right) \geq 0$
2. $d\left(\underline{x},\underline{y}\right) = 0 \Rightarrow \underline{x} = \underline{y}$
3. $d\left(\underline{x},\underline{y}\right) = d\left(\underline{y},\underline{x}\right)$
4. $d\left(\underline{x},\underline{y}\right) \leq d(\underline{x},\underline{z}) + d\left(\underline{z},\underline{y}\right), \underline{z} \in \mathbb{R}^n$

Le distanze e coefficienti interessanti disponibili in letteratura sono:

1. **Distanza Euclidea**: $d\left(\underline{x},\underline{y}\right) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$, costo $O(n)$
2. **Distanza Hamming**: $d\left(\underline{x},\underline{y}\right) = \sum_{i=1}^{n}|x_i - y_i|$, costo $O(n)$
3. **Coefficiente di Dice**: $s\left(\underline{x},\underline{y}\right) = 1 - \frac{2*\underline{x}*\underline{y}}{|\underline{x}|^2+|\underline{y}|^2}$, costo $O(3*n)$
4. **Coefficiente coseno**: $s\left(\underline{x},\underline{y}\right) = 1 - \frac{\underline{x}*\underline{y}}{|\underline{x}|*|\underline{y}|}$, costo $O(2*n)$

Le ultime due metriche sono coefficienti, infatti si è usata la notazione $s(\dots)$ che sta per similarità in quanto non possiedono la proprietà 4 di disuguaglianza triangolare.

La distanza migliore è quella Hamming in quanto ci si risparmia il calcolo del quadrato per ogni componente e la radice, tuttavia da considerazioni empiriche circa le prime sperimentazioni si è preferito usare una versione modificata di tale distanza poiché essa non rispondeva in maniera puntuale alla percezione di similarità tra due testi. A titolo di esempio si considerino i due vettori colonna della matrice di rilevanza R rappresentanti due generici testi $\underline{d_1}$ e $\underline{d_2}$:

$$\underline{d_1} = \left(R_{1,1}, R_{2,1}, R_{3,1}, R_{4,1}, R_{5,1}, R_{6,1}\right)$$

$$\underline{d_2} = \left(R_{1,2}, R_{2,2}\right)$$

Essi condividono i termini t_1 e t_2 e la distanza di Hamming fornirà:

$$d\left(\underline{d_1},\underline{d_2}\right) = \left|R_{1,3}\right| + \left|R_{1,4}\right| + \left|R_{1,5}\right| + \left|R_{1,6}\right|$$

Assumendo che, come facilmente avviene:

$$R_{1,1} = R_{1,2} \ \wedge \ R_{2,1} = R_{2,2}$$

Mentre se consideriamo altri due testi:

$$\underline{d_3} = \left(R_{3,3}, R_{4,3}\right)$$

$$\underline{d_4} = \left(R_{5,4}\right)$$

La distanza di Hamming:

$$d\left(\underline{d_3}, \underline{d_4}\right) = \left|R_{3,3}\right| + \left|R_{4,3}\right| + \left|R_{5,4}\right|$$

Ora, come è risultato evidente da prove pratiche può risultare:

$$d\left(\underline{d_1}, \underline{d_2}\right) > d\left(\underline{d_3}, \underline{d_4}\right)$$

Sebbene i testi $\underline{d_1}$ e $\underline{d_2}$ condividono dei termini mentre $\underline{d_3}$ e $\underline{d_4}$ nessun termine risultando quindi estranei tra loro.

Bisogna quindi far si che venga data maggiore importanza ai termini condivisi tra i due testi e meno importanza ai termini non condivisi. Mentre se i testi non condividono alcun termine è necessario maggiorare la distanza. Questo è ottenibile imponendo delle distorsioni non lineari nella formula del calcolo della distanza facendo cadere le proprietà di non linearità e quindi la proprietà di disuguaglianza triangolare, proprietà 4.

La formula proposta quindi diventa:

$$d\left(\underline{d_i}, \underline{d_j}\right) =$$

$$= \begin{cases} \displaystyle\sum_{t_k \ t.c. R_{k,i} \neq 0 \wedge R_{k,j} \neq 0} \left|R_{k,i} - R_{k,j}\right| + \frac{\sum_{t_k \ t.c. R_{k,i} \neq 0 \wedge R_{k,j} = 0} \left|R_{k,i}\right| + \sum_{t_k \ t.c. R_{k,i} = 0 \wedge R_{k,j} \neq 0} \left|R_{k,j}\right|}{n+1} \\ se \ \exists t_k \ t.c. R_{k,i} \neq 0 \wedge R_{k,j} \neq 0 \\ n * \left(\displaystyle\sum_{t_k \ t.c. R_{k,i} \neq 0 \wedge R_{k,j} = 0} \left|R_{k,i}\right| + \sum_{t_k \ t.c. R_{k,i} = 0 \wedge R_{k,j} \neq 0} \left|R_{k,j}\right| \right) \\ se \ \nexists t_k \ t.c. R_{k,i} \neq 0 \wedge R_{k,j} \neq 0 \end{cases}$$

Dove:

$$n = \left|\left\{t_k \; t.c. \left(R_{k,i} \neq 0 \wedge R_{k,j} = 0\right) \vee \left(R_{k,i} = 0 \wedge R_{k,j} \neq 0\right)\right\}\right|$$

Quindi data la matrice di rilevanza R calcolo una matrice delle distanze F, si usa la lettera F, dall'inglese "far", per disambiguare la notazione rispetto al più naturale D "distance" che è il vettore che contiene i testi. Questa matrice sarà triangolare superiore con tutti zeri sulla diagonale principale:

F=

	d_1	d_2	d_3	d_4	d_5	d_6
d_1		#	#	#	#	#
d_2			#	#	#	#
d_3				#	#	#
d_4					#	#
d_5						#
d_6						

Figura 4-1 Matrice delle Distanze k-NN

Il calcolo di questa matrice costerà $O\left(\frac{m*(m-1)}{2}\right)$.

k-NN quindi si comporrà di due fasi:

1. La prima fase dove dato un insieme di testi vengono costruiti dei cluster che individueranno delle classi secondo l'algoritmo esposto di seguito.
2. La seconda fase in cui incrementalmente verranno classificati nuovi testi in base alle classi di appartenenza dei primi k testi più vicini.

4.1.1.1 Inizializzazione k-NN

Una volta calcolata la matrice delle distanze F è possibile creare una classifica ordinata in base alle distanze delle associazioni tra testi, ad esempio:

$$d\left(\underline{d_2}, \underline{d_4}\right)$$
$$d\left(\underline{d_1}, \underline{d_5}\right)$$
$$\vdots$$
$$d\left(\underline{d_1}, \underline{d_2}\right)$$

$$d\left(\underline{d_2}, \underline{d_4}\right) \leq d\left(\underline{d_1}, \underline{d_5}\right) \leq \cdots \leq d\left(\underline{d_1}, \underline{d_2}\right)$$

Quindi scorrendo tale struttura dati dalla distanza più breve a quella più grande se nessuno dei due testi sta già in una classe allora essi formeranno una nuova classe, se uno dei due sta in una classe e l'altro non è stato ancora classificato allora quest'ultimo viene aggiunto. In pseudocodice:

```
Function initk-NN(F)
{
        sortDTexts:=sort(F)
        for i=1…|sortDTexts|
        {
                dx:=sortDTexts[i].dx
                dy:=sortDTexts[i].dy
                cx:=findClass(dx)
                cy:=findClass(dy)
                if cx=null ∧ cy=null
                        generateClass({dx, dy},terms({dx, dy})
                else if cx≠null ∧ cy=null
                        addToClass(dy,cx)
                else if cx=null ∧ cy≠null
                        addToClass(dx,cy)
        }
}
```

Nella funzione sono usate le sotto-funzioni:

1. $sort$ che esegue l'ordinamento della matrice F in ordine crescente di distanza.
2. $findClass$ che trova la classe di appartenenza di un dato testo $\underline{d}$.
3. $addToClass$ che aggiunge un dato testo $\underline{d}$ ad una classe di cui è riportato lo pseudocodice di seguito:

```
Function addToClass(d,c)
{
        dts:=termset({d})
        c.termset:=c.termset∪dts
        c.classet:=c.classet∪d
        tempRel:=0
        for t∈dts
                tempRel:=tempRel+R(t,d)

        c.relsClassText[d]:=tempRel
}
```

In particolare bisogna far notare come la matrice F delle distanze nel codice del presente lavoro è ottenuta selezionando un sottoinsieme dei testi da classificare da D e quindi costruendo una sottomatrice delle distanze F che è un sottoinsieme chiuso della matrice delle distanze completa. In seguito si è inserito incrementalmente un testo alla volta tra quelli non utilizzati per la costruzione di F.

4.1.1.2 Classificazione di nuovi testi

Una volta ottenute le classi dalla fase precedente da un insieme di testi in cui:

$$|D| > k$$

Si passa alla classificazione vera e propria di un testo secondo la tecnica k-NN.

Essa calcolerà una nuova dimensione per la matrice delle distanze D data dal nuovo testo da classificare $\underline{d}$, quindi costruirà una classifica dei testi più vicini rispetto a $\underline{d}$ e stimerà la classe di appartenenza valutando le classi dei primi k testi più vicini pesate con la distanza tra $\underline{d}$ e i suoi vicini.

Lo pseudocodice per la classificazione di un nuovo testo è esposto di seguito:

```
Function k-NN(C,F,d)
{
        maxdist:=max(F)
        F:=F∪newdist(d)
        minfromd:=min(F[d,:])
        if minfromd>maxdist
                generateclass({d},terms({d}))
        else
        {
                sortd:=sort(F[d,:])
                Dk:=sortd[1:k]
                c := arg min_{c_j∈C} ( Σ_{d_j∈Dk t.c.d_j∈c_j.classet} F ( d_j, d ) )
                addToClass(d,c)
        }
}
```

In questo algoritmo si calcola la distanza massima tra i testi già classificati:

```
maxdist:=max(F)
```

Quindi si aggiunge la riga e la colonna relativa alle distanze tra il nuovo testo da classificare e i testi già presenti:

```
F:=F∪newdist(d)
```

Si calcola quindi la distanza minima tra il nuovo testo $\underline{d}$ e i testi già classificati:

```
minfromd:=min(F[d,:])
```

Se $\underline{d}$ è distante da tutti gli altri testi al punto che quello più vicino a $\underline{d}$ ha una distanza superiore a tutte le distanze tra i testi già presenti allora $\underline{d}$ andrà a formare una nuova classe:

```
if(minfromd>maxdist)
        generateclass({d},terms({d}))
```

Altrimenti si costruisce l'insieme Dk e si sceglie come classe di appartenenza quella classe in cui i testi che appartengono ad essa minimizzano la distanza rispetto a $\underline{d}$:

sortd:=sort(F[d,:])
Dk:=sortd[1:k]
c:= $arg\ \min_{c_j \in C} \left(\sum_{\underline{d_j} \in Dk\ t.c. \underline{d_j} \in c_j.classet} F\left(\underline{d_j}, \underline{d}\right)\right)$
addToClass(d,c)

4.1.1.3 Parametrizzazione dell'algoritmo

Per quanto riguarda l'implementazione pratica dell'algoritmo vanno selezionati due parametri:

1. **Dimensione insieme di inizializzazione**: dato l'insieme totale dei testi D bisogna selezionare quanti e quali testi andranno a far parte dell'insieme di inizializzazione per la creazione delle prime classi. In questo lavoro è stato scelto un valore del 20% della dimensione totale di D:

$$p = |D| * q, q = 0.2$$

2. **Valore di k**: va quindi specificato anche il valore di k, per questo lavoro è stato scelto un valore di k pari al 10% della grandezza dell'insieme di inizializzazione.

$$k = \lceil (|D| * q) * 0.1 \rceil$$

4.1.1.4 Complessità k-NN

La complessità di questo algoritmo qui esposta si basa sulle considerazioni fatte nell'implementazione per questo lavoro.

Come visto k-NN si compone di due fasi:

1. Inizializzazione in cui vengono identificate le prime classi.
2. Inserimento incrementale dei testi con eventuale creazione delle classi.

La complessità quindi è facilmente prevedibile che conterrà due addendi di dimensioni confrontabili e quindi considerazioni circa la complessità asintotica pura vanno bene per un'analisi teorica ma non sono attendibili dal punto di vista pratico escludendo una delle due componenti.

Si parte quindi col costruire la sottomatrice delle distanze, che rappresenterà la matrice delle distanze iniziale F.

Questo consiste nel selezionare p testi e calcolare le distanze reciproche tra i testi di questo sottoinsieme. La selezione di p testi in maniera casuale da D costa: $O(p)$ e il calcolo delle reciproche distanze che è implementabile con due cicli annidati:

```
for i=1…p-1
        for j=i…p
                calcolaDistanza(testo(i),testo(j))
```

con complessità: $O\left(\frac{p^2}{2}\right)$.

Una volta creata tale matrice $F \in \mathbb{M}_{\mathbb{R}}^{p\times p}$ si chiama la funzione:

```
initk-NN(F)
```

Essa ordina gli elementi della matrice F in ordine crescente di distanze:

```
sortDTexts:=sort(F)
```

il cui costo sarebbe: $O(p^2 * \ln(p^2))$ ma per motivi legati alle strutture dati implementati questo costa: $O(p^4)$. Generando una matrice $sortDTexts \in \mathbb{M}_{\mathbb{R}}^{p\times p}$ che, usando la rappresentazione per matrici sparse, avrà cardinalità: $|sortDTexts| = \frac{p^2}{2}$. In seguito vengono cercate le classi per ogni coppia di testi, quindi:

```
for i=1…|sortDTexts|
{
        dx:=sortDTexts[i].dx
        dy:=sortDTexts[i].dy
        cx:=findClass(dx)
        cy:=findClass(dy)

        if cx=null ∧ cy=null
                generateClass({dx, dy},terms({dx, dy})
        else if cx≠null ∧ cy=null
                addToClass(dy,cx)
        else if cx=null ∧ cy≠null
                addToClass(dx,cy)
}
```

Dipenderà dal costo della ricerca delle classi dei testi dx e dy di ogni coppia di testi:

```
cx:=findClass(dx)
cy:=findClass(dy)
```

questo costerà $O(2 * |C|)$. E dipenderà anche dal costo dell'aggiunta o della generazione di una classe contenente i testi, dato che la creazione di una nuova classe ha un costo costante bisogna analizzare la funzione:

```
addToClass(dy,cx)
```

per determinare la complessità massima.

Essa richiede principalmente l'estrazione dei termini appartenenti ad un testo:

```
dts:=termset({d})
```

che può essere ottenuto con complessità costante ma, per necessità legate al linguaggio questo purtroppo costa $O(n * m)$.

In seguito vi è il ciclo:

```
for t∈dts
        tempRel:=tempRel+R(t,d)
```

che, facendo un'ipotesi fortemente conservativa si può assumere che:

$$|dts| = \ln(n)$$

Da cui si ottiene una complessità della funzione:

```
addToClass(dy,cx)
```

pari a: $O(n * m + \ln(n))$.

A fronte di queste considerazioni l'inizializzazione avrà una complessità totale di:

$$O\left(p + \frac{p^2}{2} + p^4 + \frac{p^2}{2} * (2 * |C| + n * m + \ln(n))\right)$$

Si passa quindi alla classificazione vera e propria per quei testi non interessati dalla fase di inizializzazione. La rimozione dei testi già classificati dall'insieme dei testi totali costa: $O(p)$ quindi la classificazione dei testi non classificati:

```
for d∈D-classified(D)
        kNN(C,F,d)
```

effettuerà $m - p$ iterazioni.

La funzione:

```
Function k-NN(C,F,d)
{
        maxdist:=max(F)
        F:=F□newdist(d)
        minfromd:=min(F[d,:])
        if minfromd>maxdist
                generateclass({d},terms({d}))
        else
        {
                sortd:=sort(F[d,:])
                Dk:=sortd[1:k]
                c:= arg min_{c_j∈C} ( Σ_{d_j∈Dk t.c.d_j∈c_j.classet} F ( d_j, d ) )

                addToClass(d,c)
        }
}
```

Verrà chiamata quindi $m - p$ volte e inizialmente determina il massimo dalla matrice F il cui calcolo avrà una complessità che aumenta da $O(p^2)$ alla prima chiamata di $max(\dots)$ a $O((m-1)^2)$ all'ultima chiamata, questo avrà un costo sulle $m - 1 - p$ iterazioni pari a:

$$O\left(\frac{[(m-1)^2 + p^2] * [(m-1)^2 - p^2 + 1]}{2}\right)$$

Stessa cosa il determinare la minima distanza tra i testi già classificati e il testo da classificare, sapendo che per ogni testo le distanze tra i testi già classificati vanno da p alla prima chiamata della funzione $min(\dots)$, fino a $m - 1$ all'ultima chiamata. Ho una complessità di calcolo della minima distanza pari a:

$$O\left(\frac{(m + p - 1) * (m - p)}{2}\right)$$

Supponendo l'instradamento lungo il ramo else della funzione k-NN ottengo per quanto riguarda la funzione di ordinamento, che è come quella della fase di inizializzazione:

```
sortd:=sort(F[d,:])
```

una complessità totale lungo le varie iterazioni pari a:

$$O\left(\frac{[(m-1)^4+p^4]*[(m-1)^4-p^4+1]}{2}\right)$$

Quindi si passa al calcolo della classe che minimizza le distanze tra il testo da classificare $\underline{d}$ e i primi k testi più vicini, questo costa per ogni iterazione:

$$O\big((k+1)*|C|\big)$$

E viene iterato su $m-p$ iterazioni, quindi:

$$O\big((m-p)*(k+1)*|C|\big)$$

Il secondo addendo avrà quindi una complessità di:

$$O\left(\frac{[(m-1)^2+p^2]*[(m-1)^2-p^2+1]}{2}+\frac{[(m-1)^4+p^4]*[(m-1)^4-p^4+1]}{2}\right.$$
$$\left.+(m-p)*(k+1)*|C|\right)$$

Che, considerando i termini numericamente più grandi, è asintoticamente esprimibile mediante alcune semplificazioni migliorative come:

$$O\left(\frac{(m-1)^8-p^8}{2}\right)$$

Questo risulta diversi ordini di grandezza più grande rispetto al primo addendo relativo all'inizializzazione:

$$O\left(p+\frac{p^2}{2}+p^4+\frac{p^2}{2}*(2*|C|+n*m+\ln(n))\right)$$

E soprattutto dipendente dalla dimensione relativa ai testi m che tenderà a crescere indefinitamente.

Risulta già analiticamente confrontabile questa complessità con quella ottenuta tramite l'algoritmo ExhaustiveSets che risultava essere:

$$O(p^3 * \ln p)$$

Dove in questo caso p era il numero di termini ridotto dalle precedenti fasi di riduzione delle dimensioni: $p < n$.

4.1.2 LSI

Come visto nel paragrafo 2.3.2.2 questo metodo si basa sulla decomposizione a valori singolari economica che ha un costo:

$$\forall A \in \mathbb{M}_{\mathbb{R}}^{n \times m} \Rightarrow \left(m \geq n \wedge O(m * n^2)\right) \vee \left(m < n \wedge O(n * m^2)\right)$$

Ossia è quadraticamente proporzionale alla dimensione più piccola della matrice.

Inoltre essa si basa sulla struttura latente della matrice da cui è possibile considerare il coseno tra i vettori per ottenere dei valori di similarità. L'approccio usato in questo lavoro consiste nella costruzione di una matrice di correlazione in cui gli elementi esprimono la similarità tra i vettori corrispondenti agli indici.

Di seguito sono esposte le assunzioni adottate, lo pseudocodice e il calcolo relativo alla complessità.

4.1.2.1 Procedimento e assunzioni preliminari

Si parte considerando la matrice di rilevanza $R \in \mathbb{M}_{\mathbb{R}}^{n \times m}$ e quindi si calcola la sua decomposizione a valori singolari:

$$R = U * \Lambda * \mathrm{V}^{\mathrm{T}}$$

Dove U, Λ e V rispettano le osservazioni del paragrafo 2.3.2.2.

Quindi, considerando i valori singolari sulla diagonale di Λ si calcola un vettore:

$$\underline{\lambda} \in \mathbb{R}^{r-2}\ t.c.\ \underline{\lambda} = \left(\frac{\Lambda_{2,2}}{\Lambda_{3,3}}, \frac{\Lambda_{3,3}}{\Lambda_{4,4}} \dots \frac{\Lambda_{r-1,r-1}}{\Lambda_{r,r}}\right)$$

se $\max_i(\underline{\lambda}) \geq 2$ allora $s = i$ altrimenti $s = r$. Dove $r = r(R)$ è il rango di R.

Si calcolano pertanto le matrici ridotte: $V_s \in \mathbb{M}_{\mathbb{R}}^{m \times s}$ e $\Lambda_s \in \mathbb{M}_{\mathbb{R}}^{s \times s}$ e si calcola la matrice prodotto di queste due che conterrà la struttura latente della matrice R nel senso delle colonne che rappresentano i testi.

$$L \in \mathbb{M}_{\mathbb{R}}^{m \times s}\ t.c.\ L = V_s * \Lambda_s$$

Quindi si calcola la matrice di correlazione $C \in \mathbb{M}_{\mathbb{R}}^{m \times m}$ che esprime la similarità tra le varie righe di L che rappresentano la struttura latente dei testi. Usando una notazione simil-Matlab® si può definire:

$$C_{i,j} = \frac{\underline{L}_{i,:} * \underline{L}_{j,:}}{\left\|\underline{L}_{i,:}\right\|_2 * \left\|\underline{L}_{j,:}\right\|_2}\ \forall i = 1 \dots m\ \forall j = 1 \dots m$$

Dove $\underline{L}_{i,:}$ e $\underline{L}_{j,:}$ rappresentano le righe i e j della matrice L.

Dal momento che la matrice di correlazione è simmetrica si considererà solo la parte superiore ossia con indici:

$$\forall i = 1 \dots m-1\ \forall j = i \dots m$$

Infine, fissata una soglia $t = \frac{1}{2}$ si costruiscono le classi considerando tutte le coppie di similarità nella matrice di correlazione:

$$C_{i,j} \geq t \Rightarrow mergeClasses(t_i, t_j)$$

Con t_i e t_j testi i-esimo e j-esimo.

4.1.2.2 Pseudocodice LSI

Lo pseudocodice di questo metodo ricalca i calcoli appena esposti:

```
Function LSI(R)
{
        [U,S,V]:=svd(R, 'econ')
        sv:=vectorize(S)
        s:=rank(R)
        for i=1…size(sv)-1
        {
                temp:=sv(i)/sv(i+1)
                if temp≥2 && temp>max
                {
                        s:=i
                        max:=temp
                }
          }
        Vs:=reduce(V,s)
        Ss:=reduce(S,s)
        L:=Vs*Ss
        for i=1…m-1
                for j=i+1…m
                        C(i,j):=L(i,:)*L(j,:)/(norm(L(i,:),2)*norm(L(j,:),2))
        for i=1…m-1
                for j=i+1…m
                {
                        if C(i,j)≥t
                                if exist(class(i))==true && exist(class(j))==true
                                        mergeClasses(class(i),class(j))
                                else if exist(class(i))==true && exist(class(j))==false
                                        class(i).addToClass(j)
                                else if exist(class(i))==false && exist(class(j))==true
                                        class(j).addToClass(j)
                                else
                                        classes.add(createClass(i,j))
                }
}
```

4.1.2.3 Complessità LSI

La complessità di questo algoritmo è calcolabile in maniera molto lineare.

Innanzi tutto vi è la complessità della decomposizione a valori singolari che è proporzionale al quadrato della dimensione più piccola della matrice R, quindi: $O(n * m^2) \vee O(m * n^2)$.

In seguito vi è la complessità del calcolo dell'indice in cui $\frac{\lambda_i}{\lambda_{i+1}}$ è massimo che costerà: $O(r)$ con r rango della matrice R.

Quindi per il calcolo della matrice L il costo sarà dato dal prodotto delle matrici V_s e Λ_s che avrà costo: $O(m * s^3)$.

Una volta trovata L si passa al calcolo della matrice di correlazione C che siccome è effettuato da due cicli for annidati ed è una matrice triangolare superiore esclusa la diagonale principale costerà: $O\left(\frac{m*(m-1)}{2}\right)$ e infine analogamente si costruiscono le classi scandendo C con costo uguale a quello appena calcolato: $O\left(\frac{m*(m-1)}{2}\right)$.

La complessità totale pertanto sarà, sommando tutti i termini:

$$O\big((n * m^2 \vee m * n^2) + r + m * s^3 + m * (m-1)\big)$$

Supponendo che sia $m > n > r > s$ avrò una complessità asintotica pari a:

$$O(m^2)$$

Questa complessità soffre di problemi analoghi a quella calcolata per k-NN ossia è dipendente dai testi che possono crescere a dismisura, a differenza della complessità asintotica calcolata per l'algoritmo proposto, ExhaustiveSets:

$$O(p^3 * \ln p)$$

Che dipende dal numero di termini.

4.2 Sistema di test

Come indicato nel paragrafo 3.12.2.1 viene qui fornita la descrizione del sistema di test nelle due componenti principali:

1. **Hardware**:
   CPU: Intel® Pentium® Dual CPU T2370 @1.73GHz
   Memoria principale, RAM: 2 slot di memoria DDR2 1024MB PC2-5300 @333MHz.
   Memoria di massa, hard disk: 320GB 2,5'' cache 16MB SATA @7200Rpm.
2. **Sistema Operativo**: Microsoft Windows 7 Professional® Service Pack 1 64bit.
3. **Java Virtual Machine**: JRE 1.7 update 5 64bit.
4. **Processi concorrenti in esecuzione**: kernel del sistema operativo e Avast!Antivirus®.
5. **Codice da eseguire e codice di ispezione**: essi sono indicati nei paragrafi precedenti di questo lavoro.

4.3 Insiemi di test

Come indicato nel paragrafo 3.12.3 vengono di seguito fornite informazioni maggiormente dettagliate circa gli insiemi di test scelti.

4.3.1 Insieme di test: Reuters-21578

L'insieme di test scelto nel paragrafo 3.12.3 è Reuters-21578. Esso è composto da 22 files con estensione .sgm che contengono 1000 estratti di articoli Reuters ciascuno tranne l'ultimo che ne contiene 578.

Inoltre vengono forniti dei files di testo contenenti i termini usati nei testi, gli acronimi, i nomi delle organizzazioni pubbliche e private presenti, i luoghi geografici, i topics e infine le descrizioni dei topics.

Insieme a questi files viene fornito un Document Type Definition (o DTD) riportato di seguito:

```
<!ELEMENT LEWIS (REUTERS+)>
<!ELEMENT REUTERS
(DATE,(UNKNOWN|MKNOTE)*,TOPICS,PLACES,PEOPLE,ORGS,EXCHANGES,COMPANIES,UNKNOW
N?,TEXT)>
<!ELEMENT HEAD (#PCDATA)>
<!ELEMENT DATE (#PCDATA)>
<!ELEMENT TOPICS (#PCDATA|D)*>
<!ELEMENT PLACES (#PCDATA|D)*>
<!ELEMENT PEOPLE (#PCDATA|D)*>
<!ELEMENT ORGS (#PCDATA|D)*>
<!ELEMENT EXCHANGES (#PCDATA|D)*>
<!ELEMENT COMPANIES (#PCDATA|D)*>
<!ELEMENT TEXT (#PCDATA|TITLE|BODY|AUTHOR|DATELINE)*>
<!ELEMENT TITLE ANY>
<!ELEMENT BODY ANY>
<!ELEMENT DATELINE (#PCDATA)>
<!ELEMENT AUTHOR (#PCDATA)>
<!ELEMENT D (#PCDATA)>
<!ELEMENT UNKNOWN (#PCDATA)>
<!ELEMENT MKNOTE (#PCDATA)>

<!ATTLIST REUTERS
                        OLDID CDATA #REQUIRED
                        NEWID CDATA #REQUIRED
                        CSECS CDATA #IMPLIED
                        N CDATA "--"
                        SET CDATA "XXX"
                        CGISPLIT CDATA #REQUIRED
                        LEWISSPLIT CDATA #REQUIRED
                        TOPICS (YES|NO|BYPASS) "YES">
<!ATTLIST TOPICS
                TYPE CDATA "DSET">
<!ATTLIST PLACES
                TYPE CDATA "DSET">
<!ATTLIST PEOPLE
                TYPE CDATA "DSET">
<!ATTLIST EXCHANGES
```

```
                    TYPE CDATA "DSET">
<!ATTLIST ORGS
                    TYPE CDATA "DSET">
<!ATTLIST COMPANIES
                    TYPE CDATA "DSET">
<!ATTLIST TEXT
                    TYPE (NORM|BLAH|UNKNOWN|UNPROC|BRIEF) "NORM">

<!ENTITY lt "&lt;">
<!ENTITY amp "&amp;">

<!-- The following definitions are for downline larkup tokenisation -->

<!ELEMENT W (#PCDATA)>
<!ATTLIST W C CDATA "W">
<!ATTLIST W N CDATA #IMPLIED>
```

Come si può notare ogni estratto è racchiuso in un tag <REUTERS> i cui figli principali sono: la data, il topic, i luoghi di interesse, le persone, le organizzazioni e il testo vero e proprio. Il testo quindi conterrà il titolo, un corpo e l'autore.

Per quanto riguarda questo lavoro si è considerato come testo in ingresso il titolo e come classe di afferenza il topic. Si sono quindi estratti i testi che erano associati a un qualunque topic, escludendo quindi quei testi non afferenti ad alcun topic tramite la lettura dell'attributo TOPICS del tag <REUTERS>.

Il DTD e i relativi file .sgm sono stati opportunamente modificati e aggiornati per renderli compatibili con la libreria javax.xml.parsers relativa alla distribuzione 7 di Java.

Si è preso come insieme di test i primi 300 testi con classificazione presenti.

4.3.2 Insieme di test: Torcia

Questo dataset consiste in due file di tipo Comma Separated Values.

Un file contiene i termini di riferimento per le classi, il formato adottato è:

```
Classe-1; Termine-1, Termine-2, Termine-n;
Classe-2; Termine-1; Termine-2; Termine-n;
```

Un file contenente i testi con le relative classi di appartenenza:

```
Testo-1; Classe-1, Classe-2, Classe-n;
Testo-2; Classe-2, Classe-3, Classe-n;
```

Si è preso come insieme di testi un dataset composto da 90 testi classificati manualmente.

4.4 Modello di misura della qualità

Per la misura della qualità degli algoritmi di classificazione è necessario associare le classi create con quelle lette. Una classe è identificata o dall'etichetta associata oppure, in mancanza di questa, dai testi che ne fanno parte.

Quindi un algoritmo può classificare in base alle classi identificate dall'etichetta o in base all'aggregazione tra i termini. Per quanto riguarda le misure di qualità degli algoritmi è necessario avere una associazione tra classi esistenti di confronto e classi create dagli algoritmi. Nel caso in cui anche l'algoritmo si attenga alle etichette delle classi, l'associazione è immediata e diretta. Nel caso in cui l'algoritmo classifichi liberamente per aggregazione è necessario introdurre il concetto di "similarità tra classi".

4.4.1 Similarità tra classi

Il problema pertanto verte sul calcolo della similarità tra classi che possono essere viste come insieme anonimo di testi. Per calcolare la similarità tra insiemi si è scelto di usare l'indice di Jaccard che è definito in (33) come:

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

Con A e B insiemi anonimi di elementi.

Quindi, relativamente alla notazione usata nella tesi:

$$J(c_a, c_b) = \frac{|classet(c_a) \cap classet(c_b)|}{|classet(c_a) \cup classet(c_b)|}$$

4.4.2 Associazione delle classi

Per associare le classi create con quelle reali si è usato il metodo più lineare possibile ossia associare alla classe reale la classe creata con similarità maggiore.

Questo comporta una serie di considerazioni:

1. **Aggregazione**: l'aggregazione, come esposto nel paragrafo 3.12.1.3, è effettuata tramite microaveraging, quindi mediando gli indici ottenuti sulle classi di riferimento. Le classi di riferimento sono quelle reali, ossia lette dal dataset Reuters tramite i topics.
2. **Classi orfane**: alcune classi create possono rimanere senza corrispondente classe letta.
3. **Associazioni multiple**: alcune classi create posso avere più classi lette corrispondenti.

Pertanto la relazione che persiste è una funzione suriettiva.

Lo pseudocodice che realizza questa associazione tra le classi è esposto di seguito:

```
Function associate(readedClasses, createdClasses)
{
        for rc∈readedClasses
        {
                maxSim:=0
                maxClass:={}
                for cc∈createdClasses
                        if Jaccard(rc,cc)>maxSim
                        {
                                maxSim:=Jaccard(rc,cc)
                                maxClass:=cc
                        }
                association[rc]:=cc
        }
        return association
}
```

Questo algoritmo costa $O(c)$ con $c = |C|$ che è pari alla cardinalità dell'insieme delle classi.

4.5 Confronto con ExhaustiveSets

Vengono forniti in questo paragrafo i risultati dei confronti tra l'algoritmo base, ExhaustiveSets e gli algoritmi individuati tra quelli presenti in letteratura: k-NN e LSI.

4.5.1 Risultati qualitativi

Vengono fornite tre tabelle, una per ogni algoritmo confrontato, in cui sono misurate le metriche di qualità in relazione alle riduzioni scelte, sono state usate tre cifre significative:

1. **ExhaustiveSets**:

Riduzioni			**Metriche**					
Stop-word removal	**Suffix stemming**	**Term weighting**	**Precision: P**	**Recall: R**	**Fallout: F**	**Overlap: O**	**Error: E**	**F_α-measure: F_α**
			0.365	0.365	0.000	0.365	0.630	0.365
		√	0.365	0.365	0.000	0.365	0.635	0.365
	√		0.381	0.365	0.000	0.365	0.635	0.373
	√	√	0.365	0.365	0.000	0.365	0.635	0.365
√			0.381	0.365	0.000	0.365	0.635	0.373
√		√	0.365	0.365	0.000	0.365	0.635	0.365
√	√		0.381	0.365	0.000	0.365	0.635	0.373
√	√	√	0.397	0.365	0.000	0.365	0.635	0.380

Tabella 4 Metriche qualità ExhaustiveSets con dataset Reuters-21578

2. **k-NN**:

Riduzioni			**Metriche**					
Stop-word removal	**Suffix stemming**	**Term weighting**	**Precision: P**	**Recall: R**	**Fallout: F**	**Overlap: O**	**Error: E**	**F_α-measure: F_α**
			0.000	0.0635	0.000	0.000	1.000	0.000
		√	0.000	0.0159	0.000	0.000	1.000	0.000
	√		0.000	0.206	0.000	0.000	1.000	0.000
	√	√	0.000	0.000	0.000	0.000	1.000	0.000
√			0.000	0.000	0.000	0.000	1.000	0.000
√		√	0.000	0.000	0.000	0.000	1.000	0.000
√	√		0.000	0.0952	0.000	0.000	1.000	0.000
√	√	√	0.000	0.0159	0.000	0.000	1.000	0.000

Tabella 5 Metriche qualità k-NN con dataset Reuters-21578

3. **LSI**:

Riduzioni			Metriche					
Stop-word removal	**Suffix stemming**	**Term weighting**	**Precision: P**	**Recall: R**	**Fallout: F**	**Overlap: O**	**Error: E**	**F_α-measure: F_α**
			0.0158	0.000	0.000	0.000	1.000	0.000
		√	0.000	0.000	0.000	0.000	1.000	0.000
	√		0.0158	0.000	0.000	0.000	1.000	0.000
	√	√	0.0158	0.000	0.000	0.000	1.000	0.000
√			0.0158	0.000	0.000	0.000	1.000	0.000
√		√	0.0158	0.000	0.000	0.000	1.000	0.000
√	√		0.0158	0.000	0.000	0.000	1.000	0.000
√	√	√	0.0158	0.000	0.000	0.000	1.000	0.000

Tabella 6 Metriche qualità LSI con dataset Reuters-21578

4.5.2 Risultati prestazionali

Le metriche di prestazione vertono totalmente sui tempi di esecuzione degli algoritmi. Per la misura di queste metriche sono state inserite istruzioni di checkpoint nei punti individuati per la loro misura.

Al solito sono stati misurati gli impatti delle varie fasi di riduzione selezionate.

Vengono esposte qui delle tabelle che raccolgono i dati relativi alle classificazioni fatte con i tre algoritmi e relative alle varie operazioni preliminari:

Riduzioni			**Metriche**					
Stop-word removal	**Suffix stemming**	**Term Weight**	**t_{par} (s)**	**t_{rel} (s)**	**$t_p=t_{par}+t_{rel}$ (s)**	**t_{exe} (s) ExhaustiveSets**	**t_{exe} (s) k-NN**	**t_{exe} (s) LSI**
			4.882	0.330	5.212	48.629	3.873	130.615
		√	3.791	0.322	4.113	30.742	3.763	133.397
	√		3.494	0.376	3.870	66.109	3.514	118.183
	√	√	3.806	0.657	4.463	33.061	3.592	108.870
√			3.182	0.330	3.512	12.069	3.326	108.531
√		√	3.011	0.671	3.682	8.651	3.620	108.467
√	√		3.291	0.335	3.626	17.059	3.328	106.030
√	√	√	3.416	0.657	4.073	8.617	3.780	106.482

Tabella 7 Prestazioni algoritmi con dataset Reuters-21578

4.5.3 Considerazioni

I risultati ottenuti sono in linea con quanto esposto nei paragrafi Capitolo 1 e 3.1, ossia un algoritmo stabile rispetto alle riduzioni implementate, individuate nei paragrafi 3.7 e 3.9, in particolare:

1. **Influenza tra riduzioni e qualità**: la qualità delle classificazioni ottenute con le riduzioni applicate non deve variare sostanzialmente, pena l'instabilità dell'algoritmo stesso.
2. **Influenza tra riduzioni e prestazioni**: un buon algoritmo deve avere miglioramenti in termini prestazionali relativamente alle riduzioni implementate. Dal momento che il numero di testi non è riducibile pena la non classificazione degli stessi l'unica cosa che si può fare è agire sui termini e quindi un buon algoritmo deve operare su questi proprio come l'algoritmo proposto in questo lavoro.
3. **Vincolo di tempo reale**: come esposto si vuole ottenere un algoritmo capace di operare in tempo reale.

I primi due punti sono stati esauditi pienamente, seppur con valori qualitativi bassi. Il terzo punto non è stato esaudito in quanto i risultati ottenuti non sono accettabili.

Sono però risultate evidenti alcune discrepanze tra gli algoritmi fin qui confrontati:

1. **Qualità delle classificazioni**: k-NN e LSI offrono una qualità delle classificazioni nulla, essi soffrono il fatto che i testi sono molto corti e spesso i titoli, assunti come testi di analisi, non riportano nemmeno i termini che sono l'oggetto del contenuto.
2. **Importanza dei sinonimi**: ExhaustiveSets non presenta risultati soddisfacenti in termini qualitativi, esso offre valori di precision e recall attorno a $\cong 0.37$ come è valutabile dalla Tabella 4. Questo è dovuto al fatto che spesso i termini sono sinonimi tra loro ma l'algoritmo non lo tiene in considerazione facendo decadere le prestazioni. Questa limitazione in questo lavoro non era sormontabile in quanto la classificazione doveva tenere in considerazione solamente la sintassi dei testi. Lavori successivi possono valutare il peso dell'uso dei sinonimi aggregando i termini in tal senso.

3. **Confronti prestazionali**: a dispetto da quanto immaginato ExhaustiveSets offre prestazioni peggiori rispetto a k-NN. Questo è dovuto da un'assunzione parzialmente non rispettata nella scelta del dataset Reuters-21578, ossia che il numero di testi fosse maggiore dal numero di termini, nelle simulazioni è risultato che a fronte di 300 testi erano presenti circa 700 termini. Si ottiene una stabilizzazione del numero di termini quando si supera i 10000 testi ma una tale mole di dati avrebbe reso impossibile effettuare le simulazioni con gli algoritmi k-NN e LSI sulla macchina in uso.

4.6 Confronto con ExhaustiveSets+Meta

Come anticipato nel paragrafo 3.12.3 si confrontano gli algoritmi ExhaustiveSets e la sua estensione ExhaustiveSets+Meta usando come dataset Torcia.

Dimostrata la solidità dell'algoritmo base circa le riduzioni implementate e viste le considerazioni fatte per ExhaustiveSets+Meta nel paragrafo 3.11.2 si adottano le riduzioni che rimuovono le stop-word e la riduzione di term weighting. Non è possibile adottare la riduzione di suffix stemming in quanto i testi presenti nel dataset Torcia sono in lingua italiana e non è disponibile in letteratura un algoritmo di stemming relativo.

Quindi si forniscono i risultati qualitativi e prestazionali di ExhaustiveSets e quelli di ExhaustiveSets+Meta in dipendenza al parametro di soglia $INCLUSIONPERCENTAGE$ che verrà fatto variare nell'insieme di valori individuati nel paragrafo 3.11.5.

4.6.1 Risultati qualitativi

1. **ExhaustiveSets**:

Metriche					
Precision: P	**Recall: R**	**Fallout: F**	**Overlap: O**	**Error: E**	**F_α-measure: F_α**
1.000	0.2224	0.000	0.224	0.243	0.612

Tabella 8 Metriche qualità ExhaustiveSets con dataset Torcia

2. **ExhaustiveSets+Meta**:

INCLUSIONPERCENTAGE	Metriche					
	Precision: P	**Recall: R**	**Fallout: F**	**Overlap: O**	**Error: E**	**F_α-measure: F_α**
0.1	0.634	0.837	0.267	0.566	0.219	0.721
0.2	0.647	0.834	0.242	0.574	0.208	0.729
0.3	0.647	0.826	0.238	0.570	0.211	0.726
0.4	0.646	0.821	0.241	0.566	0.214	0.722
0.5	0.641	0.782	0.220	0.546	0.220	0.705
0.6	0.658	0.783	0.197	0.556	0.206	0.715
0.7	0.651	0.712	0.174	0.520	0.225	0.680
0.8	0.651	0.637	0.140	0.476	0.230	0.644
0.9	0.662	0.592	0.110	0.459	0.229	0.625

Tabella 9 Metriche qualità ExhaustiveSets+Meta con dataset Torcia

4.6.2 Risultati prestazionali

1. **ExhaustiveSets**:

Metriche			
t_{par} (s)	**t_{rel} (s)**	**$t_p=t_{par}+t_{rel}$ (s)**	**t_{exe} (s)**
0.639	0.329	0.968	132.438

Tabella 10 Metriche prestazionali ExhaustiveSets con dataset Torcia

2. **ExhaustiveSets+Meta**:

INCLUSIONPERCENTAGE	Metriche			
	t_{par} (s)	t_{rel} (s)	$t_p=t_{par}+t_{rel}$ (s)	t_{exe} (s)
0.1	0.687	0.328	1.015	0.718
0.2	0.640	0.345	0.985	0.655
0.3	0.655	0.329	0.984	0.671
0.4	0.655	0.329	0.984	0.718
0.5	0.624	0.329	0.953	0.655
0.6	0.640	0.329	0.969	0.656
0.7	0.624	0.329	0.953	0.656
0.8	0.639	0.344	0.983	0.639
0.9	0.640	0.329	0.969	0.640

Tabella 11 Metriche prestazionali ExhaustiveSets+Meta con dataset Torcia

4.6.3 Considerazioni

Le considerazioni da fare riguardo il confronto tra i due algoritmi oggetto di questo paragrafo sono di due tipi: prestazionali e qualitative.

Le considerazioni prestazionali vertono sulla soddisfazione del vincolo di classificazione in tempo reale. Le considerazioni qualitative vertono sulla capacità di ottenere valori soddisfacenti relativi alle metriche di qualità individuate.

Considerazioni prestazionali

Confrontando le tempistiche ottenute nella classificazione del dataset Reuters-21578 con il dataset Torcia il tempo di esecuzione di ExhaustiveSets è peggiorato nonostante una cardinalità inferiore dell'ultimo dataset. Questo effetto è dovuto al fatto che il dataset Torcia è un'estrazione tramite crawling da twitter di testi relativi alle emergenze quindi con parecchi termini in comune. Questa particolarità fa aumentare il numero delle passate necessarie a ExhaustiveSets per costruire tutte le combinazioni utili e possibili.

In termini prestazionali quindi ExhaustiveSets+Meta surclassa di due ordini di grandezza ExhaustiveSets in quanto operando con l'ausilio di metadati è possibile operare con combinazioni di termini più precise e immediate.

ExhaustiveSets+Meta garantisce quindi il vincolo di esecuzione in tempo reale in quanto per classificare una finestra di circa 100 testi impiega complessivamente circa 2 secondi tra operazioni preliminari e classificazione.

Considerazioni qualitative

Il risultato sorprendente evidenziato dalle misure relative alla qualità è che l'algoritmo ExhaustiveSets in questo secondo confronto offre una precision massima. Questo risultato non è attendibile in quanto è ottenuto a fronte di una recall bassa. La giustificazione è dovuta al fatto che i testi che vengono classificati in una classe generica sono pochi e sicuramente afferenti a quella specifica classe. Per converso sono pochissimi i testi rilevati (su un dataset di 90 testi spesso quelli associati con una classe sono uno, massimo due, chiaramente corretti mentre non ve ne sono di scorretti) rispetto a quelli da rilevare. Quei pochi, insomma, è pressoché impossibile vengano classificati non correttamente.

In poche parole ExhaustiveSets sul dataset Torcia opera lontano dal suo break-even point mentre come è facile notare ExhaustiveSets+Meta opera in prossimità di tale punto.

Per quanto riguarda ExhaustiveSets+Meta si nota come l'algoritmo non presenta valori di precision e recall massimi in quanto soffre del fatto che i testi sono brevi e la classificazione avviene considerando solo aspetti sintattici per favorirne la rapidità.

Questo per le fasi iniziali della classificazione, nelle fasi seguenti intervenendo l'apprendimento, tali metriche non possono che migliorare.

I valori di qualità ottenuti sono tuttavia pari ai valori ottenuti presenti in letteratura nella classificazione di testi più lunghi, con fasi di training e tempi di esecuzione maggiori.

Entrando nello specifico le metriche ottenute si nota come ExhaustiveSets+Meta offre valori di $Overlap$ significativamente migliori rispetto a ExhaustiveSets e valori di $Error$ molto bassi a fronte di una $Recall$ abbastanza elevata. Questo significa che, a differenza di ExhaustiveSets, l'algoritmo esteso classifica con buona frequenza e qualità.

Per quanto riguarda il valore di soglia che regola l'apprendimento si nota come con valori di soglia bassi si facilita l'apprendimento e l'algoritmo quindi presenta una $Recall$ significativa a scapito di una $Precision$ più bassa. Questo risultato era atteso in quanto, includendo con più facilità nuovi termini, l'algoritmo classifica più testi con minor precisione. Con valori di soglia elevati invece si predilige la qualità dei metadati forniti apprendendo con più difficoltà nuovi termini e scartando con facilità termini poco utili. Questo tende a migliorare sensibilmente la $Precision$, peggiorando la $Recall$.

Capitolo 5 **Conclusioni**

Il lavoro proposto ha voluto fornire una base indicativa per la classificazione guidata dalla sintassi, fornendo spunti e indicazioni per il miglioramento delle prestazioni qualitative e quantitative. Si è pervenuti ad un algoritmo, ExhaustiveSets, che si prefigge l'obiettivo di lavorare sulla dimensione più comoda e performante, i termini, insieme ad altre ipotesi esposte nel paragrafo 3.10.2. Verificata la non applicabilità di questo in un contesto di esecuzione in tempo reale si è valutata l'opportunità di estenderlo con l'uso di metadati. Mantenendo le assunzioni che rendevano tale algoritmo performante su testi brevi si è quindi ampliato il metodo applicando computazioni analoghe sui metadati, pervenendo all'algoritmo ExhaustiveSets+Meta.

Alla luce di quanto considerato in questo lavoro ExhaustiveSets offre una buone base di partenza nella classificazione sintattica di testi brevi. Esso, seppur in forma primitiva, fornisce risultati qualitativi migliori degli algoritmi più performanti presenti in letteratura, si veda il paragrafo 2.4.

Come detto introducendo l'opportunità di valutare termini sinonimi durante la classificazione è possibile migliorare ulteriormente le prestazioni in termini di qualità di ExhaustiveSets. Tale ottimizzazione invece rimane un punto interrogativo per quanto riguarda i due algoritmi usati come confronto, k-NN e LSI.

Questa problematica è stata superata con l'ausilio di metadati ottenendo un algoritmo ExhaustiveSets+Meta che garantisce l'esecuzione in tempo reale migliorando le metriche qualitative. Inoltre si è introdotta la capacità di apprendimento all'algoritmo in

modo che, operando su finestre di testi e incrementalmente, possa ulteriormente raffinare la capacità di classificazione.

Chiaramente non è possibile fornire indicazioni utili circa le tecniche proposte in un uso concreto. Quello che si può fare è confrontare i risultati dell'algoritmo proposto e di un algoritmo semantico a fronte di un set di dati già classificati provenienti da social network, questa attività tuttavia risulta difficoltosa proprio per la mancanza di grossi dataset classificati in letteratura e per la dispendiosità di fornire una classificazione manuale e puntuale di grosse moli di dati.

Bibliografia

1. The Free Dictionary - class definition. [Online] http://www.thefreedictionary.com/class.

2. The Free Dictionary - category definition. [Online] http://www.thefreedictionary.com/category.

3. **Grossman, Frieder.** *Information Retrieval Algorithms and Heuristics.* s.l. : Springer, 2004.

4. **Manning, Cristopher and Raghavan, Prabhakar.** *An Introduction to Information Retrieval.* 2007.

5. **Salton, Gerard and Buckley, Christopher.** Term-weighting approaches in automatic text retrieval. *Information Processing & Management.* 1988, Vol. 24, 5, pp. 513 - 523.

6. **Broder, Andrei.** A taxonomy of web search. *SIGIR Forum.* 2002, Vol. 36, 2, pp. 3-10.

7. **Maron, M.E.** *Automatic Indexing: An Experimental Inquiry.* RAND Corporation. Santa Monica : s.n., 1961.

8. **Sebastiani, Fabrizio.** *A Tutorial on Automated Text Categorisation.* Istituto di Elaborazione dell'Informazione, Consiglio Nazionale delle Ricerche. Pisa : s.n., 2002.

9. **Joachims, Thorsten.** *A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization.* School of Computer Science, Carnegie-Mellon University. Pittsburg : s.n., 1996.

10. **Kjersti, Aas and Line, Eikvil.** *Text Categorisation: A Survey.* Norwegian Computing Center. Oslo : s.n., 1999.

11. **Rish, I.** *An empirical study of the naive Bayes classifier.* T.J. Watson Research Center, IBM.

12. **Hirotoshi, Taira and Masahiko, Haruno.** Feature Selection in SVM Text Categorization. *AAAI-99 Procedings.* 1999.

13. **Joachims, Thorsten.** *Text Categorization With Support Vector Machines: Learning with Many Relevant features.* Informatik LS8, Universitat Dortmund. Dortmund : s.n.

14. **Hearst, Matti A.** Support vector machines. *IEEE Intelligent Systems.* 1998, pp. 18 - 28.

15. **Burges, Christopher J.C.** A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Directory.* 1998, pp. 121 - 167.

16. Lagrange Multiplier. *Wikipedia.* [Online] http://en.wikipedia.org/wiki/Lagrange_multiplier.

17. **Golub, G. and Reinsch, C.** *Handbook for matrix computation II.* New York : Springer-Verlag, 1971.

18. **Papadimitriou, Christos H., et al.** Latent Semantic Indexing: A Probabilistic Analysis. *PODS.* 1998, pp. 159 - 168.

19. **Cohen, William W. and Singer, Yoram.** Context-Sensitive Learning Methods for Text Categorization. *ACM Transaction on Information System.* 1999, Vol. 17, 2, pp. 141-173.

20. **Denison, David G. T., Mallick, Bani K. and Smith, Adrian F. M.** *A Bayesian CART algorithm.* Department of Mathematics, Imperial College. London : s.n., 1998.

21. **Schapire, Robert E. and Singer, Yoram.** *BoosTexter: A System for Multiclass Multi-label Text Categorization.* AT&T Labs, AT&T. Florham Park, NJ : s.n., 1998.

22. **Freund, Yoav and Schapire, Robert E.** *Experiments with a New Boosting Algorithm.* AT&T Research, AT&T. Murray Hill, NJ : s.n., 1996.

23. JavaCC Home. *JavaCC.* [Online] http://javacc.java.net/.

24. LexTek Stop Words. [Online] http://www.lextek.com/manuals/onix/stopwords1.html.

25. **Porter, M.F.** *An algorithm for suffix stripping.* Computer Laboratory, Cambridge University. Cambridge : s.n.

26. IBM Archives: System/370 Model 165. [Online] http://www-03.ibm.com/ibm/history/exhibits/mainframe/mainframe_PP3165.html.

27. **Lewis, David D.** Term Clustering of Synctactic Phrases. *SIGIR Conference.* 1990, pp. 385 - 404.

28. —. An evaluation of phrasal and clustered representations on a text categorization task. *SIGIR.* 1992, pp. 37 - 50.

29. **Maltraversi, Marco.** *SEO e SEM Guida avanzata al web marketing.* Milano : Edizioni FAG Milano, 2011.

30. **Lewis, David D.** *Evaluating Text Categorization.* Computer and Information Science Dept., University of Massachusetts. Amherst, MA : s.n.

31. Profiling (computer programming). *Wikipedia.* [Online] http://en.wikipedia.org/wiki/Profiling_%28computer_programming%29.

32. Distanza (matematica). *Wikipedia.* [Online] http://it.wikipedia.org/wiki/Distanza_%28matematica%29.

33. **Real, Raimundo and Vargas, Juan M.** The Probabilistic Basis of Jaccard's Index of Similarity. *Systematic Biology.* 1996, Vol. 45, pp. 380 - 385.

34. **Epifani, Ilenia, Ladelli, Lucia and Posta, Gustavo.** *Appunti per il corso di Calcolo delle Probabilità.* Milano : s.n., 2005.

35. **Strzalkowski, Tomek.** *Robust Text Processing in Automated Information Retrieval.* Courant Institute of Mathematical Sciences, New York University. 1994.

36. **Bozzon, Alessandro, et al.** *Search Computing: Information Retrieval.* Dipartimento di Elettronica e Informazione, Politecnico di Milano. Milano : s.n., 2008.

37. **Salton, G., Wong, A. and Yang, C. S.** A Vector Space Model for Automatic Indexing. *Information Retrieval and Language Processing.* Novembre 1975, pp. 613 - 620.

38. **Lewis, David D. and Ringuette, Marc.** *A Comparison of Two Learning Algorithms for text Categorization.* Las Vegas : s.n., 1994.

39. **Yiming, Yang.** An Evaluation of Statistical Approaches to Text Categorization. *Information Retrieval I.* 1998, pp. 69 - 90.

40. **Schwartz, Candy.** Web Search Engines. *Journal of the American Society for Information Science.* 1998, pp. 974 - 982.

41. **Yu, Cui, et al.** *Indexing the Distance: An Efficient Method to KNN Processing.* 2001.

42. **Osaragi, Toshihiro.** *Spatial Clustering Method for Geographic Data.* Centre for Advanced Spatial Analysis, University College London. Londra : s.n., 2002.

43. **Quarteroni, Alfio, Sacco, Riccardo and Saleri, Fausto.** *Matematica Numerica.* Milano : Springer, 2004.