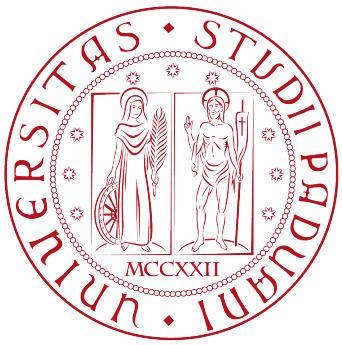
Università degli Studi di Padova



Dipartimento di matematica

Corso di Laurea Magistrale in Informatica

Sviluppo di un *tool* di supporto per la classificazione di *e-mail* concernenti l'assistenza a sistemi informativi

Candidato: Mirko Polato
Relatore: Ch.mo Prof. Alessandro Sperduti

Luglio 2013

Alla mia famiglia e
ad Alessandra.

Indice

1 Introduzione **1**

2 Text Classification **5**
2.1 Definizione 6
2.1.1 Single/Multi Label classification 6
2.1.2 Hard/Soft classification 7
2.1.3 Category-Pivoted classification 7
2.2 Approccio Machine Learning 7
2.2.1 Valutazione delle performance 8
2.2.2 Indicizzazione dei documenti 10
2.2.3 Riduzione dello spazio delle *feature* 13
2.3 Classificazione automatica di *e-mail* 15
2.3.1 *Concept drift* 16
2.3.2 Stato dell'arte 17
2.4 Classificatore Probabilistico *Naïve Bayes* 19
2.5 *Support Vector Machine* 22
2.5.1 Definizione 22

3 Sistema di classificazione preesistente **25**
3.1 Scenario d'uso 25
3.1.1 Prima versione 27
3.1.2 Seconda versione 27
3.1.3 Terza versione 29
3.2 Architettura generale 31
3.2.1 Struttura del classificatore 31
3.2.2 *Thread* di classificazione 32
3.2.3 *Servlet* per la presentazione dei risultati 34
3.2.4 *Thread* di *feedback* 35
3.3 Tecnologie utilizzate 35
3.3.1 *DBMS*: *MySQL* 35
3.3.2 Gestione dei file *XML*: *SAX* 35
3.3.3 *Machine Learning*: *Mallet* 35

4 Il nuovo sistema **37**
4.1 Scenario operativo 37
4.1.1 L'azienda coinvolta 37
4.1.2 Tassonomia 37
4.2 Flusso dei dati 41
4.3 Requisiti del nuovo sistema 41

4.4 L'ambiente di Test 45

5 Fase di indicizzazione **47**
5.1 Il *bug* del sistema iniziale 47
5.2 Configurazione dell'ambiente di Test 47
5.3 Analisi dei messaggi 48
5.4 *Pre-processing* 48
5.4.1 Nuovo dizionario 48
5.4.2 *Stop Words* 49
5.4.3 Riconoscimento della lingua 50
5.4.4 Gestione dei valori numerici 50
5.4.5 Parti considerate 51
5.4.6 L'importanza dell'oggetto 51
5.4.7 Gestione degli allegati 53
5.4.8 Il nuovo algoritmo di *pre-processing* 54
5.5 Indicizzazione 54
5.5.1 *Feature weighting* 55
5.5.2 Feature selection 56

6 Fase di classificazione **59**
6.1 Il classificatore *SVM* 59
6.1.1 *Kernel* polinomiale 60
6.1.2 *Kernel* RBF 66
6.2 Classificazione sulla tassonomia 67
6.2.1 Stato dell'arte 68
6.2.2 Categorie sulle foglie 69
6.2.3 Nuovo algoritmo di *training* 70
6.2.4 Nuovo algoritmo di classificazione 71
6.2.5 Classificazione con *Naïve Bayes* 73
6.2.6 Il problema del sottoalbero 6.3.2 73
6.3 *Term clustering* 76
6.3.1 *Term clustering* manuale 76
6.3.2 *Term clustering* automatico 78
6.4 Aggiornamento del modello 82
6.5 Presentazione dei risultati e gestione dei *feedback* 84
6.5.1 Multiutenza 84

7 Riconoscimento automatico dei cambiamenti nella tassonomia **87**
7.1 Classi obsolete 87
7.2 Specializzazione 88

7.3 *Adaptive Resonance Theory* (*ART*) 88
7.3.1 Architettura di base 89
7.3.2 ART2 91
7.4 Applicazione di ART2 97
7.5 Test 98

8 Conclusioni e sviluppi futuri **103**
8.1 Possibili sviluppi futuri 104

Elenco delle figure

2.1 I vettori di supporto di una *SVM*. 24

3.1 Esempio di tassonomia. 26
3.2 Flusso dei dati nel vecchio sistema durante la classificazione. . . 28
3.3 Flusso dei dati nel vecchio sistema per la gestione dei *feedback*. . 29
3.4 Architettura generale del sistema. 32
3.5 Flusso di esecuzione del *thread* di *training*. 33
3.6 Flusso di esecuzione del *thread* di classificazione. 34

4.1 Tassonomia della *knowledge base*. 38
4.2 Nuova tassonomia. 40
4.3 Flusso dei dati nel nuovo sistema. 42

5.1 Flusso della funzione di *tokenizzazione* del sistema precedente. . 49
5.2 *Performance* del sistema precedente (*Naïve Bayes*). 50
5.3 Classificatore con *Upper case multiplication* (2x). 53
5.4 Prestazioni con la gestione degli allegati. 54
5.5 Diagramma di flusso della nuova fase di *pre-processing*. 55
5.6 Prestazioni con l'utilizzo di *Tf-Idf*. 56
5.7 Prestazioni senza l'utilizzo della *Feature Selection*. 56

6.1 Esempio di struttura della tassonomia con nodi interni con esempi associati. 70
6.2 Esempio di struttura della tassonomia con nodi interni vuoti. . . 71
6.3 Prestazioni sull'intera tassonomia con *Naïve Bayes*. 73
6.4 Prestazioni con la ripetizione dei valori numerici (5x). 74
6.5 Prestazioni con l'adozione del *term clustering* manuale. 77
6.6 Prestazioni al primo livello con l'adozione del *term clustering* manuale. 78
6.7 Prestazioni con l'adozione del *term clustering* automatico Smith-Waterman ($\Omega = 0.5$). 81
6.8 Diagramma di flusso dei nuovi *thread*. 83

7.1 Architettura della rete neurale *ART* 90
7.2 Architettura della rete neurale *ART2* 92

Elenco delle tabelle

4.1 Distribuzione delle *e-mail* su un *dataset* di 755 *e-mail*. 39
4.2 Distribuzione delle *e-mail* con la nuova tassonomia. 41

5.1 *Performance* del classificatore ripetendo le *feature* dell'oggetto (*ES* medio 0.19). 52

6.1 Prestazioni con *kernel* polinomiale: $d = 2$, $\gamma = 2^{-5}$ 60
6.2 Prestazioni con *kernel* polinomiale: $d = 3$, $\gamma = 2^{-5}$ 61
6.3 Prestazioni con *kernel* polinomiale: $d = 4$, $\gamma = 2^{-5}$ 61
6.4 Prestazioni con *kernel* polinomiale: $d = 2$, $\gamma = 2^{-7}$ 62
6.5 Prestazioni con *kernel* polinomiale: $d = 3$, $\gamma = 2^{-7}$ 62
6.6 Prestazioni con *kernel* polinomiale: $d = 4$, $\gamma = 2^{-7}$ 63
6.7 Prestazioni con *kernel* polinomiale: $d = 2$, $\gamma = 2^{-9}$ 63
6.8 Prestazioni con *kernel* polinomiale: $d = 3$, $\gamma = 2^{-9}$ 64
6.9 Prestazioni con *kernel* polinomiale: $d = 4$, $\gamma = 2^{-9}$ 64
6.10 Prestazioni con *kernel* polinomiale: $d = 2$, $\gamma = 2^{-11}$ 65
6.11 Prestazioni con *kernel* polinomiale: $d = 3$, $\gamma = 2^{-11}$ 65
6.12 Prestazioni con *kernel* polinomiale: $d = 4$, $\gamma = 2^{-11}$ 66
6.13 Prestazioni con *kernel* RBF ($2^{-3} \leq \gamma \leq 2^{-7}$) 67
6.14 Prestazioni con *kernel* RBF ($2^{-9} \leq \gamma \leq 2^{-15}$) 67
6.15 Matrice di confusione del sottoalbero 6. 73
6.16 Matrice di confusione del sottoalbero 6.3. 74
6.17 Matrice di confusione del sottoalbero 6.3.2. 74
6.18 Matrice di confusione del sottoalbero 6 dopo la modifica. 75
6.19 Matrice di confusione del sottoalbero 6.3 dopo la modifica. . . . 75
6.20 Matrice di confusione del sottoalbero 6.3.2 dopo la modifica. . . 75
6.21 *Cluster* individuati manualmente. 76
6.22 Matrice di confusione del sottoalbero 6 con *term clustering* manuale. 77
6.23 Matrice di confusione del sottoalbero 6.3 con *term clustering* manuale. 77
6.24 Matrice di confusione del sottoalbero 6.3.2 con *term clustering* manuale. 77
6.25 *Cluster* individuati con Smith-Waterman, $\Omega = 0.5$ (*False Positive = 1*). 81
6.26 *Cluster* individuati con Needleman–Wunsch, $\Omega = 0.5$ (*False Positive = 43*). 82

7.1 Confronto tra *ART2*, *EM* e *KM*. (Parte 1/3) 100

7.2 Confronto tra *ART2*, *EM* e *KM*. (Parte 2/3) 100
7.3 Confronto tra *ART2*, *EM* e *KM*. (Parte 3/3) 100

1 Introduzione

I messaggi di posta elettronica rappresentano una delle forme di comunicazione più veloce, economica e accessibile dei nostri tempi. L'utilizzo di questo mezzo di comunicazione è in continua crescita, sia per uso privato che per uso professionale. Quando una *e-mail* viene consultata da un utente, in base alle proprie preferenze, può decidere se salvarla oppure ignorarla. Spesso il salvataggio dei messaggi di posta elettronica avviene in una struttura di cartelle, solitamente gerarchica. Mantenere questa struttura ordinata e controllata può diventare problematico se il traffico di messaggi in ingresso risulta troppo elevato. Questo accade non di rado ad aziende che offrono servizi ai clienti, i quali, per ogni tipo di esigenza, non esitano a chiedere informazioni. In questi contesti, avvalersi di un addetto che si occupa dello smistamento del traffico in entrata può non essere una soluzione soddisfacente. Per suddetti motivi si è resa indispensabile l'adozione di meccanismi in grado di organizzare e classificare le *e-mail* in modo automatico.

Il lavoro di tesi, descritto in questo documento, ha come obiettivo lo sviluppo di un sistema di classificazione automatica di *e-mail*, che offre altresì degli strumenti utili per la gestione della struttura tassonomica delle *directory* in cui esse sono memorizzate. Il sistema nasce sotto lo stimolo di una esigenza concreta manifestata da un'azienda Padovana che si occupa dell'installazione e della manutenzione di infrastrutture informatiche per l'ottimizzazione della gestione di piattaforme *IT*. L'azienda offre oltre all'assistenza a domicilio anche un sistema di *help desk* allo scopo di offrire un servizio attivo 24 ore su 24. Nel loro lavoro i tecnici attingono ad una *knowledge base* centralizzata e allo stesso tempo alimentano la base di conoscenza con informazioni proprie, sotto forma di messaggi di posta elettronica, inviati ad una casella dedicata. In questo scenario il classificatore deve analizzare e distribuire le *e-mail* in ingresso alla casella di posta in una tassonomia prefissata. Il sistema è stato realizzato partendo da uno preesistente sviluppato presso l'Università di Padova in precedenti lavori di tesi. Tale progetto, nato nel 2007, ha subìto diverse modifiche

ed è stato adattato alle esigenze delle aziende che hanno collaborato al suo sviluppo.

Classificare in modo automatico un messaggio di posta elettronica richiede l'utilizzo di tecniche di *Machine Learning* (*ML*) adeguate. Nello specifico, essendo le *e-mail* costituite principalmente da testo, si ha a che fare con la cosiddetta *Text classification* (TC), una delle branche più studiate dell'*Information retrieval*. Questa disciplina si occupa, dato un insieme di categorie/argomenti (classi) ed una collezione di documenti scritti in linguaggio naturale, di associare ad ogni documento la/le categoria/e corrispondente/i. Rispetto ai classici documenti, la posta elettronica, presenta alcune peculiarità che devono essere prese in considerazione durante la fase di *pre-processing* necessaria a trasformare il testo, scritto in linguaggio naturale, in un oggetto facilmente manipolabile dagli algoritmi di apprendimento automatico. Lo stesso dominio in cui opera l'azienda è una fonte preziosa di informazione che può essere sfruttata per definire regole *ad hoc* atte all'aumento delle prestazioni della classificazione. Inoltre, a differenza del sistemi preesistente che si limitava alla classificazione di primo (al più secondo) livello, il nuovo sistema è in grado di classificare sull'intera tassonomia. Infine, è stato necessario adottare un meccanismo di *feedback*, ovvero premettere all'utente, che ha conoscenza della tassonomia, di confermare o riassegnare la classificazione corretta alle *e-mail*. In questo modo il *training set* viene continuamente alimentato, migliorando l'apprendimento del nuovo modello. Questo è stato reso possibile grazie all'utilizzo di un *Web Server* che espone una *servlet* di presentazione dei risultati. Agli utenti è sufficiente collegarsi ad una pagina *web* pubblica, autenticarsi, e dare il proprio *feedback* agendo su un menu a tendina.

Oltre alla fase di classificazione, presente anche nei sistemi precedenti, è stato aggiunto un *tool* per la gestione semi automatica della tassonomia di classi. Con il trascorrere del tempo, le tecnologie evolvono e con esse anche i prodotti che l'azienda offre ai propri clienti. Come conseguenza, la tassonomia di classi della *knowledge base* muta e necessita di continue revisioni. Per questo motivo è stato richiesto lo sviluppo di uno strumento che propone periodicamente delle modifiche alla struttura di cartelle. Le modifiche possibili sono: la rimozione di una classe obsoleta o la nascita di nuove classi. Questo problema è stato affrontato con tecniche di *clustering* e nello specifico con reti neurali *ART*.

I test sono stati effettuati utilizzando come tecniche di apprendimento *Naïve Bayes* e le *Support Vector Machine*. Per le *SVM* è stato necessario eseguire una batteria di test per ricercare la combinazione di parametri che garantisse le migliori prestazioni. Per la fase di gestione della tassonomia, come detto

precedentemente, l'algoritmo adottato è *ART2*, che fa parte delle reti neurali basate sulla teoria della risonanza adattiva (*Adaptive Resonance Theory*). *ART2* a differenza di altre reti come *ART*, *FuzzyART* o *ARTMAP* è in grado di gestire valori di *input* continui (analogici) e non è supervisionato, quindi non richiede la definizione a priori del numero di *cluster* da generare.

Il presente documento è composto dai seguenti capitoli:

Capitolo 2 Panoramica sulla *Text classification* e la sua applicazione nell'ambito della classificazione dei messaggi di posta elettronica;

Capitolo 3 Descrizione del sistema preesistente, valutandone punti di forza e debolezze;

Capitolo 4 Analisi del nuovo scenario d'uso e definizione dei requisiti;

Capitolo 5 Descrizione delle modifiche e delle innovazioni apportate alla fase di indicizzazione;

Capitolo 6 Descrizione delle modifiche e delle innovazioni apportate alla fase di classificazione;

Capitolo 7 Analisi del problema della gestione della tassonomia e panoramica sulle reti neurali *ART* e il loro utilizzo in questo contesto;

Capitolo 8 Conclusioni e possibili sviluppi futuri.

2
Text Classification

In ambito accademico il termine *Information Retrieval* può essere definito come segue [5]: Information retrieval consiste nel trovare materiale (spesso documenti) di natura non strutturata (spesso testo) che soddisfi una determinata necessità all'interno di una collezione molto ampia.

Il settore dell'*Information Retrieval* è stato studiato fin dagli anni '70, però è solo con l'avvento di *Internet*, negli anni '90, che questa disciplina comincia a suscitare un interesse sempre maggiore. Oltre al *web*, che rappresenta esso stesso un archivio di documenti (le pagine *web*), si aggiunge la crescente disponibilità di documenti in formato digitale. Questo *overload* di informazioni ha portato alla necessità di ideare metodi che riescano ad accedere a tali informazioni in modo flessibile.

La *Text Classification* (TC - a.k.a. *Text Categorization*) è una delle branche dell'IR che cerca di facilitare questo compito. Con *Text Classification* si intende quella disciplina che si occupa, dato un insieme di categorie/argomenti (dette spesso classi) ed una collezione di documenti scritti in linguaggio naturale, di associare ad ogni documento la/le categoria/e corrispondente/i. Inizialmente questo processo di categorizzazione veniva svolto manualmente, poi, con l'avvento dei computer, si sono sviluppate tecniche per svolgerlo in maniera automatica, in questo caso parliamo di *Automatic Text Classification*. Da quanto appena detto è possibile dividere la *TC* in base a questi due approcci:

- *Knowledge Engineering approach*: si classificano i documenti sulla base di regole definite manualmente da un esperto del dominio. Questo approccio, che potenzialmente può raggiungere performance superiori ad altri approcci automatici, non è utilizzato poiché la definizioni di tali regole può richiedere anni.

- *Machine Learning Approach*: tramite un processo induttivo viene generato un classificatore a partire da un insieme di esempi pre-classificati

(Apprendimento supervisionato).

Nel seguente documento viene trattato il *Machine Learning Approach*.

2.1 Definizione

La *Text Classification* si occupa di assegnare un valore booleano ad ogni coppia $\langle d_j, c_i \rangle \in \mathscr{D} \times \mathscr{C}$, dove $\mathscr{D}$ è l'insieme dei documenti e $\mathscr{C} = \{c_1, ..., c_{|C|}\}$ è l'insieme di categorie. Il valore booleano indica se il documento appartiene o non appartiene alla categoria indicata.
Formalmente, *TC* è una funzione $\mathscr{F}$ definita come:

$$\mathscr{F} : \mathscr{D} \times \mathscr{C} \rightarrow \{T, F\}.$$

Dove con F si intende *False*, il documento non appartiene alla categoria, e con T si intende *True*, il documento appartiene alla cateogoria. La funzione $\widetilde{\mathscr{F}}$ che approssima $\mathscr{F}$ è chiamata classificatore (oppure modello o ipotesi) e la sua definizione è analoga a quella di $\mathscr{F}$.

A questa definizione generale possono essere aggiunti alcuni vincoli che distinguono diverse tipologie di classificazione.

2.1.1 Single/Multi Label classification

Quando si tenta di classificare un documento non è scontato che quest'ultimo appartenga in maniera esclusiva ad un'unica categoria. Sulla base dell'applicazione in cui si sta eseguendo la classificazione questi documenti possono essere assegnati ad una e una sola classe oppure a più classi contemporaneamente:

- *Single-Label classification*: ogni documento appartiene esclusivamente ad una sola classe, ovvero dato un documento $d \in \mathscr{D}$

  $$\exists! c \in \mathscr{C} \text{ t.c. } \langle d, c \rangle = T.$$

- *Multi-Label classification*: un documento può appartenere a più classi contemporaneamente.

L'esempio più semplice di classificazione *Single-Label* è la classificazione binaria in cui $|\mathscr{C}| = 2$. Questo tipo di classificazione rappresenta una generalizzazione della classificazione *Multi-Label*, infatti quest'ultima può essere ottenuta ricorrendo a $|\mathscr{C}|$ classificatori binari, dove per ogni classe $c \in \mathscr{C}$ viene definito il classificatore binario

$$\mathscr{B} : \mathscr{D} \times \widetilde{\mathscr{C}} \rightarrow \{T, F\} \quad \text{con } \widetilde{\mathscr{C}} = \{c, \mathscr{C} \setminus \{c\}\}.$$

2.1.2 Hard/Soft classification

Come descritto nella sezione 2.1, un classificatore automatico deve prendere una decisione binaria per ogni coppia documento-categoria, ovvero propone una cosiddetta *hard categorization*. Non sempre questo genere di classificatori danno risultati soddisfacenti e si viene a preferire quindi metodi semi-automatici in cui il classificatore ritorna un grado di appartenenza di ogni documento rispetto alle classi, e successivamente un addetto si occuperà di scegliere quella più adeguata. Questo tipo di classificazione è detta *soft*. Spesso quello che il classificatore offre all'umano è un *ranking* di valori, nell'intervallo [0,1], detti *Categorization Status Values*.

2.1.3 Category-Pivoted classification

Nelle definizioni date finora quello che il classificatore esegue è un assegnazione (hard o soft) di un documento a una o più classi. Quindi la classificazione "gira attorno" al documento e viene appunto definita *document-pivoted classification*. Esistono però applicazioni in cui è più utile poter dire quali documenti fanno parte di una certa categoria, ovvero dato $c_i \in \mathscr{C}$ vogliamo trovare tutti i documenti $d_j \in \mathscr{D}$ tali che $\langle d_j, c_i \rangle = T$. Questo genere di classificazione è detta *category-pivoted*.

2.2 Approccio Machine Learning

Nell'approccio *Machine Learning* il classificatore viene costruito da un algoritmo detto *learner* o *trainer*, il quale, a partire da un insieme di documenti correttamente classificati (dati supervisionati), osserva le loro caratteristiche per poter prevedere la classificazione di un nuovo documento. In questo caso parliamo di algoritmo supervisionato, poiché è previsto il supporto di un umano che fornisce esempi classificati. La versione non supervisionata della classificazione è detta *clustering*.

Formalmente, l'algoritmo (Supervisionato) necessita di un *corpus* iniziale $\Omega = \{d_1, d_2, ..., d_{|\Omega|}\}$ di documenti classificati sulla base delle categorie nell'insieme $\mathscr{C} = \{c_1, c_2, ..., c_{|C|}\}$, ossia per ogni coppia $\langle d_j, c_i \rangle \in \Omega \times \mathscr{C}$ è noto il valore $\mathscr{F}(d_j, c_i)$, dove con $\mathscr{F}$ si intende la funzione definita nella sezione 2.1 come $\mathscr{F} : \mathscr{D} \times \mathscr{C} \to \{T, F\}$.
Usualmente l'insieme Ω è partizionato in tre insiemi disgiunti:

- *Training Set (TR)*: è il *corpus* che utilizza il *learner* per creare il classificatore;

- *Validation Set (TV)*: viene utilizzato per fare il *tuning* degli eventuali parametri del classificatore;
- *Test Set (Te)*: usato per valutare l'efficacia dei classificatori. Una volta che il classificatore propone una categorizzazione viene confrontata con quella data dall'esperto.
  È fondamentale che $\forall d \in \Omega$ t.c. $d \in TR \cup TV \Rightarrow d \notin Te$.
  Un approccio molto utilizzato per effettuare la scelta del classificatore è il *k-fold cross validation*, in cui vengono appresi k classificatori $\mathscr{F}_1, ..., \mathscr{F}_k$ suddividendo l'insieme Ω in k sottoinsiemi disgiunti $\Omega_1, ..., \Omega_k$ e applicando l'approccio *train-and-test* con $TR_i = \Omega \setminus \Omega_i$ e $TV_i = \Omega_i$. La valutazione dell'efficacia finale viene eseguita analizzando le prestazioni dei singoli k classificatori, e calcolando poi la media dei risultati ottenuti.

2.2.1 Valutazione delle performance

Esistono diversi indicatori per le perfomance di un classificatore, dove il più semplice è:

- *Accuracy*: misura la frazione di documenti classificati correttamente ovvero, detto N il numero di documenti classificati l'*accuracy* misura:
  $$a = \frac{N_{correct}}{N}.$$
  dove $N_{correct}$ è il numero di documenti classificati in modo corretto.

Tuttavia questa metrica non è sempre opportuna, come ad esempio in casi di classificazione binaria in cui le categorie $c, \neg c$ siano sbilanciate. Infatti in casi di questo tipo il semplice *trivial rejector*, classificatore che assegna sempre la classe più popolata, ottiene valori di *accuracy* molto elevati. In tal casi sono più significative le seguenti metriche:

- *Precision (π)*: misura la percentuale di documenti classificati come appartenenti a c_i, che effettivamente appartengono a c_i. Chiamiamo $\mathscr{C}_i$ l'insieme dei documenti che appartengono a c_i e $\widetilde{\mathscr{C}_i}$ l'insieme dei documenti classificati come c_i, la *precision* è definita come:
  $$\pi_i = \frac{|\mathscr{C}_i \cap \widetilde{\mathscr{C}_i}|}{|\mathscr{C}_i|}$$
- *Recall (ρ)*: misura la percentuale di documenti appartenenti a c_i correttamente riconosciuti come appartenenti a tale classe.
  $$\rho_i = \frac{|\mathscr{C}_i \cap \widetilde{\mathscr{C}_i}|}{|\widetilde{\mathscr{C}_i}|}$$

Per valutare i valori di *precision* e *recall* globali è necessario effettuare una media dei valori ottenuta per ogni singola classe c_i. Esistono due modi di calcolare questa media:

- *Microaveraging(μ)*: π e ρ vengono calcolate sommando tutte le singole decisioni

$$\pi^{\mu} = \frac{\sum_{i=1}^{|\mathscr{C}|} |\mathscr{C}_i \cap \widetilde{\mathscr{C}_i}|}{\sum_{i=1}^{|\mathscr{C}|} |\mathscr{C}_i|}$$

$$\rho^{\mu} = \frac{\sum_{i=1}^{|\mathscr{C}|} |\mathscr{C}_i \cap \widetilde{\mathscr{C}_i}|}{\sum_{i=1}^{|\mathscr{C}|} |\widetilde{\mathscr{C}_i}|}$$

- *Macroaveraging(M)*: π e ρ sono prima valutate singolarmente per ogni classe, e successivamente viene calcolata la media:

$$\pi^{M} = \frac{\sum_{i=1}^{|\mathscr{C}|} \pi_i}{|\mathscr{C}|}$$

$$\rho^{M} = \frac{\sum_{i=1}^{|\mathscr{C}|} \rho_i}{|\mathscr{C}|}$$

Presi singolarmente *precision* e *recall* possono dare risultati molto diversi poiché dipendono fortemente dalla distribuzione degli esempi nelle categorie. In base alle esigenze può essere più utile tenere in maggiore considerazione un indice piuttosto dell'altro. Una formula largamente utilizzata per combinare π e ρ è la seguente:

$$F_{\beta} = \frac{(\beta^2 + 1)\pi\rho}{\beta^2\pi + \rho}$$

dove β è un parametro che indica quanta importanza dare a π rispetto a ρ. Se si pone $\beta = 0$ allora $F_{\beta} = \pi$, mentre se $\beta = +\infty$ si ha che $F_{\beta} = \rho$. Infine, se $\beta = 1$ si ottiene F_1 che rappresenta la media armonica tra *precision* e *recall*

$$F_1 = \frac{2\pi\rho}{\pi + \rho}.$$

Nel caso si utilizzi il *ranking* è opportuno elencare altre due metriche:

- *Classification gap*: è dato dalla posizione della classe corretta nel *ranking* proposto. Per un insieme di documenti $\mathscr{D}$, è la somma dei *classification gap* di tutti i documenti dell'insieme;
- *Average Classification gap*: misura invece il valore medio del *classification gap* nell'insieme.

2.2.2 Indicizzazione dei documenti

Per poter utilizzare i documenti, le tecniche di apprendimento e classificazione, effettuano alcune elaborazioni preliminari per portare i documenti, scritti in linguaggio naturale, in un formato accessibile e comodo all'algoritmo, riducendo al minimo la perdita di informazioni. Formalmente, la fase cosiddetta di *pre-processing*, è una funzione definita come:

$$\mathscr{P} : d \rightarrow IREP$$

dove d è un documento e $IREP$ è la sua rappresentazione interna (*Internal RE-Presentation*). Tale funzione approssima il significato del documento attraverso l'estrazione di un insieme di *features*. Comunemente la fase di *pre-processing* è costituita dalle seguenti sottofasi:

1. Trasformazione dei documenti in sequenza di caratteri;
2. Suddivisione in *token*;
3. Normalizzazione;
4. Indicizzazione.

Trasformazione in sequenza di caratteri

I documenti in ingresso possono avere formati digitali diversi, che vanno da codifiche standard (es. UTF-8, ASCII) a formati proprietari (es. Microsoft Word). C'è bisogno quindi di riconoscere il formato e applicare la decodifica appropriata per trasformarli in puro testo. Una volta estratto il testo è possibile che il documento abbia al suo interno parti intrattabili (es. il documento presenta una parte corrotta) o indesiderate (es. *tag HTML*), queste devono essere eliminate poiché potrebbero compromettere le fasi successive.

Tokenization

Il testo grezzo in questa fase viene suddiviso in parti, dette *token*, ognuno dei quali rappresenta un'istanza di un tipo, maggiore è il numero di tipi maggiore sarà il numero di *token* estratti.
In questa fase, in base alle esigenze, è necessario decidere come trattare la punteggiatura. Generalmente questa viene trascurata, in altri casi è preferibile considerare solo i confini di frase tralasciando la punteggiatura interna. La scelta deve basarsi sul contesto in cui si sta operando.

Normalizzazione

La suddivisione in *token* non fa alcun tipo di considerazioni semantiche o di natura linguistica e questo porta ad avere *token* di tipo diverso anche se rappresentano uno stesso termine dal punto di vista semantico. Per ovviare (parzialmente) a questo problema si sono ideate diverse tecniche, le più comuni sono le seguenti:

- Rimozioni dei segni diacritici: anche in questo caso il contesto influisce sull'approccio, ma generalmente per rendere il testo uniforme si procede con la semplice rimozione dei segni diacritici. In lingue come l'inglese tale rimozione è quasi ininfluente, mentre in lingue come italiano o francese può avere effetti indesiderati.

- *Case folding*: l'uso delle lettere maiuscole è quasi esclusivamente una questione di natura formale e, a parte casi particolari, non ha a che vedere con la semantica. Quindi ridurre tutto il testo in minuscolo è una soluzione accettabile che aiuta a ridurre ambiguità dovute al *case* delle parole.

- *Stemming*: lo *stemming* rappresenta una forma di standardizzazione che estrae la radice di una parola, eliminando affissi e desinenze. In questo modo viene ridotto il numero di tipi aumentando così la frequenza di alcuni di essi. Bisogna considerare però che lo *stemming* si limita a troncare la parola mediante un'analisi morfologica semplificata che non sempre è corretta.

- *Lemmatization*: è un'altra forma di standardizzazione e consiste nella riduzione di una forma flessa di una parola alla sua forma canonica (non marcata), detta lemma. Rispetto alla precedente tecnica, questa si occupa anche di disambiguare tra le diverse forme base a cui può corrispondere una forma flessa.

- *Word Sense Disambiguation*: non è raro che una stessa parola abbia significati diversi in base al contesto in cui è posta, è quindi errato raggruppare indiscriminatamente nello stesso tipo *token* uguali. Questa tecnica cerca proprio di ovviare a questo problema, individuando le ambiguità che possono essere:

  - Omonimia: fenomeno per cui una stessa forma ortografica e fonologica esprime più significati;
  - Polisemia: i diversi significati attribuiti alla stessa parola sono legati etimologicamente e semanticamente. Si distingue dall'omonimia in

quanto, nel caso dell'omonimia, i diversi significati di un lessema si trovano a essere rappresentati da un'unica forma ortografica solo per caso.

Le tecniche utilizzate per effettuare la *word sense disambiguation* possono basarsi o su un dizionario (*dictionary-based*) oppure su algoritmi di *machine learning*.

- Casi particolari: esistono casi particolari in cui è possibile ridurre parti di testo in classi di equivalenza, come ad esempio le date ("26/10/1985" è equivalente a "26 ottobre 1985"), oppure le diverse forme di articolo in italiano o ancora la variazione grafica tra inglese *british* e inglese americano.

Indicizzazione

L'indicizzazione consiste nel trasformare il testo in un vettore di termini con associato un valore. Questo vettore viene definito *feature vector* dove i termini sono appunto le *feature*. I valori associati a queste *feature* sono fondamentali per la classificazione poiché sono ciò che gli algoritmi manipolano per produrre la loro ipotesi.
Esistono diversi approcci per assegnare questi pesi alle *feature*, i più semplici sono:

- *Feature Frequency*: viene semplicemente assegnato il numero di volte che la *feature* appare nel documento;
- *Feature Presence*: assegna un valore binario in cui 1 indica la presenza della *feature* mentre 0 l'assenza di quest'ultima.

Questi due approcci, pur essendo molto semplici, possono dare buoni risultati e sono applicabili anche non avendo a disposizione l'intero *training set*. Il problema principale è che il valore di una *feature* dipende esclusivamente dal documento in cui si trova senza tener conto degli altri, questo può portare a sovrastimare il valore di alcune di esse e a sottostimare il valore di altre.

Un approccio che prende in considerazione la totalità dei documenti (quindi necessita di tutto il *training set*) per valutare il peso di una *feature* è lo schema *Tf-Idf* (*Term frequency–Inverse document frequency*).
Tf-idf è una funzione $\mathcal{T}$ composta dalle funzioni $\mathcal{T}_{tf}$ (*tf*) e $\mathcal{T}_{idf}$ (*idf*), che dato un termine t_k nel documento d_j calcola:

$$\mathcal{T}(t_k, d_j) = \mathcal{T}_{tf}(t_k, d_j) \times \mathcal{T}_{idf}(t_k, d_j)$$

con

$$\mathscr{T}_{tf}(t_k, d_j) = \frac{\#(t_k, d_j)}{|d_j|}$$

$$\mathscr{T}_{idf}(t_k, d_j) = \log \frac{|Tr|}{\#_{Tr}(t_k)}$$

dove $\#(t_k, d_j)$ indica il numero di occorrenze di t_k in d_j e $\#_{Tr}(t_k)$ il numero di documenti che contiene t_k. Questo valore può essere normalizzato nell'intervallo $[0, 1]$, quando necessario, applicando la *cosine normalization*, definita come:

$$w(t_k, d_j) = \frac{\mathscr{T}(t_k, d_j)}{\sqrt{\sum_{s=1}^{|Tr|}(\mathscr{T}(t_s, d_j))^2}}$$

2.2.3 Riduzione dello spazio delle *feature*

Una volta completata l'indicizzazione solitamente si ottiene un insieme molto vasto di *feature*, detto *feature space*. Il fatto che sia molto ampio può causare principalmente due problemi: in primo luogo le *performance* del classificatore possono risentire dell'elevato numero di *feature* da gestire e in secondo luogo molte sono irrilevanti e alcune di queste possono portare al cosiddetto fenomeno dell'*overfitting*.
Con *overfitting* si intende quando un modello statistico si adatta ai dati osservati usando un numero eccessivo di parametri, in questo caso le *feature*.

Feature selection

Quando il nuovo spazio delle *feature* rappresenta un sottoinsieme dello spazio di partenza si parla di *feature selection*. Esistono diverse tecniche che effettuano una selezione delle *feature*, di seguito vengono descritte le più utilizzate.

Rimozione delle *stop words*

Vengono eliminati dal testo i termini di uso comune e che hanno scarsa influenza sul contenuto informativo del documento. Queste *feature* sono dette *stop words*, un esempio lo sono gli articoli, le preposizioni e le congiunzioni. Per effettuare la rimozione ci si basa solitamente su un dizionario che contiene le *stop words* della lingua con cui è scritto il documento.

Document frequency

La *Document frequency*, come dice il nome stesso, è la frequenza di una *feature* nella collezione di documenti, ossia in quanti documenti questa appare. Applicando un semplice *thresholding* che scarta tutti i termini poco frequenti (al di sotto della soglia) si riduce lo spazio delle *feature*. Esperimenti dimostrano che considerando solo il primo 10% delle parole più frequenti non si riducono le prestazioni del classificatore.

Information gain

L'*information gain* cerca di calcolare la quantità di informazione che una certa *feature* offre per predire una certa categoria. Per poter calcolare questa quantità bisogna prima definire il concetto di entropia che misura il grado di impurità di un'arbitraria collezioni di esempi. Detto D è un sottoinsieme del *training set* e assumendo che gli esempi in D possano appartenere alle classi dell'insieme $\mathscr{C}$, l'entropia è definita come segue:

$$\mathscr{H}(D) = -\sum_{c \in \mathscr{C}} P(c|D) \log P(c|D)$$

Ora è possibile definire l'*information gain* come:

$$IG(Tr,t) = \mathscr{H}(Tr) - \sum_{x \in \{Tr_{t^+}, Tr_{t^-}\}} \frac{|x|}{|Tr|} \mathscr{H}(x)$$

dove Tr_{t^+} e Tr_{t^-} indicano l'insieme di documenti del *training set* che contengono o meno il termine t. Come per la *document frequency* anche quì si applica un valore di soglia per eliminare tutti i termini con valore inferiore.

Chi-squared

Un altro metodo molto popolare è il χ^2. In statistica, il test χ^2 è applicato per verificare l'indipendenza tra due eventi, dove due eventi A e B sono indipendenti se $P(A,B) = P(A)P(B)$ o, in modo equivalente, $P(A|B) = P(A)$ e $P(B|A) = P(B)$. Nell'ambito della *feature selection*, i due eventi sono l'occorrenza di un termine e l'occorrenza della classe. Si può quindi creare un *ranking* dei termini in base alla classe applicando:

$$\chi^2(Tr,t,c) = \sum_{e_t \in 0,1} \sum_{e_c \in 0,1} \frac{(N_{e_t e_c} - E_{e_t e_c})^2}{E_{e_t e_c}}$$

dove e_t è pari a 1 se il documento contiene il termine t e e_c è 1 se il documento appartiene alla classe c. N è la frequenza osservata ed E il valore atteso.

2.3 Classificazione automatica di *e-mail*

La classificazione automatica di *e-mail* ([34], [27], [22], [13]) può essere definita come un caso particolare di classificazione automatica di testi. Formalmente, è una funzione $\widetilde{\mathscr{F}} : \mathscr{M} \times \mathscr{C} \rightarrow \{T, F\}$ dove $\mathscr{M}$ è l'insieme dei documenti scritti nel formato standard *MIME* (*Multipurpose Internet Mail Extensions*). Questo tipo di classificazione presenta alcune peculiarità specifiche:

- Natura semi-strutturata
- Formato non puramente testuale
- Contenuto multilingua
- Lunghezza del testo molto ridotta

Natura semi-strutturata

Come detto precedentemente le *e-mail* sono oggetti di tipo *MIME message*. *MIME* rappresenta uno standard di *internet* e al suo interno, oltre al testo della *e-mail* contiene diversi metadati semi strutturati. Un messaggio di posta elettronica contiene informazioni quali, data di invio, data di ricezione, allegati, mittente e lo stesso testo è suddiviso tra l'oggetto e il corpo. Tutte queste parti sono strutturate all'interno di un documento *MIME* e non è triviale capire quali di questi siano utili alla classificazioni e quali invece possano essere trascurate.

Formato non puramente testuale

Nella sezione 2.2.2 viene discusso il problema della trasformazione del documento in testo puro, prima decodificandolo e poi scartando le parti indesiderate. Le *e-mail* non hanno bisogno di alcuna decodifica poiché lo standard *MIME* supporta solo caratteri *ASCII*. Nonostante ciò un messaggio può contenere degli allegati e il loro formato non è vincolato. Inoltre non è raro che all'interno del corpo ci siano parti di *HTML* o un contenuto non testuale, come ad esempio immagini.

Contenuto multilingua

Una caratteristica tipica delle *e-mail* è la non unicità della lingua in cui è scritta. Il caso più tipico è quello in cui il corpo è scritto in una lingua e gli allegati sono in una lingua diversa, oppure nel corpo stesso appaiono parti in lingua

diversa, dovuto ad esempio ad un copia-incolla. Questo comporta alcune problematiche nella procedura di pre-elaborazione, descritta nella sezione 2.2.2, per esempio lo *stemming* o la rimozione delle *stop words*.

Lunghezza del testo molto ridotta

Generalmente il corpo di una *e-mail* è molto ridotto e deve quindi essere classificato sulla base di pochissime informazioni. Questa situazione può essere ribaltata se si considerano gli allegati che spesso hanno una lunghezza notevole.

2.3.1 *Concept drift*

È possibile, sia per variazioni dello scenario applicativo che per la semplice evoluzione dei bisogni informativi dell'utente, che la distribuzione delle classi vari nel tempo. Questa variazione può riguardare sia l'insieme delle classi, aggiungendone di nuove ed eliminando le obsolete, sia il modo in cui vengono classificate le *e-mail* nelle classi esistenti. Questo cambiamento nelle proprietà statistiche della distribuzione è fonte di problemi poiché il modello, creato tramite tecniche di *ML*, diviene meno accurato. In contesti in cui il *concept drift* ha buone probabilità di verificarsi è importante l'utilizzo di tecniche che riescano a rilevare questo cambiamento nella distribuzione dei dati (vedi [1]) in modo da poter controllare e modificare il modello.

2.3.2 Stato dell'arte

Un esempio di applicazione della *Text classification* alle *e-mail* si ha con lo *spam filtering*. In [41], Tretyakov (2004), fa una panoramica dei più popolari metodi di apprendimento automatico applicati allo *spam filtering*. Vengono confrontati il classificatore *Naïve Bayes* (NB), *k Nearest Neighbour* (*k*NN), *Artificial Neural Network* (ANN) e *Support Vector Machine* (SVM). I suoi risultati mostrano come metodi semplici (es. NB) raggiungano buoni risultati e siano abbastanza competitivi rispetto a tecniche più sofisticate come ad esempio SVM. Tretyakov inoltre mostra come la combinazione di più classificatori (NB ∪ SVM) possano ottenere prestazioni migliori rispetto ai classificatori presi singolarmente. Come avevano già mostrato, Drucker et al. (1999) [21], SVM raggiunge prestazioni migliori rispetto ai concorrenti. Nel loro lavoro Drucker, Wu e Vapnik hanno confrontato metodi quali *Ripper*, Rocchio e gli alberi di decisione con *boosting*. Sahami et al. (1998) [31] hanno utilizzato un approccio probabilistico con *Naïve Bayes* ottenendo ottimi risultati. Hanno inoltre mostrato come l'utilizzare informazioni derivanti dal dominio in cui si sta operando porti ad un aumento delle prestazioni.
Vi sono chiaramente altri problemi legati alla gestione delle *e-mail*, come ad esempio la necessità di identificare correttamente i diversi *thread*. Lewis et al. (1997) [14] dimostrano che riconoscere i *thread* corrispondenti alle conversazioni tra più persone può essere trattato come un compito di *language processing*. Un ulteriore problema, che verrà trattato ampiamente in questa tesi, è l'organizzazione delle *e-mail* in categorie o, come vengono definite da Giacoletto et al. [13] (2003), *folder*. Nel loro lavoro Giacoletto et al. evidenziano che nel corso del tempo le attività aziendali sono in continuo cambiamento e quindi è difficile che una struttura di *directory* rimanga invariata. Viene proposto un metodo basato sul *clustering*, in particolare l'algoritmo *k-Means*, che cerca di proporre una nuova struttura delle *directory*.
Payne e Edwards [16] (1995) per costruire il loro *Mail Agent Interface*, agente che organizza automaticamente i messaggi di posta elettronica, hanno confrontato due paradigmi di apprendimento, CN_2 (Clark e Niblett, 1989 [7]) basato sull'induzione di regole e IBPL1, che è una versione estesa dell'algoritmo *Memory Based Reasoning* [11]. I loro esperimenti hanno mostrato come IBPL1 raggiunga prestazioni migliori rispetto a CN_2. Riprendendo sempre il concetto di generazione di un insieme di regole, Cohen (1996) [8], propone il suo algoritmo *RIPPER*, che si basa sull'individuazione delle parole chiave. Confronta lo stesso algoritmo che da un lato considera solo la presenza o l'assenza delle *feature* dall'altro invece calcola *Tf-Idf*. Sorprendentemente Cohen mostra che le prestazioni migliori sono ottenute senza valutare la frequenza. Come fatto da Sahami per lo *spam filtering*, Provost (1999) [32] utilizza NB

e dimostra che questo approccio, nell'ambito della *Text Categorization* applicata ai messaggi di posta elettronica, ottiene risultati migliori dell'algoritmo proposto da Cohen. L'anno successivo Rennie [33], sempre con *Naïve Bayes* realizza un sistema di classificazione che è in grado di proporre per ogni *e-mail* in ingresso tre possibili classi di appartenenza. I risultati ottenuti sono ottimi, infatti con alta probabilità la classe di appartenenza compare nelle tre proposte dal sistema. Questo sistema, pur essendo semi-automatico, comporta una notevole riduzione del carico di lavoro dell'addetto specializzato alla supervisione finale.
Nel 2000 Kiritchenko et al. [19] sono i primi a utilizzare le SVM per la classificazione di *e-mail* in cartelle, mostrando come questa tecnica porti generalemente a prestazioni migliori rispetto all'approccio bayesano. Lo stesso Kiritchenko et al. (2004) [37] affronta il problema legato al flusso temporale delle *e-mail*. Infatti, le *e-mail* assumono un certo significato se considerate all'interno di una specifica sequenza. In questo lavoro viene proposto una soluzione che prende in considerazione le relazioni temporali in una sequenza di messaggi, individuando dei pattern temporali da integrare alle informazioni utilizzate dai metodi di apprendimento *content-based*. Martin et al. (2005) [40] si sono spinti oltre analizzando il traffico in entrata della casella di posta per individuare le *behavioral feature* utili a riconoscere lo *spam*. Bekkerman et al. (2004) [34] hanno effettuato una serie di test utilizzando famosi *banchmark* come *Enron* e *SRI corpora*, confrontando tecniche come NB, SVM, *MaxEntropy* e *Wide-Margin Winnow*, mostrando che in quasi tutti i test SVM ha prestazioni migliori rispetto ai concorrenti.
Un altro aspetto importante da valutare quando si ha a che fare con messaggi di posta elettronica, come detto nella sezione 2.3, riguarda la struttura e i campi in cui si articola un' *e-mail*. Bisogna valutare quale di queste parti sia utile alla classificazione e quali invece possano essere trascurate riducendo così lo spazio delle *feature*. Nel 2009 Lampert et al. [3] hanno proposto un sistema basato su SVM che individua all'interno del messaggio 9 segmenti, differenziati in base a criteri grafici, lessicali e ortografici. I test hanno fornito risultati incoraggianti soprattutto se viene ridotto il numero di segmenti estratti a 2 (*accuracy* del 93.6%).
In contesti molto ampi, ad esempio grosse aziende, è possibile che il flusso di *e-mail* in entrata sia davvero molto elevato e quindi non è possibile applicare tecniche di classificazione automatica non incrementali. Carmona et al. (2010) [22] presentano *GNUsmail*, un *framework open-source* per la classificazione di e-mail in *real-time*. In questo lavoro vengono presentate diverse tecniche incrementali quali *incremental NB*, *Adaptive Hoeffding Tree*, *Majority Class*, *OzaBag* e *OzaBoost*, affiancati da tecniche di *Drift Detection* come EDDM e DDM. Per farlo hanno utilizzato le due librerie gratuite *Weka* e *MOA* (*Massive*

Online analisys) per gli algoritmi e come *dataset* Enron. Hanno mostrato come le prestazioni migliori si hanno con *OzaBag* con *DDM* (*Dynamic Drift Detection)*, anche se in generale le *performance* sono equilibrate, con l'eccezione di *Majority class*.

In questo documento verranno affrontati i problemi relativi all'organizzazione delle *e-mail* in *folder*, o categorie. I metodi di apprendimento automatico utilizzati sono *Naïve Bayes* e le *Support Vector Machine* che sono presentati rispettivamente nelle sezioni 2.4 e 2.5. Verrà inoltre analizzato il problema del cambiamento della struttura di *directory* nel corso del tempo e verrà proposto un algoritmo che cerca di capire quando questo sta avvenendo.

2.4 Classificatore Probabilistico *Naïve Bayes*

Nella classificazione di testi si hanno a disposizione una serie di documenti $d \in \mathscr{D}$, dove $\mathscr{D}$ è il *document space*, e un insieme di classi $\mathscr{C} = \{c_1, c_2, ..., c_{|\mathscr{C}|}\}$. Tipicamente il *document space* è uno spazio ad alta dimensionalità mentre le classi sono definite da un umano in base alle proprie esigenze. Quando si applicano algoritmi di apprendimento automatico per la classificazione di testi, quello che si sta cercando è una funzione γ che mappa documenti in classi:

$$\gamma : \mathscr{D} \rightarrow \mathscr{C}$$

Questo tipo di apprendimento è detto supervisionato poiché si ha bisogno di un insieme di documenti pre-classificati ($Tr = \{\langle \vec{d}_1, c_{\vec{d}_1} \rangle, \langle \vec{d}_2, c_{\vec{d}_2} \rangle, ..., \langle \vec{d}_{|Tr|}, c_{\vec{d}_{|Tr|}} \rangle\}$) forniti da un supervisore. Quindi un metodo di apprendimento automatico può essere definito come una funzione Γ che dato un insieme di allenamento Tr restituisce in output una funzione di classificazione γ:

$$\Gamma(Tr) = \gamma.$$

Un esempio di algoritmo supervisionato è il *multinomial Naïve Bayes*, o semplicemente *Naïve Bayes* che è un classificatore probabilistico. Nei classificatori probabilistici la probabilità che un documento $\vec{d}_j = \langle t_{1j}, t_{2j}, ..., t_{nj} \rangle$, con$n$ pari al numeri di termini diversi in $\vec{d}_j$, appartenga alla classe c_i è calcolata come

$$P(c_i|\vec{d}_j) = \frac{P(c_i)P(\vec{d}_j|c_i)}{P(\vec{d}_j)}$$

dove $P(\vec{d}_j)$ è la probabilità a priori del documento $\vec{d}_j$ e $P(c_i)$ è la probabilità che un documento scelto casualmente appartenga alla classe c_i. Stimare la

probabilità condizionata $P(\vec{d}_j|c_i)$ è problematico a causa dell'elevato numero di possibili vettori $\vec{d}_j$ (lo stesso vale anche per $P(\vec{d}_j)$). Per riuscire ad alleviare questo problema viene spesso fatta questa forte assunzione: ogni coordinata dei vettori documento, vista come variabile casuale, è statisticamente indipendente dalle altre. Formalmente, questa assunzione di indipendenza può essere tradotta come

$$P(\vec{d}_j|c_i) = \prod_{k=1}^{n} P(t_{kj}|c_i).$$

I classificatori probabilistici che fanno questa assunzione sono detti classificatori *Naïve Bayes*.
L'obiettivo da raggiungere, nella classificazione di testi, è trovare la classe più adatta al documento $\vec{d}_j$ o, in altre parole, trovare la classe *maximum a posteriori* (*MAP*) c_{MAP}:

$$\begin{aligned} c_{MAP} &= \arg\max_{c\in\mathscr{C}} P(c|t_{1j}, t_{2j}, ..., t_{nj}) \\ &= \arg\max_{c\in\mathscr{C}} \frac{P(t_{1j}, t_{2j}, ..., t_{nj}|c)P(c)}{P(t_{1j}, t_{2j}, ..., t_{nj})} \\ &= \arg\max_{c\in\mathscr{C}} P(t_{1j}, t_{2j}, ..., t_{nj}|c)P(c) \end{aligned}$$

Dalla definizione precedente e ricordando l'assunzione di indipendenza possiamo quindi dedurre il classificatore ottimo *Naïve Bayes*:

$$c_{NB} = \arg\max_{c\in\mathscr{C}} P(c)\prod_{i} P(t_{ij}|c).$$

Bisogna notare che il secondo termine è una serie di prodotti tra più probabilità condizionate ed è sufficiente che uno solo dei suoi termini sia nullo per mandare tutto a zero. Ciò succede se per una data classe c esiste almeno un'istanza che non possiede un attributo di valore t_i, quindi si avrebbe:

$$P(t_i|c) = 0 \;\Rightarrow\; P(c)\prod_{i} P(t_i|c_i) = 0$$

Tipicamente per risolvere questo problema si applica il cosiddetto *smoothing* additivo o *smoothing* di Laplace, che fissato un $\alpha > 0$ si calcola:

$$P(t_i|c) \leftarrow \frac{n_c + \alpha p}{n + \alpha}$$

in cui:

- n_c è il numero di esempi che appartengono alla classe c e contengono t_i;
- n è il numero di esempi che appartengono alla classe c;
- p è la probabilità a priori di $P(t_i|c)$.
- α è il peso dato a priori (ossia il numero di esempi "virtuali").

In certi casi si può effettuare un'altra assunzione, ossia che $\forall i, j\ P(c_i) = P(c_j)$, la probabilità delle classi è uniforme. In questo caso si può ulteriormente semplificare e scegliere l'ipotesi *Maximum likelihood* (*ML*)

$$c_{ML} = \arg\max_{c \in \mathscr{C}} P(\vec{d}_j|c).$$

Se l'assunzione fatta risultasse vera allora si avrebbe $c_{MAP} = c_{ML}$.

Vediamo ora come utilizzare questi concetti per realizzare un classificatore di testi.

Fase di apprendimento

Consideriamo $\mathscr{E}$ l'insieme degli esempi di apprendimento e $\mathscr{C}$ l'insieme delle classi (*target*) possibili.

NaïveBayesTrainer($\mathscr{E}$, $\mathscr{C}$)

1. Fase di *pre-processing*: si collezionano tutti i *token* utili che occorrono in $\mathscr{E}$ in un vocabolario $\mathscr{V}$

$$\mathscr{V} \leftarrow \forall t \in \mathscr{E} \text{ t.c. } t \text{ è un token}$$

2. Stimare i termini $P(c_j)$ e $P(t_k|c_j)$
    1. Per ogni valore di *target* $c_j \in \mathscr{C}$
    2. $\mathscr{D}_j \leftarrow$ sottoinsieme di $\mathscr{E}$ per cui il valore di *target* è c_j
    3. $P(c_j) \leftarrow \frac{|\mathscr{D}_j|}{|\mathscr{E}|}$
    4. $\mathscr{U}_j \leftarrow \bigcup_{d \in \mathscr{D}_j} d$
    5. $n \leftarrow$ numero totale di *token*, anche duplicati, in $\mathscr{U}_j$
    6. Per ogni $t_k \in \mathscr{V}$
        1. $n_k \leftarrow$ numero di volte che t_k occorre in $\mathscr{U}_j$
        2. $P(t_k|c_j) \leftarrow \frac{n_k+1}{n+|\mathscr{V}|}$ (applicazione dello *smoothing* di Laplace)

Fase di classificazione

In questa fase consideriamo di avere $\mathcal{V}$ costruito nella fase di apprendimento e un nuovo documento d da classificare.

NaïveBayesClassify(d)

1. Individuare in d tutti i termini che appartengono al vocabolario $\mathcal{V}$

   $$\mathcal{P} \leftarrow \text{tutte le posizioni in cui compare un termine } t \in \mathcal{V}$$

2. Ritornare come output la classe c_j che massimizza la probabilità a posteriori

   $$\arg\max_{c_j \in \mathcal{C}} P(c_j) \prod_{i \in \mathcal{P}} P(t_i|c_j).$$

2.5 *Support Vector Machine*

Le *Support Vector Machine* (*SVM*) costituiscono un insieme di metodi di apprendimento supervisionato per la regressione e la classificazione di *pattern*. Appartengono alla famiglia dei classificatori lineari generalizzati e sono anche noti come classificatori a massimo margine, poiché allo stesso tempo minimizzano l'errore empirico di classificazione e massimizzano il margine geometrico. Nonostante le *SVM* siano classificatori lineari, grazie al cosiddetto *kernel trick*, riescono in modo efficiente ad ottenere classificazioni non lineari. Il *kernel trick* consiste nel mappare i vettori, che rappresentano i dati, in uno spazio ad alta dimensionalità (*feature space*), nel quale i dati hanno una più alta probabilità di essere linearmente separabili (Teorema di Cover). Il trucco (*trick*) sta nel fatto che questa mappatura non viene realmente eseguita, ma grazie a delle speciali funzioni, dette funzioni *kernel*, è possibile calcolare nello spazio di origine il prodotto scalare tra i vettori come se fossero mappati nel *feature space*.

2.5.1 Definizione

Prima di dare una definizione formale alle *SVM* è necessario definire altri concetti fondamentali. Quando si parla di classificatori lineari, si intendono dei metodi che tentano di approssimare una funzionare lineare, tale da dividere lo spazio in due parti, uno che contiene gli esempi positivi e l'altro quelli negativi. Questa funzione discriminante può essere definita come segue:

$$f(\vec{x}) = \vec{w}^T \vec{x} + b$$

in cui $\vec{w}$ è il vettore dei pesi delle *feature* e b è il *bias*. Possiamo quindi definire l'iperpiano

$$\vec{x} : f(\vec{x}) = \vec{w}^T \vec{x} + b = 0$$

che divide lo spazio in due parti. Il segno della funziona discriminante $f(\vec{x})$ indica dove si trova il punto rispetto al piano. L'iperpiano appena definito viene detto *decision boundary*. All'inizio di questa sezione è stato detto che le *SVM* sono dei classificatore a massimo margine è quindi necessario definire questo concetto. Chiamiamo $\vec{x}_+$ e $\vec{x}_-$ rispettivamente il punto positivo e negativo più vicino all'iperpiano. Il margine tra un'iperpiano f e una insieme di dati D, considerando $\vec{x}_+$ e $\vec{x}_-$ equidistanti da f è definito come:

$$m_D = \frac{1}{2}\vec{w}_N^T(\vec{x}_+ - \vec{x}_-)$$

dove $\vec{w}_N$ è un vettore con stessa direzione di $\vec{w}$ e norma pari a 1. Eseguendo alcune semplificazioni si può ottenere una nuova formulazione per il margine:

$$m_D = \frac{1}{||\vec{w}||}$$

L'idea alla base delle *SVM* è di massimizzare il margine m_D ovvero, dalla definizione appena data, minimizzare $||\vec{w}||$. Quindi il problema da risolvere è un problema di ottimizzazione, e nello specifico di minimizzazione, del tipo:

$$\underset{\vec{w},b}{\text{minimize}} \ \frac{1}{2}||\vec{w}||^2$$

$$s.t. \ y_i(\vec{w}^T\vec{x} + b) \geq 1 \quad i \in 1, 2, ..., n.$$

I vincoli garantiscono che il classificatore classifichi correttamente gli esempi, assumendo i dati linearmente separabili. Questo però spesso non accade quindi è necessario modificare tali vincoli e penalizzare gli errori commessi, ottenendo così una nuova formulazione per il problema di ottimizzazione:

$$\underset{\vec{w},b}{\text{minimize}} \ \frac{1}{2}||\vec{w}||^2 + C\sum_{i=1}^{n}\xi_i$$

$$s.t. \ y_i(\vec{w}^T\vec{x} + b) \geq 1 - \xi_i \quad \xi_i \geq 0$$

Le ξ_i sono dette variabili di *slack*, e hanno la funzione di permette che ci siano errori nella classificazione (*misclassification*). Gli errori commessi hanno un costo lineare (che dipende dal parametro C) rispetto allo scostamento ξ_i dal margine. Se $\xi_i \in [0, 1]$ allora il dato sta all'interno del margine altrimenti

siamo in presenza di un errore di classificazione. Questi errori sono penalizzati dall'aggiunta del fattore C nella funzione obiettivo. Infatti questo fattore è moltiplicato per la somma degli errori commessi facendo di conseguenza aumentare il valore dell'intera funzione. La variabile C stabilisce una relazione tra la necessità di massimizzare il margine e quella di minimizzare il valore si *slack*. Questa tipologia di *Support Vector Machine* è detta *Soft Margin SVM*. Per risolvere questo problema di ottimizzazione ci viene in aiuto la teoria della dualità. Infatti, passando alla sua forma duale e utilizzando il metodo dei moltiplicatori di Langrange si ha:

$$\underset{\alpha}{\text{maximize}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \vec{x}_i^T \vec{x}_j$$

$$s.t. \sum_{i=1}^{n} y_i \alpha_i = 0 \quad \alpha_i \in [0, C]$$

Possiamo definire il vettore ottimo $\vec{w}$ dei pesi in funzione degli esempi di input:

$$\vec{w} = \sum_{i=1}^{n} y_i \alpha_i \vec{x}_i$$

Gli esempi di input $\vec{x}_i$ tali per cui $\alpha_i > 0$ sono detti vettori di supporto. Nella Figura 2.1 sono evidenziati i vettori di supporto.

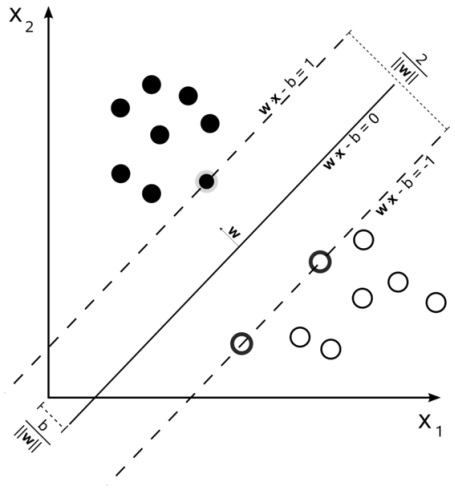


Figura 2.1: I vettori di supporto di una *SVM*.

3
Sistema di classificazione preesistente

Lo sviluppo del sistema di classificazione automatica di posta elettronica prende avvio da una precedente versione del sistema realizzata nel corso di progetti di ricerca applicata a partire dal 2007 (si vedano i lavori di Tomasoni [30], Kopliku [2], De Nardo [26] e Astegno [27]).
Nelle prossime sezioni verrà fornita una panoramica del sistema di classificazione iniziale, mettendo in evidenza lo scenario d'uso e la struttura dell'applicazione con un'analisi dettagliata delle singole componenti, valutandone pregi e difetti.

3.1 Scenario d'uso

Inizialmente il sistema di classificazione è stato sviluppato su stimolo di un'azienda Padovana che si occupa dell'installazione e della manutenzione di infrastrutture informatiche per l'ottimizzazione della gestione di piattaforme *IT*. L'azienda offre oltre all'assistenza a domicilio anche un sistema di *help desk* allo scopo di offrire un servizio attivo 24 ore su 24. Nel loro lavoro i tecnici attingono ad una *knowledge base* centralizzata e allo stesso tempo alimentano la base di conoscenza con informazioni proprie, in modo da facilitare il compito a tecnici che in futuro riscontreranno un problema simile. A questo scopo l'azienda ha adottato un sistema documentale, *OWL*, che fornisce strumenti per la raccolta, la gestione e la ricerca di documenti. *OWL* garantisce inoltre un accesso multiutente alla *knowledge base* pubblicando i documenti su un *web server Apache*. Per facilitare la ricerca e la gestione della base di conoscenza, questa è strutturata in maniera tassonomica. Nei livelli più alti si trovano le macroclassi, generalmente il nome dei produttori dei diversi servizi e man mano che si scende nella tassonomia si raggiungono nodi sempre più specifici. Tale struttura non è statica, come sottolineato da Giacoletto et al. [13], ma si

adatta all'andamento del mercato IT. Quindi è possibile col passare del tempo l'emergere di nuove classi e allo stesso tempo altre classi possono diventare obsolete. La Figura 3.1 rappresenta la struttura di un'ipotetica tassonomia, i nodi blu rappresentano le foglie.

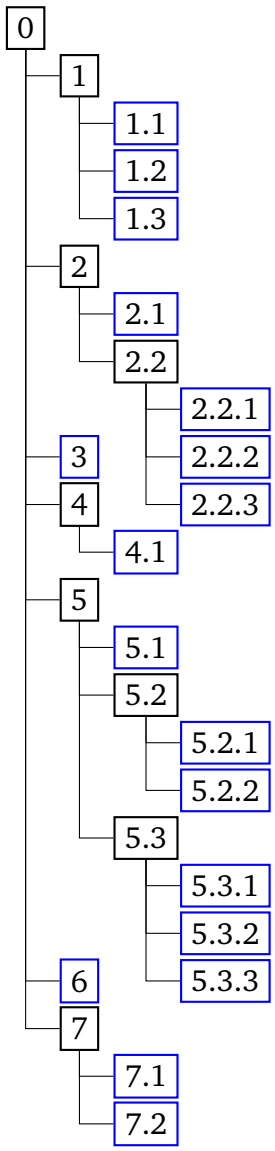


Figura 3.1: Esempio di tassonomia.

Prima dello sviluppo del sistema di classificazione i tecnici inviavano i documenti ad un'apposita casella di posta e, in un momento successivo, un addetto si occupava di categorizzarli nella base di conoscenza. Il sistema di classificazione quindi è stato utilizzato dall'azienda come strumento di supporto per

l'addetto alla categorizzazione.

3.1.1 Prima versione

Nella sua prima versione, (Tomasoni [30]), il sistema era composto da due componenti principali: il classificatore, che si occupava di classificare i messaggi di posta elettronica, e un servizio *web* che presentava i risultati all'addetto. Il flusso del sistema era il seguente:

Componente di classificazione

1. Controllo della casella *IMAP* e lettura di eventuali nuovi messaggi;
2. Classificazione delle nuove *e-mail* (se presenti);
3. Salvataggio delle *e-mail* classificate in un *database*;
4. Vai a (1).

A fine giornata, ad un orario prefissato, il classificatore veniva fermato per dare la possibilità di apprenderne uno nuovo grazie alle nuove *e-mail* che sono state confermate dall'addetto.

Componente di presentazione (*feedback*)

1. Controllo della presenza di nuovi messaggi nel *database*;
2. Presentazione dei messaggi classificati in una pagina *web*;

Ogni qualvolta l'addetto classifica le nuove *e-mail*, presentate dalla componente di *feedback*, queste vengono messe a disposizione del *learner*.

3.1.2 Seconda versione

Successivamente l'azienda cominciò ad utilizzare un software per la gestione dei *ticket*, *OTRS*, e nel suo lavoro De Nardo [26] modificò il flusso del sistema interfacciandosi appunto al sistema *OTRS*. Il nuovo flusso di dati durante la classificazione è riassunto in Figura 3.2. Come si può notare il componente di classificazione non ha subito grosse variazioni rispetto alla versione precedente, se non per l'invio dei risultati in un file *XML* al sistema di *ticketing*. Il componente di *feedback*, invece, non è più caratterizzato dal servizio *web* ma da uno scambio di file *XML* tra il classificatore e il sistema *OTRS*. Una volta

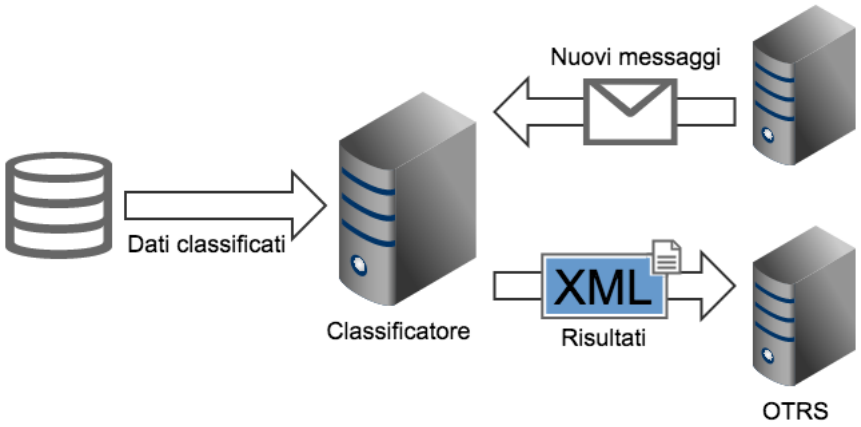


Figura 3.2: Flusso dei dati nel vecchio sistema durante la classificazione.

ricevuto il file in modo asincrono (tramite una *HTTP get*), viene valutato dall'addetto il quale ritornerà a sua volta un nuovo file *XML* con le classificazioni corrette dei nuovi messaggi. In Figura 3.3 viene riassunto questo secondo flusso di dati. I file *XML* per poter essere letti venivano pubblicati via *HTTP* ad un indirizzo pubblico, del tipo:

http://<service_address>

Inviare una richiesta a questo indirizzo significava ottenere come risposta l'ultimo lotto di *e-mail* prodotto. In caso non ci fossero lotti disponibili veniva ritornato l'errore *HTTP 404:Not Found*. Era possibile inoltre richiedere un lotto specifico indicandolo nell'indirizzo:

http://<service_address?lotto=<number>

Oltre al protocollo è stata definita anche la struttura dei file *XML*. Quest'ultimi contenevano diverse informazioni utili, oltre alle proposte di classificazione per i lotti, come il numero del lotto e la data di creazione. Di seguito viene mostrato un esempio di file *XML*.

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <classificatore>
3     <lotto>
4       <data>DD/MM/YYYY HH:MM:SS</data>
5       <numero>numero identificativo lotto</numero>
6     </lotto>
7     <e-mail>
8       <uid>uid assegnato da IMAP</uid>
9       <mid>ID assegnato da IMAP</mid>
10      <coda>
```

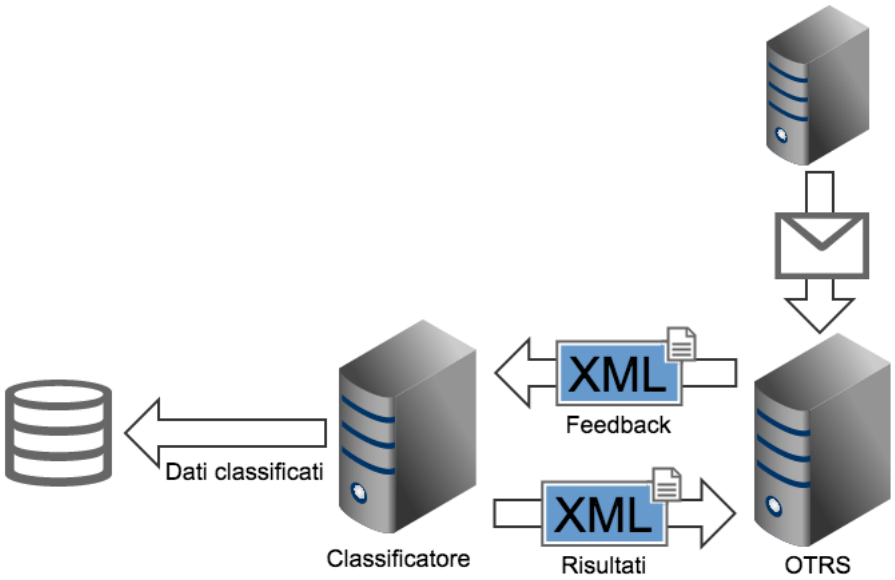


Figura 3.3: Flusso dei dati nel vecchio sistema per la gestione dei *feedback*.

```
11          <id1> id classe 1</id1>
12          <id2> id classe 2</id2>
13          <id3> id classe 3</id3>
14          <id4> id classe 4</id4>
15          <id5> id classe 5</id5>
16        </coda>
17      </e-mail>
18      <e-mail>
19        <uid>uid assegnato da IMAP</uid>
20        <mid>ID assegnato da IMAP</mid>
21        <coda>
22          <id1> id classe 1</id1>
23          <id2> id classe 2</id2>
24          <id3> id classe 3</id3>
25          <id4> id classe 4</id4>
26          <id5> id classe 5</id5>
27        </coda>
28    </e-mail>
29  </classificatore>
```

3.1.3 Terza versione

In seguito, in base alle esigenze di una nuova azienda, oltre ad altri cambiamenti e miglioramenti, la struttura dei file *XML* fu leggermente modificata (Astegno [27]) aggiungendo, ai già presenti *uid* e *mid*, un nuovo identificativo, l'*id_activity* con lo scopo di identificare l' *e-mail* all'interno del lotto. La nuova struttura viene presentata qui di seguito.

```
1  <classificatore>
2     <lotto>
3       <data>DD/MM/YYYY HH:MM:SS</data>
4       <numero>numero identificativo lotto</numero>
5     </lotto>
6     <e-mail>
7       <idActivity>id_activity azienda</idActivity>
8       <uid>uid assegnato da IMAP</uid>
9       <mid>ID assegnato da IMAP</mid>
10      <coda>
11        <id1> id classe 1</id1>
12        <id2> id classe 2</id2>
13        <id3> id classe 3</id3>
14        <id4> id classe 4</id4>
15        <id5> id classe 5</id5>
16      </coda>
17    </e-mail>
18 </classificatore>
```

Il cambiamento del protocollo in questo nuovo contesto ha portato alla modifica della lettura dei *feedback*. Per riuscire ad ottenere la lista di file, corrispondenti a messaggi di posta elettronica non ancora controllati, si deve fare una richiesta *HTTP* di questo tipo:

http://<service_address>?listFiles=true

oppure è possibile chiedere uno specifico file di *feedback* grazie al suo *id*:

http://<service_address>?getFile=<id_activity>.

Infine l'ultima modalità prevista consente di modificare il nome del file *XML*, specificato attraverso il suo *id_ activity*, introducendo il suffisso *DONE* per segnalare che la lettura del file è avvenuta con successo:

http://<service_address>?setDone=<id_activity>.

3.2 Architettura generale

Nonostante le diverse necessità e il cambiamento di alcuni protocolli, l'architettura generale del sistema è rimasta pressoché invariata. All'interno del sistema è possibile individuare quattro componenti principali:

- Componente di memorizzazione: caratterizzato da un *DBMS* (*DataBase Management System*) in cui vengono memorizzati i dati. Nello specifico viene utilizzato un database *MySQL* con due tabelle: nella prima sono memorizzate le *e-mail* classificate dall'utente e che quindi possono essere utilizzate per il *training* del classificatore, nella seconda invece ci sono i messaggi ancora da confermare;
- *Server e-mail*: *server*, nello specifico *IMAP*, che rappresenta la fonte di messaggi che arrivano in ingresso al classificatore;
- Componente di classificazione: che rappresenta il cuore dell'applicazione, ovvero il classificatore, che contiene al suo interno le componenti di *Machine Learning* necessarie. Oltre alle componenti di *training* e *classification* ci sono altre due componenti di supporto per il caricamento dei dati dal *DBMS* e per la gestione dei *feedback* dell'utente. Questo componente viene descritto in dettaglio nella sezione 3.2.1;
- Sistema aziendale: che consente l'integrazione dei risultati (tramite *OTRS*) di classificazione all'interno dell'azienda.

3.2.1 Struttura del classificatore

Il classificatore è caratterizzato principalmente da tre *thread*: *thread* di *training*, *thread* di classificazione e di *feedback*. Oltre a questi c'è poi la *Servlet* che presenta i risultati della classificazione all'utente. Tutti questi componenti accedono a dei dati condivisi ed è pertanto necessaria una gestione controllata degli accessi alle risorse.

Thread di *training*

Questo *thread*, essendo molto complesso e potenzialmente molto oneroso sia in termini di calcolo della macchina sia in termini di tempo, viene eseguito solo una volta al giorno nelle ore notturne in cui il traffico di *e-mail* in ingresso è minimo. Il *thread* esegue le seguenti operazioni:

1. Viene acquisito il *lock* sulla base di dati;

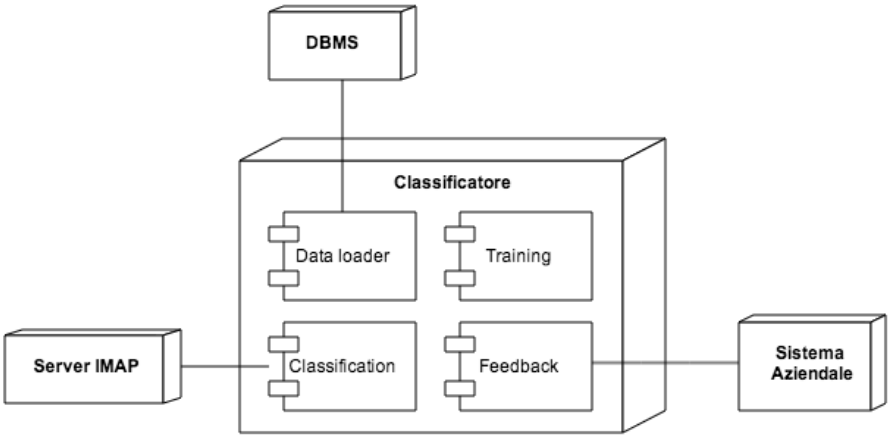


Figura 3.4: Architettura generale del sistema.

2. Viene controllata se esistono nuovi dati supervisionati all'interno della base di dati;
3. Viene rilasciato il *lock* sulla base di dati;
4. In caso non ci siano nuovi dati:
    1. Il *thread* termina / va in *sleep*;
5. In caso ce ne siano:
    1. I dati vengono passati al *trainer* che produce un nuovo classificatore;
    2. Il nuovo classificatore viene salvato;
    3. Il *thread* termina / va in *sleep*;

La Figura 3.5 illustra il flusso di esecuzione del *thread*. Come si può notare come prima, e anche ultima, operazione viene controllato se il *training* è attivo, in caso affermativo allora le operazioni appena descritte vengono eseguite, in caso contrario il *thread* termina immediatamente.

3.2.2 *Thread* di classificazione

A differenza del *thread* appena descritto, quello di classificazione è continuamente attivo e viene eseguito ogni 5 minuti, in modo che si avvicini il più

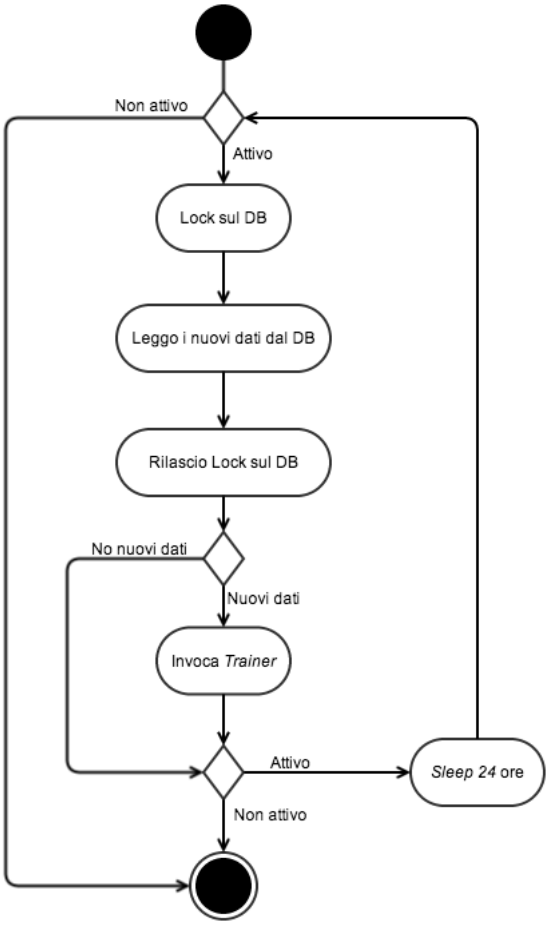


Figura 3.5: Flusso di esecuzione del *thread* di *training*.

possibile a un servizio *on-demand*. Il flusso di esecuzione è illustrato in Figura 3.6, mentre di seguito vengono descritte le varie operazioni:

1. Vengono prelevate le nuove *e-mail* dalla casella *IMAP*;
2. Se non esistono nuove e-mail:
   - Il *thread* termina / va in sleep;
3. Se esistono nuove *e-mail*:
   1. Carica il classificatore generato nella fase di *training*;
   2. Classifica i nuovi messaggi;

3. Crea il file *XML* contenente il lotto appena classificato;
4. Il *thread* termina / va in sleep;

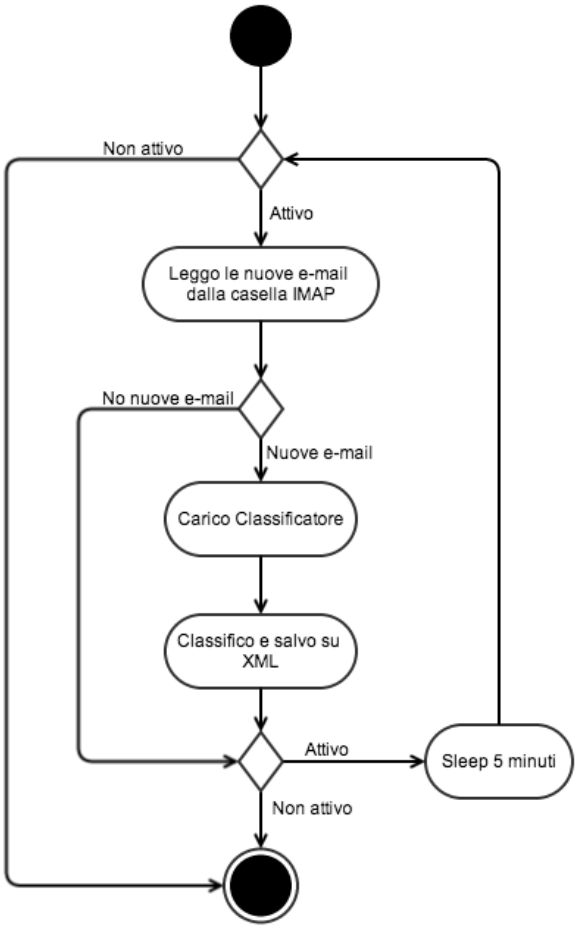


Figura 3.6: Flusso di esecuzione del *thread* di classificazione.

3.2.3 *Servlet* per la presentazione dei risultati

Il *Web Application Server* adottato è *Apache Tomcat* che consente la gestione, in modo visuale, della *servlet* per la presentazione dei risultati della classificazione. La *Servlet* prende i risultati direttamente dalla *directory* in cui vengono salvati i file *XML* dal *thread* di classificazione, e li rende disponibili in indirizzi specifici (si veda sezione 3.1.3).

3.2.4 *Thread* di *feedback*

L'algoritmo per la lettura del feedback sfrutta la definizione del protocollo di comunicazione descritto in precedenza. Il flusso del *thread* è il seguente:

1. Viene letto l'ultimo lotto prodotto (tramite protocollo *HTTP*) e decodificato il numero corrispondente;
2. Lettura di tutti i lotti compresi tra l'ultimo letto e il lotto appena ricevuto, in questo modo vengono letti tutti i lotti nuovi;
3. Memorizzazione del numero dell'ultimo lotto che verrà utilizzato all'avvio successivo come descritto in (2).

3.3 Tecnologie utilizzate

In questa sezione verranno discusse le tecnologie utilizzate per la creazione del sistema preesistente dando particolare enfasi alla libreria di *Machine Learning*. Il sistema è interamente sviluppato in Java ed è pensato per funzionare in ambiente *Linux*.

3.3.1 *DBMS*: *MySQL*

MySQL (http://www.mysql.it/), è un *Relational database management system* (*RDBMS*), composto da un *client* con interfaccia a riga di comando e un *server*, entrambi disponibili sia per sistemi *Unix* o *Unix-like* come *GNU/Linux* che per *Windows*, anche se prevale un suo utilizzo in ambito *Unix*.

3.3.2 Gestione dei file *XML*: *SAX*

SAX (http://www.saxproject.org) è un insieme di interfacce per il *parsing* e l'elaborazione di file *XML*. Dell'interfaccia sono disponibili differenti implementazioni, di cui almeno una deve essere installata per permettere l'utilizzo delle interfacce. *SAX* appartiene alla categorie di interfacce basate su eventi in modo da garantire maggiore efficienza. Il *parsing* del file avviene quindi generando una serie di eventi, che l'applicazione può intercettare e gestire in modo analogo a quanto avviene per le interfacce grafiche.

3.3.3 *Machine Learning*: *Mallet*

Mallet è una *suite* Java pensata per applicazioni di apprendimento automatico ai testi. Al suo interno contiene diversi pacchetti tra cui sofisticati *tool* per la

classificazione di documenti: efficienti metodi per convertire testo in vettori di *feature*, algoritmi di apprendimento automatico come *Naïve Bayes*, *MaxEntropy* e Alberi di decisione. Inoltre offre diversi strumenti per la valutazione di questi algoritmi.
All'interno del sistema questa libreria rappresenta una parte fondamentale poiché sia la *Feature selection*, che tutta la fase di classicazione, *training* compreso, è gestita da *Mallet*. L'unica eccezione è fatta per la parte, sviluppata da Astegno [27], che usa le *Support Vector Machine*, in cui è utilizzata la libreria *LibSVM* [6].

4
Il nuovo sistema

A partire da questo capitolo verrà presentato il nuovo scenario operativo e verranno definiti i requisiti sorti dalle esigenze dell'azienda coinvolta. Saranno inoltre analizzate le diverse scelte che sono state prese durante l'implementazione

4.1 Scenario operativo

4.1.1 L'azienda coinvolta

L'azienda coinvolta nell'attività di tesi si occupa di offrire soluzioni su misura per ottimizzare la gestione di piattaforme IT alle aziende del manifatturiero e dei servizi, agli enti pubblici, alle strutture sanitarie e alle organizzazioni *no-profit*. L'azienda offre oltre all'assistenza a domicilio anche un sistema di *help desk* allo scopo di offrire un servizio attivo 24 ore su 24. Lo scenario operativo è lo stesso che è descritto nella sezione 3.1.

4.1.2 Tassonomia

Nel corso degli anni il mondo dell'*IT* si è evoluto e con esso anche la tassonomia di classi della *knowledge base* aziendale. In Figura 4.1 è illustrata la tassonomia (per motivi di riservatezza i veri nomi delle classi sono stati sostituiti da valori numerici). Si può notare che il numero di classi coinvolte è notevole e questo, come vedremo successivamente, porterà a dover fare alcune considerazioni sia sull'algoritmo che su altri aspetti della classificazione. Generalmente al primo livello della tassonomia si trovano i nomi dei produttori dei servizi che l'azienda offre. Man mano che si scende in profondità lungo la tassonomia si incontrano nodi sempre più specifici sino ad arrivare alle versioni dei software.
Sulla base di questa tassonomia è possibile dare una stima sulla distribuzione dei messaggi di posta in ingresso. Dalla Tabella 4.1 si può subito evincere

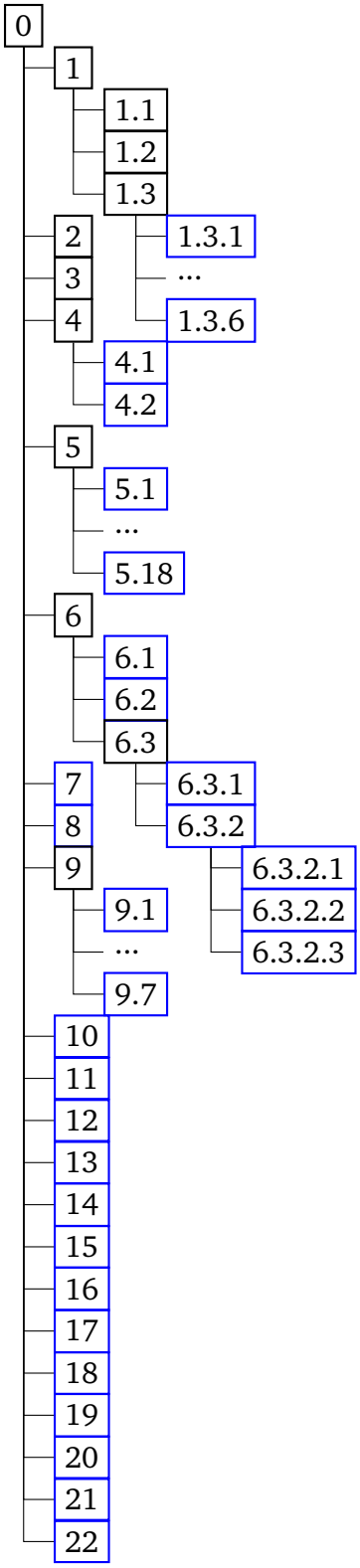


Figura 4.1: Tassonomia della *knowledge base*.

che non c'è una distribuzione uniforme, anzi, le *e-mail* tendono a concentrarsi maggiormente su tre classi particolari.

Classe	**% Messaggi**
1	10.6
2	21.2
3	4.9
4	4.9
5	2.9
6	20.2
7	2.6
8	2.7
9	20.6
10	2.4
11	2.1
12	1.9
13	0.6
14	0.3
15	0.1
16	0.3
17	0.1
18	0
19	0.4
20	0.4
21	0.3
22	0.5

Tabella 4.1: Distribuzione delle *e-mail* su un *dataset* di 755 *e-mail*.

Essendo la distribuzione così disomogenea si sono fatte alcune considerazioni riguardo alla struttura. E' possibile notare come ben 10 classi siano al di sotto dello 0.7% , inoltre, la classe 18 sembra essere ormai obsoleta. Per questi motivi si è deciso di eliminare la classe 18 e di creare un nodo padre per tutte quelle che hanno una probabilità inferiore all'1%. La nuova tassonomia è presentata in Figura 4.2. Il nuovo nodo è contrassegnato dal numero 23. La Tabella 4.2 mostra le stime sulla distribuzione dei messaggi al primo livello con la nuova tassonomia.

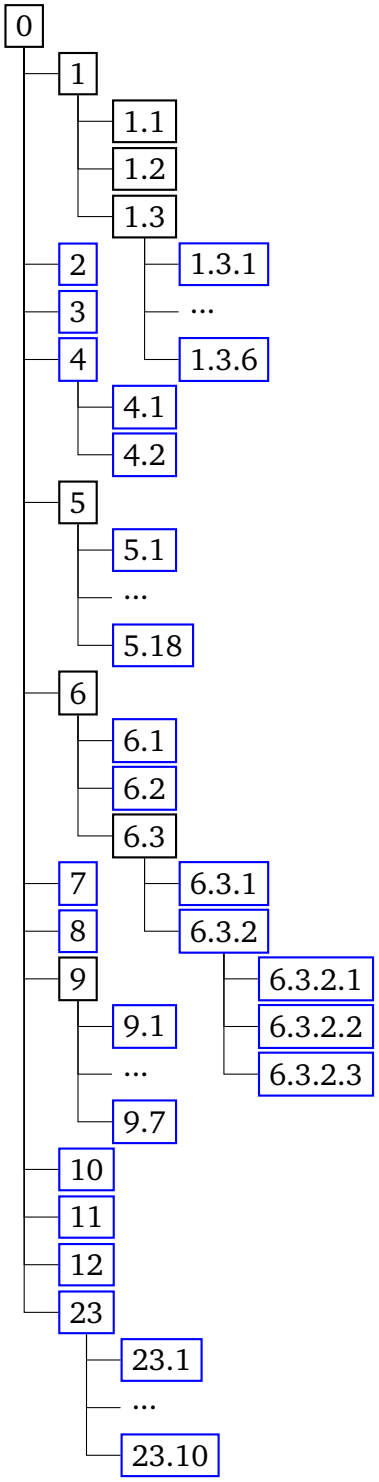


Figura 4.2: Nuova tassonomia.

Classe	% Messaggi
1	10.6
2	21.2
3	4.9
4	4.9
5	2.9
6	20.2
7	2.6
8	2.7
9	20.6
10	2.4
11	2.1
12	1.9
23	3.0

Tabella 4.2: Distribuzione delle *e-mail* con la nuova tassonomia.

4.2 Flusso dei dati

Con l'abbandono dell'interfacciamento con il sistema *OTRS* si è dovuto pensare ad un nuovo flusso dei dati. Una volta che il classificatore ha terminato di classificare nuovi messaggi, pone quest'ultimi nel *DB* dove la *servlet* in modo asincrono li andrà a leggere per poterli presentare all'utente. Il flusso completo è rappresentato in Figura 4.3. Essendo la pagina *web* pubblica, all'interno del sistema aziendale, sarà necessario gestire la multi utenza, è possibile infatti che più utenti diano il loro *feedback* per una stessa serie di messaggi. È altresì possibile che la classificazione sia discordante a causa del fatto che una stessa *e-mail* può appartenere a più classi.

4.3 Requisiti del nuovo sistema

Prima di iniziare il lavoro di tesi è stato necessario definire assieme all'azienda i nuovi requisiti sulla base delle loro esigenze.
Per prima cosa l'azienda ha cambiato il sistema *OTRS* per la gestione dei *ticket* ed è stata persa tutta la parte dedicata alla gestione dei file *XML* secondo il protocollo descritto nel Capitolo 3. Per questo motivo si è deciso di tornare ad adottare il sistema di gestione dei *feedback* utilizzato prima che l'azienda utilizzasse l'*OTRS*, ossia, una *Servlet* che presenta i risultati della classificazione in una pagina pubblica e accessibili a tutti i dipendenti autorizzati. L'azienda

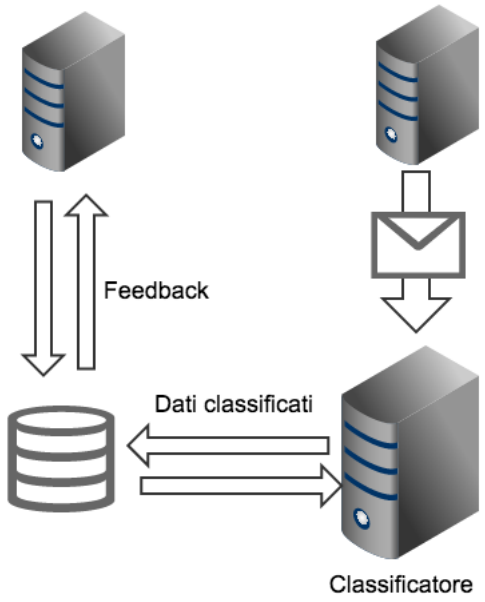


Figura 4.3: Flusso dei dati nel nuovo sistema.

ha inoltre richiesto la possibilità da parte del sistema automatico di riconoscere quando una classe sta diventando obsoleta e quando invece una classe molto ampia contiene al suo interno delle sottoclassi. In questo modo la modifica della tassonomia è gestita in maniera semi automatica sgravando parte del lavoro agli addetti. Di seguito sono elencati, descritti e divisi per tipologia tutti i requisiti del nuovo sistema.

Requisiti Funzionali [F]

Obbligatori [O]

- **FO_01**: Per il funzionamento del sistema è necessario fornire una tassonomia di classi.
- **FO_02**: La relazione tra una classe padre e le classi figlio nella tassonomia è di tipo inclusivo.
- **FO_03**: Per il funzionamento del sistema è necessario fornire un insieme di documenti pre-classificati sulla base della tassonomia del requisito FO_01.
- **FO_04**: Uno stesso documento può appartenere a più classi.

- **FO_05**: I documenti devono essere in formato testuale.
- **FO_06**: I documenti devono essere trasformati nel formato adeguato per il classificatore.
- **FO_07**: Il sistema apprende il modello sulla base dei documenti pre-classificati (FO_03).
- **FO_08**: Il sistema salva il proprio modello in un file.
- **FO_09**: Il sistema è in grado di interfacciarsi a una casella di posta elettronica dedicata.
- **FO_10**: Il sistema classifica sulla base dell'intera tassonomia (oltre il primo livello).
- **FO_11**: Il sistema ritorna il *ranking* ordinato delle classi per ogni documento classificato.
- **FO_12**: Il sistema presenta i risultati in una *servlet* ad un indirizzo pubblico.
- **FO_13**: L'utente deve fornire il *feedback* tramite la *servlet*.
- **FO_14**: Il sistema deve poter ricevere i *feedback* degli utenti.
- **FO_15**: Il sistema deve essere presente su una macchina all'interno dell'azienda.
- **FO_16**: Il sistema è in grado di aggiornare il proprio modello sulla base dei nuovi documenti classificati dagli utenti.
- **FO_17**: Il sistema deve aggiornarsi ad ogni nuovo documento classificato dall'utente.

Desiderabili [D]

- **FD_01**: La tassonomia può essere modificata dall'utente in modo dinamico.
- **FD_02**: Il sistema può supportare diverse lingue.
- **FD_03**: Il sistema può gestire l'introduzione di regole associative fissate a priori.

- **FD_04**: Il sistema può gestire i *feedback* di più utenti contemporaneamente.
- **FD_05**: Il sistema può gestire *feedback* diversi per uno stesso documento.
- **FD_06**: Il sistema può gestire i *feedback* di più utenti contemporaneamente.
- **FD_07**: Il sistema può accorgersi se una classe diventa obsoleta.
- **FD_08**: Il sistema può accorgersi se una classe può essere divisa in sottoclassi.
- **FD_09**: Il sistema può gestire i file allegati.
- **FD_10**: Il sistema dovrebbe presentare all'utente la lista delle classi potenzialmente obsolete.
- **FD_11**: Il sistema dovrebbe presentare all'utente la lista delle classi potenzialmente emergenti.
- **FD_12**: Il sistema dovrebbe presentare per ogni classe potenzialmente emergente una lista di termini associati.

Requisiti di Qualità [Q]

Desiderabili [D]

- **QD_01**: Introduzione di una componente per individuare e evidenziare termini simili o correlati.
- **QD_02**: Il sistema dovrebbe prevedere una componente per selezionare i termini più significativi nel testo.

4.4 L'ambiente di Test

Per poter valutare le prestazioni del classificatore è indispensabile creare un ambiente di test adeguato. A questo scopo è stato creato un progetto che esegue il classificatore e lo valuta secondo alcune metriche. Le metriche adottate principalmente sono due:

1. *Error Score (ES)*: l'errore viene valutato in base alla posizione che assume nel *ranking* la classificazione corretta. Ad esempio, se la classe corretta appare in posizione 0 allora non c'è alcun errore e l' *error score* vale 0. Se invece la classificazione corretta appare in posizione 3 allora si ha un *error score* pari a 3. L' *error score* corrisponde al *Classification gap* descritto nella Sezione 2.2.1.

2. *Tree distance (TD)*: questo valore rappresenta la distanza della classe assegnata rispetto alla classe corretta nella tassonomia. La distanza è calcolata sul numero di archi che vanno da una classe ad un'altra. Ovviamente se la classe è corretta la distanza vale 0, mentre se l'errore avviene tra una classe padre e una classe figlio l'errore è pari a 1. La differenza tra questo tipo di errore e il precedente è che questo tiene conto della vicinanza tra le classi.

5
Fase di indicizzazione

In questo capitolo verranno analizzate le peculiarità del testo delle *e-mail* in modo da estrapolare il maggior numero di informazioni utili per poter ottimizzare questa fase. La conoscenza del dominio applicativo giocherà un ruolo molto importante all'interno di questa fase.

5.1 Il *bug* del sistema iniziale

Prima di entrare nel dettaglio dell'analisi dei messaggi e delle fasi successive è necessario sottolineare che il classificatore di partenza, a causa dell'aggiornamento delle librerie (in particolar modo *Mallet*), non funzionava correttamente. Questo malfunzionamento era dovuto ad un *bug* presente nel calcolo dell'*Information Gain*, tecnica utilizzata per la *feature selection*. Il metodo, che avrebbe dovuto restituire la lista di *feature* ordinata per *information gain*, assegnava ad ogni *feature* un valore pari a 0 e conseguentemente l'ordinamento non aveva effetto. Questo comprometteva in maniera significativa l'esito della classificazione. Per questo motivo prima di procedere con qualsiasi tipo di analisi e test si è dovuto sostituire la componente sviluppandone una nuova.

5.2 Configurazione dell'ambiente di Test

I test descritti nelle prossime sezioni sono stati eseguiti con la tecnica del *k-cross fold validation* con $k = 10$, su un *dataset* di 755 *e-mail*. Per ogni classe vengono considerate le prime 100 *feature* ordinate in base al valore di *Information Gain*. Ad ogni *feature* è associato un peso pari alla frequenza (*Feature Frequency*). La configurazione appena descritta vale per tutti i test a meno che non sia diversamente specificato.

5.3 Analisi dei messaggi

I messaggi di posta elettronica in ingresso alla casella dedicata, arrivano da tecnici specializzati con l'intento di ampliare la *knowledge base* aziendale. Nonostante non siano state definite regole o *best practice* per quanto riguarda la stesura delle *e-mail* è possibile notare alcune caratteristiche comuni alla maggior parte di esse. Una delle caratteristiche più importanti è la rilevanza dell'oggetto: al suo interno l'oggetto del messaggio contiene termini che riguardano in maniera diretta (ma spesso anche indiretta) la classe di appartenenza. Questo fatto del resto non dovrebbe essere una novità visto che l'oggetto è pensato per dare un'idea del contenuto del corpo utilizzando poche parole. Un'altra particolarità è l'uso non controllato della lingua italiana mista alla lingua inglese: questo spesso sembra essere dovuto a "copia-incolla" da siti specializzati. Per lo stesso motivo gli stessi indirizzi *internet* compaiono non di rado all'interno del corpo dei messaggi. Infine, si è notato che il testo è ricco di termini tecnologici, legati al dominio aziendale, e questi spesso non compaiono all'interno di dizionari, sarà quindi necessario provvedere che non vengano trascurati.

5.4 *Pre-processing*

In Figura 5.1 è raffigurato il flusso della funzione che trasforma il testo grezzo in un insieme di *token* del sistema precedente. La figura illustra il flusso per un solo termine, tale funzione viene ripetuta per tutti i termini presenti nel testo del messaggio.
Nella rappresentazione finale ad ogni parola viene affiancato il termine corretto, se necessario, e successivamente ridotto a *stem*. Questo processo avviene indistintamente sia per l'oggetto che per il corpo del messaggio. In Figura 5.2 è possibile vedere le *perfomance* del classificatore iniziale con il vecchio metodo di *tokenizzazione*. Il test è stato eseguito con *Naïve Bayes* sul primo livello della tassonomia con *error score* pari a 0,54.

5.4.1 Nuovo dizionario

Il sistema precedente era pensato per gestire messaggi esclusivamente in lingua italiana. L'azienda coinvolta nel lavoro di tesi però riceve *e-mail* sia in lingua italiana che in lingua inglese, questo ha portato all'adozione di un nuovo dizionario. Oltre all'introduzione del dizionario inglese con oltre 300.000 parole si è ampliato quello italiano, anch'esso con circa 300.000 parole. Questi due dizionari contengono la quasi totalità delle parole rispettivamente in

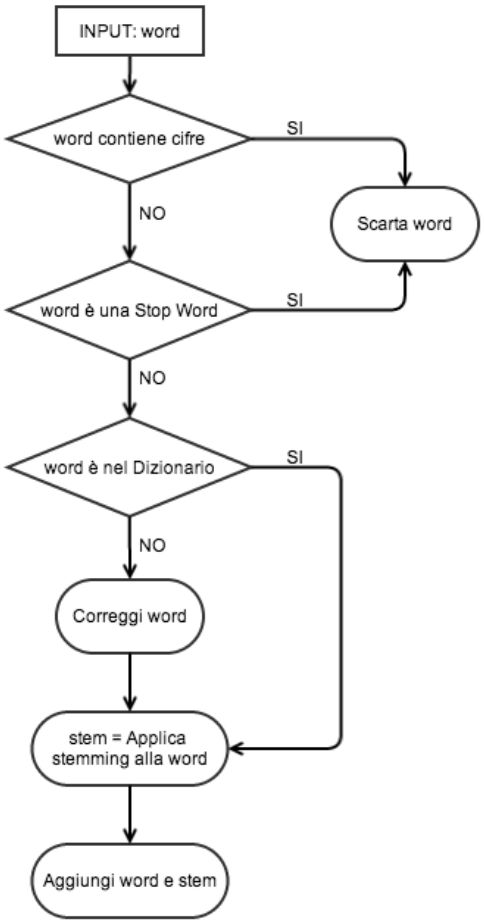


Figura 5.1: Flusso della funzione di *tokenizzazione* del sistema precedente.

lingua inglese e italiana però molti termini usati frequentemente in ambito informatico non sono presenti. Per questo motivo è stato aggiunto un terzo dizionario contenente termini e acronimi di natura tecnologica.

5.4.2 *Stop Words*

Il sistema adottava una lista di *stop words* italiane per l'eliminazione delle parole più comuni e ininfluenti ai fini della classificazione. Con l'introduzione del dizionario inglese, e la conseguente gestione di una seconda lingua, è stato necessario aggiungere una nuova lista di *stop words* che contenesse i termini inglesi più comuni come articoli, congiunzioni e preposizioni. All'interno del sistema precedente era presente una terza lista contenente *tag HTML* ed è stata mantenuta anche nel sistema attuale.

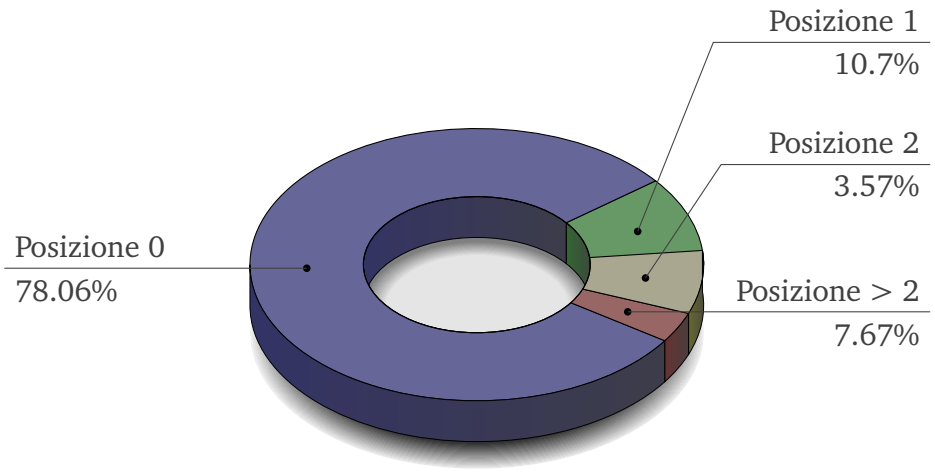


Figura 5.2: *Performance* del sistema precedente (*Naïve Bayes*).

5.4.3 Riconoscimento della lingua

L'aggiunta di un nuovo dizionario e delle corrispondenti *stop words* comporta nuove problematiche relative allo *stemming*. Infatti, se venisse applicato uno *stemmer* italiano su parole in lingua inglese si avrebbero delle storpiature senza senso poiché la struttura delle parole inglesi differisce enormemente da quelle italiane. Si è dovuto quindi aggiungere un secondo *stemmer* per supportare le parole in lingua inglese. Nasce però un secondo problema: quando applicare lo *stemmer* italiano anziché quello inglese? Per ovviare a questo secondo problema si è dovuto ricorrere ad uno strumento di *Language detection*. In questo modo il testo di un messaggio viene prima analizzato dal *Language detector* e successivamente, in base alla lingua rilevata, viene utilizzato lo *stemmer* corrispondente. Lo strumento utilizzato è una libreria Java gratuita (licenza Apache 2.0) chiamata *Language Detection Library for Java*, la quale supporta fino a 50 lingue (vedi [38]).

5.4.4 Gestione dei valori numerici

Come si può notare dal flusso in Figura 5.1, i termini contenenti cifre venivano semplicemente scartate. Essendo che il nuovo dominio applicativo riguarda l'informatica e il settore *IT* in genere, l'eliminare senza alcun tipo di controllo tutte le cifre può portare a perdita di informazioni importanti (ad es. i numeri di versione). È stata quindi rimossa la parte di algoritmo che ignora a priori qualsiasi termine contenente delle cifre e introdotto un piccolo dizionario che contiene termini di natura tecnologica, tra i quali numeri (massimo 4 cifre) e termini contenenti numeri ricorrenti nel settore dell'*IT*.

5.4.5 Parti considerate

Nel Capitolo 2 è stato detto che il formato standard per le *e-mail* è il formato *MIME*. Al suo interno contiene le diverse parti che costituiscono un messaggio di posta elettronica, però non tutte verranno prese in considerazione poiché alcune di esse non contengono informazioni utili ai fini della classificazione. Infatti, parti come mittente e destinatario non hanno alcuni valore informativo essendo indirizzi *e-mail* dei tecnici e della casella dedicata al classificatore. Le parti considerate sono quindi: oggetto, corpo e allegati.

5.4.6 L'importanza dell'oggetto

Per quanto detto nella Sezione 5.3, l'oggetto rappresenta una parte molto importante del messaggio e proprio per questo vale la pena dargli maggior peso. Per far questo si è pensato di aggiungere un coefficiente moltiplicativo ai pesi delle *feature* presenti all'interno della rappresentazione finale, in modo che i termini in esso contenuti abbiano una frequenza maggiore. Di seguito viene riportata la tabella in cui sono visualizzati i risultati dei test al variare del coefficiente moltiplicativo. I test sono stati eseguiti con *Naïve Bayes* al primo livello della tassonomia.
La tabella mostra come le percentuali nella posizione 0 aumentino all'aumentare del valore del coefficiente. Le *performance* poi si stabilizzano attorno all'intervallo $[7, 10]$. Si può inoltre notare che nonostante in prima posizione si raggiungano migliori risultati aumentando il il valore del coefficiente, la percentuale cumulativa sulle prime 5 posizioni continua a diminuire, avendo raggiunto il massimo al valore 5. Questo può indicare che alcune *e-mail* hanno nell'oggetto termini ambigui o non del tutto correlati alla classe di appartenenza e l'aumento del loro peso associato causa un degrado delle prestazioni.

Upper case multiplication

Aggiungere un coefficiente moltiplicativo a tutto il testo dell'oggetto può portare all'aumento della frequenza di termini che non hanno alcuna utilità nella classificazione. Durante l'analisi dei messaggi si è notato che spesso i termini più rilevanti nell'oggetto sono scritti in maiuscolo. Nella precedente versione del sistema il testo veniva da subito normalizzato in minuscolo e successivamente venivano applicate le varie fasi di *pre-processing*. Da quanto appena detto, nel nuovo sistema, la normalizzazione viene preceduta da una ripetizione dei termini in maiuscolo presenti all'interno dell'oggetto. In Figura 5.3

	Coefficiente				
Posizione	2	3	4	5	6
0	87.05 %	89.53 %	90.39 %	90.7 %	90.16
1	7.44 %	6.28 %	6.2 %	5.66 %	6.43 %
2	2.4 %	1.47 %	1.47 %	1.47 %	1.78 %
3	0.7 %	0.62 %	0.47 %	0.62 %	0.23 %
4	0.39 %	0.39 %	0.08 %	0.23 %	0.39 %
5	0.85 %	0.62 %	0.54 %	0.62 %	0.2 %
> 5	1.16 %	1.19 %	0.85 %	0.54	0.64 %

	Coefficiente			
Posizione	7	8	9	10
0	91.78 %	91.24 %	93.33 %	93.1 %
1	5.19 %	4.81 %	3.72 %	4.11 %
2	1.63 %	1.24 %	0.93 %	0.54 %
3	0.16 %	0.85 %	0.31 %	0.23 %
4	0.47 %	0.7 %	0.16 %	0.7 %
5	0.08 %	0.39 %	0.31 %	0.08 %
> 5	0.7 %	0.78 %	1.24 %	1.24

Tabella 5.1: *Performance* del classificatore ripetendo le *feature* dell'oggetto (*ES* medio 0.19).

sono mostrate le performance del classificatore (*NB*) con questa modifica (con una ripetizione dell'oggetto di 10 volte), al primo livello della tassonomia.

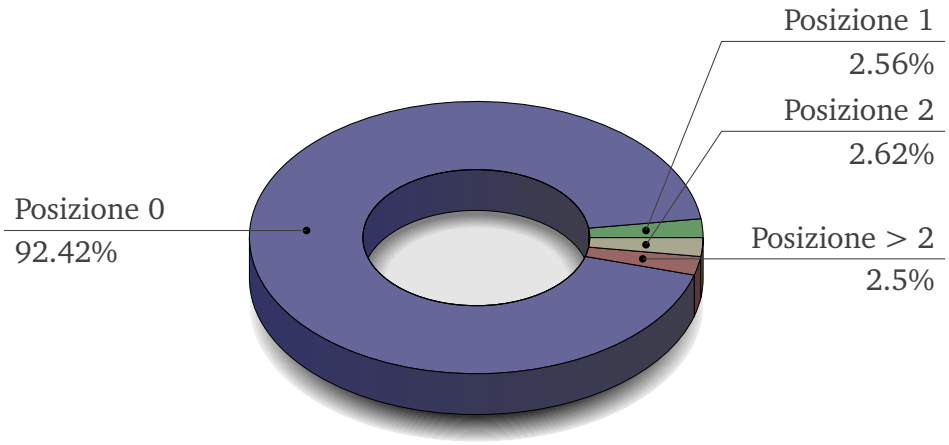


Figura 5.3: Classificatore con *Upper case multiplication* (2x).

In questo caso all'aumento delle volte per cui vengono ripetute le parole in maiuscolo non vi è un aumento delle prestazioni, e il test sopracitato è stato effettuato ripetendo tali parole 2 volte.

5.4.7 Gestione degli allegati

Il sistema preesistente non aveva integrato alcun meccanismo per la gestione degli allegati. Analizzando i messaggi dell' azienda si è notato che molti contengono allegati e si è deciso di creare dei filtri che li gestissero. I formati che il sistema è in grado di gestire sono i seguenti:

- *Microsoft Word*: doc e docx;
- *Microsoft Excel*: xls e xlsx;
- *Microsoft Power Point*: ppt e ppx;
- *Open office*: odt, odp e ods;
- Testuali: *PDF*, rtf e txt;
- Linguaggi di *MarkUp*: *XML* e *HTML*;
- File compressi: zip, tar, tar.gz e tar.bz.

La Figura 5.4 mostra le prestazioni del sistema, con lo stesso *setup* precedente, gestendo gli allegati. Come si può notare si ha un calo di circa un 2.5% dovuto probabilmente ad un contenuto troppo dispersivo degli allegati. I test successivi verranno svolti rimuovendo questa funzionalità.

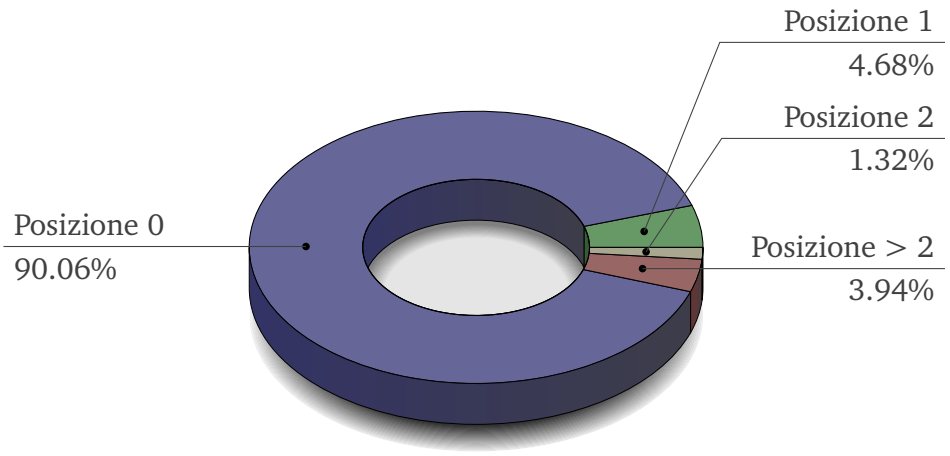


Figura 5.4: Prestazioni con la gestione degli allegati.

5.4.8 Il nuovo algoritmo di *pre-processing*

Dopo aver analizzato le diverse parti del nuovo algoritmo di *pre-processing* possiamo definirne il flusso (Figura 5.5).

Nella parte sinistra della figura è rappresentata la parte che gestisce il messaggio nella sua interezza, moltiplicando i termini in maiuscolo e successivamente i termini dell'oggetto. La parte destra invece mostra il trattamento delle singole *word* all'interno dell'*e-mail*: se la parola contiene delle cifre (es. *win2003*) allora viene aggiunta senza ulteriori controlli altrimenti, se non è una *stop word*, viene utilizzato anche il dizionario della lingua corrispondente (rilevata precedentemente). Se la parola non appartiene al dizionario, viene corretta (e ridotta a *stem*), altrimenti viene passata alla funzione di *stemming* e infine aggiunta alla rappresentazione.

5.5 Indicizzazione

Il passo successivo al *pre-processing*, come descritto nella Sezione 2.2.2, è l'indicizzazione. In questa fase viene trasformato il testo ottenuto dallo *step* precedente in un vettore di *token*, dette *feature*. Le *feature* sono affiancate da dei valori reali che rappresentano il loro peso. In questa sezione verranno discussi e testati i diversi approcci per ottenere un'indicizzazione che garantisca buone *performance* per il nuovo sistema di classificazione.

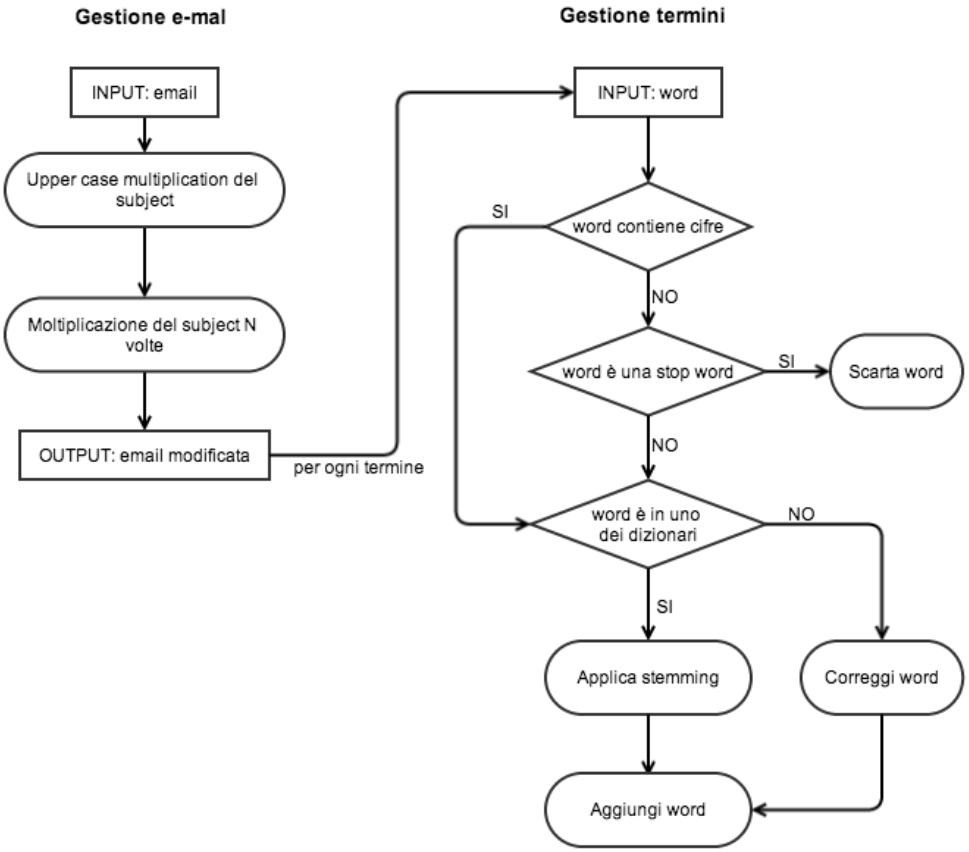


Figura 5.5: Diagramma di flusso della nuova fase di *pre-processing*.

5.5.1 *Feature weighting*

La scelta dei pesi da assegnare alle diverse *feature* può avere una grande influenza sulle prestazioni del classificatore. Nel sistema esistente veniva considerata la semplice frequenza dei termini, e i test svolti fin qui hanno mantenuto la stessa politica. Durante lo sviluppo del nuovo sistema è stato testato l'utilizzo di *Tf-Idf*, è stato quindi necessario implementare la formula descritta nella Sezione 2.2.2. Di seguito vengono riportati i risultati (classificatore *NB*) con questo tipo di peso.
Le prestazioni, con l'utilizzo di *Tf-Idf*, sono più basse rispetto alla *Feature Frequency*, è probabile che l'applicazione di *Tf-Idf* riduca parzialmente l'effetto della moltiplicazione dei pesi dei termini contenuti nell'oggetto. Nei test successivi verrà sempre utilizzata la *Feature Frequency* come peso delle *feature*.

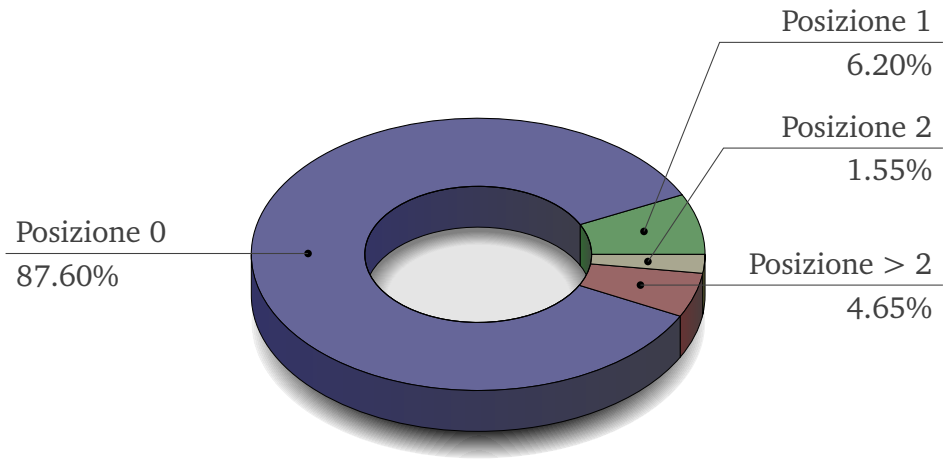


Figura 5.6: Prestazioni con l'utilizzo di *Tf-Idf*.

5.5.2 Feature selection

I test condotti finora sono stati eseguiti, come detto nella Sezione 5.2, utilizzando come tecnica di *Feature Selection* l'*Information Gain*. Vediamo le prestazioni senza l'utilizzo di alcuna tecnica per la selezione delle *feature*.

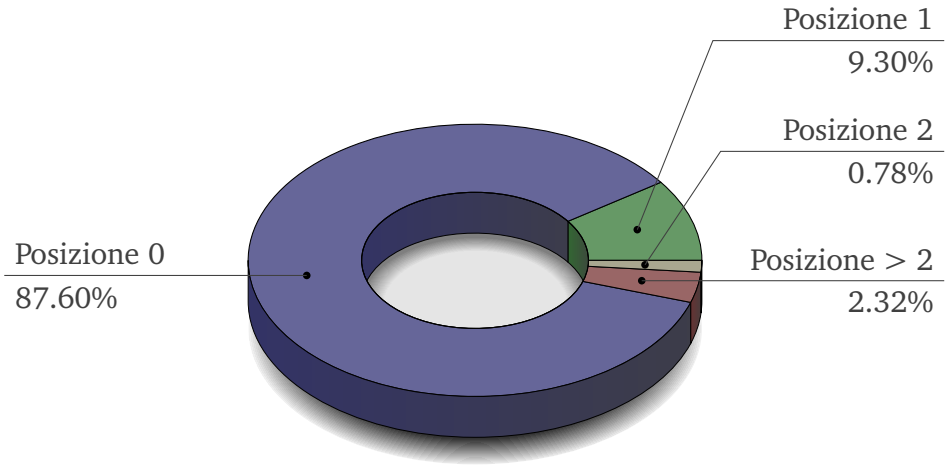


Figura 5.7: Prestazioni senza l'utilizzo della *Feature Selection*.

L'alta dimensionalità dello spazio delle *feature* è uno dei problemi che affliggono la *text categorization* e il calo delle prestazioni del classificatore ne è una prova. Inoltre, un'alta dimensionalità del *feature space* può portare a fenomeni di *overfitting*. Nei test che seguiranno verrà utilizzata come tecnica di *Feature Selection* l'*IG* come è stato per tutti i test condotti finora.

Le scelte adottate finora hanno portato ad un miglioramento delle performance rispetto alla situazione iniziale. Tuttavia queste scelte non necessariamente rappresentano la combinazione migliore possibile, ma una valutazione esaustiva di tutte le combinazione di tecniche di *pre-processing*, *feature weighting* e *feature selection* sarebbe risultato troppo oneroso.

6
Fase di classificazione

Nel capitolo precedente sono state descritte le modifiche apportate alla fase di *pre-processing* e indicizzazione utilizzando come classificatore *Naïve Bayes*. In questo capitolo verranno confrontate le prestazioni tra *NB* e *SVM*, il quale ha bisogno di una fase molto onerosa di *tuning* dei parametri. Successivamente, utilizzando il classificatore con le migliori *performance* si estenderà la classificazione all'intera tassonomia.

6.1 Il classificatore *SVM*

Tutte le prove effettuate finora sono state condotte utilizzando il classificatore probabilistico *NB*. Ora che è stata definita la fase di *pre-processing* è possibile eseguire il *tuning* dei parametri per il classificatore *SVM*. Questa tipologia di classificazione è resa possibile, all'interno del sistema, dalla libreria *LibSVM* che offre una serie di classi per la classificazione e la regressione.
Per una descrizione più dettagliata si rimanda a [27] e al sito della libreria `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`.

Oltre alla classificazione binaria *LibSVM* implementa in modo efficiente anche la classificazione *multilabel*, e questo permette di usare tale strumento per classificare le *e-mail* secondo la tassonomia. Per prima cosa si è dovuto eseguire una serie di test per trovare i valori più adatti per gli *hyperparameters*, ovvero quei parametri che garantiscono prestazioni migliori in base alla funzione *kernel* che si sta utilizzando. L'efficacia dei metodi *SVM* dipende fortemente dalla scelta della funzione *kernel*. Nelle prossime sezioni verranno presentate due delle funzioni *kernel* più utilizzate, mostrando le prestazioni ottenute con diverse combinazioni di parametri.

6.1.1 *Kernel* polinomiale

La funzione *kernel* di tipo polinomiale è definita come segue:

$$K_p(\vec{x}, \vec{x}') = (\gamma * \vec{x} * \vec{x}' + C_0)^d$$

dove d è il grado del polinomio. Come si può notare dalla definizione è necessario trovare la combinazione ottimale dei parametri γ, C_0 e d. Le tabelle seguenti mostrano i risultati utilizzando combinazioni di parametri diverse, includendo anche il parametro C definito nella sezione 2.5.

	C_0							
	0.1		1		10		100	
C	P0 (%)	*ES*	P0 (%)	*ES*	P0 (%)	*ES*	P0 (%)	*ES*
10^{-4}	85.58	0.51	86.2	0.46	88.68	0.41	**90.54**	0.34
10^{-3}	87.6	0.46	88.99	0.38	**90.54**	0.35	89.77	0.36
10^{-2}	87.52	0.48	87.98	0.47	90.47	0.33	90.39	**0.33**
10^{-1}	85.19	0.56	85.89	0.49	88.37	0.42	87.21	0.45
10^{0}	84.81	0.48	84.96	0.56	86.36	0.52	87.29	0.52
10^{1}	84.43	0.48	84.5	0.54	89.92	0.34	86.267	0.51
10^{2}	83.1	0.57	87.43	0.47	86.05	0.48	87.44	0.35
10^{3}	85.58	0.52	83.88	0.63	85.74	0.56	86.36	0.49

Tabella 6.1: Prestazioni con *kernel* polinomiale: $d = 2$, $\gamma = 2^{-5}$. P0 indica la prima posizione nel *ranking*, mentre *ES* è l'*error score*. (*k-Cross Validation* con $k = 10$)

Le tabelle mostrano come si raggiungano risultati migliori con un *kernel* polinomiale con grado basso (2 o 3). Sostanzialmente, esclusi alcuni casi, soprattutto con $C_0 = 0.1$, le prestazioni delle *SVM* superano abbondantemente l'80% toccando un picco, con la configurazione $\gamma = 2^{-9}, d = 3, C = 10^{-4}$ e $C_0 = 100$, di 91.71% con un *error score* minimo pari a 0.31. Quindi le *SVM* con *kernel* polinomiale si avvicinano alle *performance* ottenute con *NB*, anche se il *tuning* dei parametri è stato molto oneroso.

	C_0							
	0.1		1		10		100	
C	P0 (%)	*ES*	P0 (%)	*ES*	P0 (%)	*ES*	P0 (%)	*ES*
10^{-4}	84.19	0.58	85.89	0.5	88.76	0.44	88.45	0.45
10^{-3}	87.44	0.5	88.06	0.43	87.67	0.45	**89.22**	0.39
10^{-2}	82.48	0.66	84.96	0.53	84.57	0.54	86.43	0.48
10^{-1}	81.16	0.68	83.8	0.55	85.58	0.51	88.19	0.41
10^{0}	83.88	0.52	82.79	0.56	84.96	0.52	88.99	0.38
10^{1}	81.55	0.61	81.24	0.64	83.72	0.56	88.84	0.38
10^{2}	78.76	0.8	83.1	0.61	84.96	0.51	87.44	0.44
10^{3}	81.55	0.68	82.17	0.57	84.5	0.61	89.15	**0.35**

Tabella 6.2: Prestazioni con *kernel* polinomiale: $d = 3$, $\gamma = 2^{-5}$. P0 indica la prima posizione nel *ranking*, mentre *ES* è l'*error score*. (*k-Cross Validation* con $k = 10$)

	C_0							
	0.1		1		10		100	
C	P0 (%)	*ES*	P0 (%)	*ES*	P0 (%)	*ES*	P0 (%)	*ES*
10^{-4}	82.87	0.59	81.63	0.68	86.9	0.47	89.07	0.39
10^{-3}	77.83	0.77	80.08	0.7	82.02	0.62	87.52	0.48
10^{-2}	77.75	0.8	79.46	0.73	82.95	0.59	87.36	0.49
10^{-1}	75.97	0.76	19.15	0.74	82.71	0.6	89.61	0.37
10^{0}	75.97	0.75	78.29	0.79	84.34	0.56	88.84	0.43
10^{1}	76.9	0.77	77.98	0.76	87.67	0.42	83.41	0.6
10^{2}	77.52	0.71	78.76	0.73	82.17	0.68	87.29	0.46
10^{3}	76.9	0.77	79.2	0.68	82.48	0.57	**90.39**	**0.33**

Tabella 6.3: Prestazioni con *kernel* polinomiale: $d = 4$, $\gamma = 2^{-5}$. P0 indica la prima posizione nel *ranking*, mentre *ES* è l'*error score*. (*k-Cross Validation* con $k = 10$)

	C_0							
	0.1		1		10		100	
C	P0 (%)	*ES*	P0 (%)	*ES*	P0 (%)	*ES*	P0 (%)	*ES*
10^{-4}	83.41	0.52	83.88	0.52	87.83	0.4	90.16	0.38
10^{-3}	84.42	0.54	87.13	0.44	89.53	0.42	89.84	0.38
10^{-2}	88.37	0.42	87.29	0.45	**91.09**	**0.32**	89.15	0.42
10^{-1}	88.6	0.43	89.3	0.48	90.23	0.35	88.99	0.41
10^{0}	86.59	0.48	89.07	0.38	89.77	0.34	88.79	0.4
10^{1}	84.73	0.54	86.59	0.5	87.91	0.43	90.23	0.32
10^{2}	85.5	0.51	86.43	0.46	87.52	0.43	89.46	0.33
10^{3}	84.88	0.52	83.57	0.59	88.29	0.39	89.77	0.34

Tabella 6.4: Prestazioni con *kernel* polinomiale: $d = 2$, $\gamma = 2^{-7}$. P0 indica la prima posizione nel *ranking*, mentre *ES* è l'*error score*. (*k-Cross Validation* con $k = 10$)

	C_0							
	0.1		1		10		100	
C	P0 (%)	*ES*	P0 (%)	*ES*	P0 (%)	*ES*	P0 (%)	*ES*
10^{-4}	80.78	0.63	85.19	0.54	**90.16**	0.4	88.99	0.4
10^{-3}	84.65	0.56	84.42	0.55	89.38	**0.38**	89.22	**0.38**
10^{-2}	84.81	0.6	86.91	0.48	89.38	0.4	88.76	0.39
10^{-1}	85.35	0.58	87.13	0.48	87.36	0.45	89.07	0.4
10^{0}	82.87	0.64	85.04	0.52	85.89	0.53	83.57	0.54
10^{1}	82.79	0.58	87.67	0.46	87.29	0.44	87.05	0.45
10^{2}	82.64	0.62	83.02	0.55	85.97	0.54	88.29	0.39
10^{3}	81.09	0.65	84.34	0.54	88.29	0.43	88.06	0.41

Tabella 6.5: Prestazioni con *kernel* polinomiale: $d = 3$, $\gamma = 2^{-7}$. P0 indica la prima posizione nel *ranking*, mentre *ES* è l'*error score*. (*k-Cross Validation* con $k = 10$)

	C_0							
	0.1		1		10		100	
C	P0 (%)	*ES*	P0 (%)	*ES*	P0 (%)	*ES*	P0 (%)	*ES*
10^{-4}	81.71	0.67	81.71	0.61	**90.31**	0.37	87.52	0.44
10^{-3}	80.39	0.69	83.57	0.57	89.46	**0.36**	87.83	0.42
10^{-2}	83.57	0.57	85.43	0.55	86.55	0.51	88.76	0.39
10^{-1}	80.31	0.7	83.18	0.62	88.29	0.4	89.07	0.41
10^{0}	78.53	0.54	80.47	0.51	87.44	0.43	88.89	0.34
10^{1}	75.27	0.84	81.32	0.63	87.67	0.42	88.84	0.37
10^{2}	77.44	0.78	81.55	0.7	86.43	0.51	87.67	0.43
10^{3}	77.44	0.74	82.64	0.61	87.05	0.46	88.76	0.4

Tabella 6.6: Prestazioni con *kernel* polinomiale: $d = 4$, $\gamma = 2^{-7}$. P0 indica la prima posizione nel *ranking*, mentre *ES* è l'*error score*. (*k-Cross Validation* con $k = 10$)

	C_0							
	0.1		1		10		100	
C	P0 (%)	*ES*	P0 (%)	*ES*	P0 (%)	*ES*	P0 (%)	*ES*
10^{-4}	80.31	0.6	85.74	0.41	86.2	0.44	87.36	0.44
10^{-3}	83.1	0.53	85.43	0.46	87.91	0.38	**91.78**	0.33
10^{-2}	86.05	0.46	87.98	0.41	90.93	0.34	91.09	0.31
10^{-1}	87.98	0.47	90.54	0.36	91.09	**0.31**	88.99	0.39
10^{0}	88.14	0.43	90.78	0.34	89.22	0.39	88.29	0.42
10^{1}	87.67	0.49	88.68	0.42	87.29	0.48	87.13	0.44
10^{2}	87.05	0.44	87.06	0.46	87.29	0.46	89.22	0.38
10^{3}	85.89	0.47	86.74	0.5	88.29	0.44	89.77	0.34

Tabella 6.7: Prestazioni con *kernel* polinomiale: $d = 2$, $\gamma = 2^{-9}$. P0 indica la prima posizione nel *ranking*, mentre *ES* è l'*error score*. (*k-Cross Validation* con $k = 10$)

	C_0							
	0.1		1		10		100	
C	P0 (%)	*ES*	P0 (%)	*ES*	P0 (%)	*ES*	P0 (%)	*ES*
10^{-4}	82.95	0.59	84.11	0.55	89.84	0.36	**91.71**	**0.33**
10^{-3}	73.67	0.78	84.76	0.53	90.93	0.35	88.29	0.43
10^{-2}	84.03	0.6	87.91	0.41	90.16	0.36	88.99	0.42
10^{-1}	85.19	0.51	89.38	0.38	90.16	0.36	88.29	0.45
10^{0}	87.67	0.49	88.76	0.37	87.52	0.46	87.44	0.46
10^{1}	86.51	0.53	87.21	0.44	88.29	0.42	87.75	0.41
10^{2}	82.71	0.56	86.74	0.47	88.6	0.4	88.37	0.4
10^{3}	82.48	0.57	85.81	0.49	86.59	0.49	89.46	0.39

Tabella 6.8: Prestazioni con *kernel* polinomiale: $d = 3$, $\gamma = 2^{-9}$. P0 indica la prima posizione nel *ranking*, mentre *ES* è l'*error score*. (*k-Cross Validation* con $k = 10$)

	C_0							
	0.1		1		10		100	
C	P0 (%)	*ES*	P0 (%)	*ES*	P0 (%)	*ES*	P0 (%)	*ES*
10^{-4}	78.68	0.83	83.1	0.56	90.35	0.36	89.46	0.37
10^{-3}	78.84	0.72	86.59	0.5	**90.47**	0.37	88.53	0.41
10^{-2}	80	0.69	86.9	0.46	88.91	0.41	88.84	0.38
10^{-1}	83.49	0.58	87.52	0.45	89.46	0.4	87.67	0.44
10^{0}	85.12	0.53	87.13	0.51	88.76	0.42	90.08	**0.33**
10^{1}	80.23	0.72	85.43	0.53	88.99	0.45	87.21	0.47
10^{2}	80	0.7	85.27	0.53	87.29	0.43	90	0.36
10^{3}	81.24	0.62	84.73	0.56	88.06	0.45	89.22	0.39

Tabella 6.9: Prestazioni con *kernel* polinomiale: $d = 4$, $\gamma = 2^{-9}$. P0 indica la prima posizione nel *ranking*, mentre *ES* è l'*error score*. (*k-Cross Validation* con $k = 10$)

	C_0							
	0.1		1		10		100	
C	P0 (%)	*ES*	P0 (%)	*ES*	P0 (%)	*ES*	P0 (%)	*ES*
10^{-4}	57.6	1.06	85.27	0.47	86.67	0.42	88.14	0.39
10^{-3}	84.34	0.47	87.83	0.41	86.12	0.44	89.77	**0.3**
10^{-2}	84.73	0.51	86.2	0.41	88.99	0.39	90.54	0.38
10^{-1}	85.66	0.48	90.47	0.34	**90.93**	0.36	90.85	0.34
10^{0}	87.65	0.46	89.92	0.42	88.53	0.4	86.67	0.46
10^{1}	86.06	0.47	86.98	0.48	86.59	0.46	87.05	0.44
10^{2}	84.87	0.46	86.68	0.47	87.91	0.45	85.81	0.52
10^{3}	86.22	0.46	86.51	0.46	86.74	0.45	86.36	0.49

Tabella 6.10: Prestazioni con *kernel* polinomiale: $d = 2$, $\gamma = 2^{-11}$. P0 indica la prima posizione nel *ranking*, mentre *ES* è l'*error score*. (*k-Cross Validation* con $k = 10$)

	C_0							
	0.1		1		10		100	
C	P0 (%)	*ES*	P0 (%)	*ES*	P0 (%)	*ES*	P0 (%)	*ES*
10^{-4}	41.78	1.53	84.34	0.51	87.6	0.43	87.98	0.41
10^{-3}	73.8	0.74	82.17	0.52	87.98	0.45	90.31	**0.31**
10^{-2}	81.71	0.61	85.27	0.47	90.16	0.41	90	0.36
10^{-1}	84.42	0.54	**90.7**	0.37	89.77	0.38	88.6	0.42
10^{0}	87.89	0.45	89.61	0.42	88.76	0.41	86.43	0.48
10^{1}	84.4	0.53	86.98	0.48	85.81	0.49	88.84	0.39
10^{2}	86.12	0.47	86.74	0.46	86.90	0.44	86.74	0.47
10^{3}	86.43	0.46	86.43	0.45	86.28	0.53	85.74	0.51

Tabella 6.11: Prestazioni con *kernel* polinomiale: $d = 3$, $\gamma = 2^{-11}$. P0 indica la prima posizione nel *ranking*, mentre *ES* è l'*error score*. (*k-Cross Validation* con $k = 10$)

	C_0							
	0.1		1		10		100	
C	P0 (%)	*ES*	P0 (%)	*ES*	P0 (%)	*ES*	P0 (%)	*ES*
10^{-4}	41.16	1.62	84.03	0.48	**91.09**	**0.33**	87.98	0.41
10^{-3}	56.43	1.31	84.42	0.49	89.84	0.36	88.06	0.41
10^{-2}	80.23	0.73	85.97	0.48	89.07	0.38	89.84	0.37
10^{-1}	81.4	0.64	89.22	0.42	88.29	0.4	87.6	0.43
10^{0}	83.34	0.56	87.98	0.43	87.21	0.46	87.05	0.45
10^{1}	84.03	0.53	87.6	0.46	87.29	0.45	86.51	0.48
10^{2}	85.58	0.5	86.61	0.45	86.2	0.46	85.97	0.53
10^{3}	86.32	0.49	85.19	0.5	88.45	0.41	85.97	0.46

Tabella 6.12: Prestazioni con *kernel* polinomiale: $d = 4$, $\gamma = 2^{-11}$. P0 indica la prima posizione nel *ranking*, mentre *ES* è l'*error score*. (*k-Cross Validation* con $k = 10$)

6.1.2 *Kernel* RBF

La funzione *kernel* di tipo RBF (*Radial Basis Function*) è definita come segue:

$$K_{RBF}(\vec{x}, \vec{x}') = exp(-\gamma||\vec{x} - \vec{x}'||^2) \quad \gamma > 0.$$

Anche questa funziona, come la precedente, contiene dei parametri ed è stato quindi necessario eseguire una serie di test per valutare la combinazione migliore. I parametri in questo caso sono: γ e C. Di seguito sono riportati i risultati dei test.

Come si può notare dalle Tabelle 6.13 e 6.14, valori vicini a 1 di γ causano un degrado notevole delle prestazioni, le quali raggiungono il loro massimo con $\gamma = 2^{-13}$ e $C = 10$. Complessivamente le *performance* dei due *kernel* si equivalgono anche se con il *kernel* polinomiale si raggiungono valori minori di *error score*.

	γ					
	2^{-3}		2^{-5}		2^{-7}	
C	P0 (%)	*ES*	P0 (%)	*ES*	P0 (%)	*ES*
10^-4	26.2	2.11	32.4	1.88	31.78	3.07
10^-3	26.98	2.06	30.54	1.86	45.43	1.45
10^-2	30.85	2.04	36.74	1.86	53.49	1.2
10^-1	30.08	2.03	39.53	1.75	60.47	1.23
10^0	30.85	2.04	39.84	1.79	67.6	0.98
10^1	31.78	2.03	40.16	1.82	67.29	0.99
10^2	32.4	2.03	41.86	1.76	67.75	1.08
10^3	32.09	2	40.31	1.83	70.54	0.76

Tabella 6.13: Prestazioni con *kernel* RBF ($2^{-3} \leq \gamma \leq 2^{-7}$). P0 indica la prima posizione nel *ranking*, mentre *ES* è l'*error score*. (*k-Cross Validation* con $k = 10$)

	γ							
	2^{-9}		2^{-11}		2^{-13}		2^{-15}	
C	P0 (%)	*ES*	P0 (%)	*ES*	P0 (%)	*ES*	P0 (%)	*ES*
10^-4	59.69	1.44	77.83	0.6	74.73	0.68	70.85	0.74
10^-3	69.61	0.78	80.93	0.55	81.09	0.54	77.21	0.63
10^-2	80.62	0.62	84.65	0.47	84.65	0.49	84.19	0.52
10^-1	78.14	0.61	84.5	0.51	86.67	0.44	86.2	0.41
10^0	82.95	0.63	91.16	0.33	91.32	0.33	88.84	0.4
10^1	83.26	0.53	89.3	0.37	**90.39**	**0.35**	90.08	0.42
10^2	80.78	0.64	86.2	0.54	88.68	0.42	89.77	0.39
10^3	84.96	0.51	89.46	0.36	89.77	0.38	88.37	0.38

Tabella 6.14: Prestazioni con *kernel* RBF ($2^{-9} \leq \gamma \leq 2^{-15}$). P0 indica la prima posizione nel *ranking*, mentre *ES* è l'*error score*. (*k-Cross Validation* con $k = 10$)

6.2 Classificazione sulla tassonomia

Finora abbiamo mostrato le prestazioni del classificatore sul primo livello della tassonomia, e nel suo lavoro Astegno [27] aveva esteso la classificazione anche al secondo livello. Su richiesta dell'azienda (requisito FO_10) la classificazione deve coinvolgere l'intera tassonomia. Prima di descrivere l'approccio con cui si è affrontato questo problema vediamo lo stato dell'arte nell'ambito della classificazione gerarchica.

6.2.1 Stato dell'arte

Nell'affrontare il problema della classificazione gerarchica si possono utilizzare principalmente due approcci [4]:

- Approccio *Big-Bang*: un documento può essere classificato in qualsiasi categoria della gerarchia utilizzando un solo *step* di classificazione;
- Approccio *Top-Down Level-Based*: la classificazione di un documento è ottenuta grazie alla collaborazione di più classificatori costruiti ad ogni livello della gerarchia. In un livello posso esserci più classificatori, uno per ogni nodo (interno o foglia). Quindi il documento viene prima classificato al livello radice, poi viene passato al livello successivo. Questo processo viene iterato finché non viene raggiunta una foglia.

Approccio *Big-Bang*

Nel loro lavoro Labrou and Finin (1999) [43], hanno sviluppato un classificatore *Rocchio-Like* che classifica documenti in un Grafo diretto aciclico (*DAG*). Per ogni categoria viene creato un vettore di pesi, detto profilo (*category profile*), utilizzando il nome della categoria, i titoli e una breve descrizione dei documenti che appartengono ad essa. Per calcolare la similarità tra un nuovo documento e la classe viene calcolata la *cosine similarity* tra il vettore profilo e il vettore caratteristico del documento. Sasaki et al. (1998) [28] hanno utilizzato un approccio diverso con l'algoritmo *RIPPER* [8] che sulla base dei documenti di *training* crea un insieme di regole le quali indicano, per una categoria, le parole che il documento deve contenere per essere assegnato a tale classe. Un metodo simile è stato utilizzato anche da Wang et al. (2001), il quale genera delle regole per classificare i documenti in una tassonomia. La differenza tra questi due lavori è che nel primo un documento è assegnato ad una e una sola classe nel secondo un documento può appartenere ad un insieme di classi. Lo stesso anno Toutanova et al. [25] proposero un'estensione del classificatore *Naïve Bayes* in grado di classificare un documento all'interno di una gerarchia di classi. La gerarchia degli argomenti è utilizzata per stimare la probabilità condizionata dei termini in modo da ottenere una differenziazione tra le parole nella gerarchia sulla base al loro livello di specificità.

Approccio *Top-Down Level-Based*

Nel lavoro di Koller e Sahami [9] utilizzano un classificatore gerarchico basato su *Naïve Bayes* dove ad ogni nodo interno corrisponde un classificatore *NB*. Mostrarono che questo metodo ha prestazioni migliori (su un *dataset* estratto

dalla collezione *Reuters*) rispetto alla classificazione *flat*, sempre con *NB*, se viene estratto un numero limitato di *feature*. Nel caso il numero di *feature* selezionate è elevato i due metodi hanno prestazioni comparabili. In [36], D'Alessio et al. descrivono le categorie tramite delle *feature* estratte dal *training set* usando un algoritmo chiamato *ACTION* (*Automatic Classification for Full-Text Documents*). Ad ogni categoria, un classificatore binario (o m-ario) è utilizzato per determinare se un documento appartiene alla categoria o a una sua discendente. Dumais e Chen [39] proposero un classificatore basato su *SVM* per classificare pagine *web* in una struttura gerarchica virtuale dove solo le foglie possono avere documenti associati. Nei loro esperimenti hanno confrontato le prestazioni di questo classificatore con quello di un classificatore *SVM* non gerarchico, e hanno mostrato come il primo ottenga un *accuracy* più elevata. Un ovvio problema di questo approccio è che se un documento viene classificato in modo errato in un nodo padre (antenato) vanifica le classificazioni che avvengono nei nodi figli, poiché l'errore avviene a monte.

6.2.2 Categorie sulle foglie

L'approccio adottato in questa tesi è il *Top-Down Level-Based*. L'intuizione è che, avendo una tassonomia abbastanza ampia, una classificazione di tipo *Big-Bang* sia computazionalmente più onerosa poiché deve considerare tutte le classi della gerarchia.

Questo tipo di approccio, come detto nella Sezione 6.2.1, classifica i documenti sulle foglie della tassonomia, però è possibile che un documento appartenga ad una classe che non sta in una foglia poiché il suo contenuto è generale rispetto a tale categoria e quindi non avrebbe senso specializzare la sua classificazione. In questi casi c'è bisogno o di un meccanismo di arresto, che è in grado di fermare la discesa lungo l'albero della gerarchia, oppure si dovrebbe in qualche modo poter considerare anche i nodi interni come possibili foglie. Il caso in esame soffre di questo problema, infatti alcuni messaggi di posta appartengono a classi poste in nodi interni poiché sono generiche di quel sottoalbero.

La Figura 6.1 mostra un esempio di tassonomia, in cui nodi rettangolari rappresentano classi che contengono esempi associati e quelli circolari non ne contengono. Come si può vedere il nodo *a* contiene esempi ed è un nodo interno:
Per ovviare questo problema è stato usato un approccio che considera un nodo interno, con esempi associati ad esso, figlio di se stesso. In Figura 6.2 si può vedere come viene risolto il problema. Il nodo *a* viene replicato (*a*') come figlio

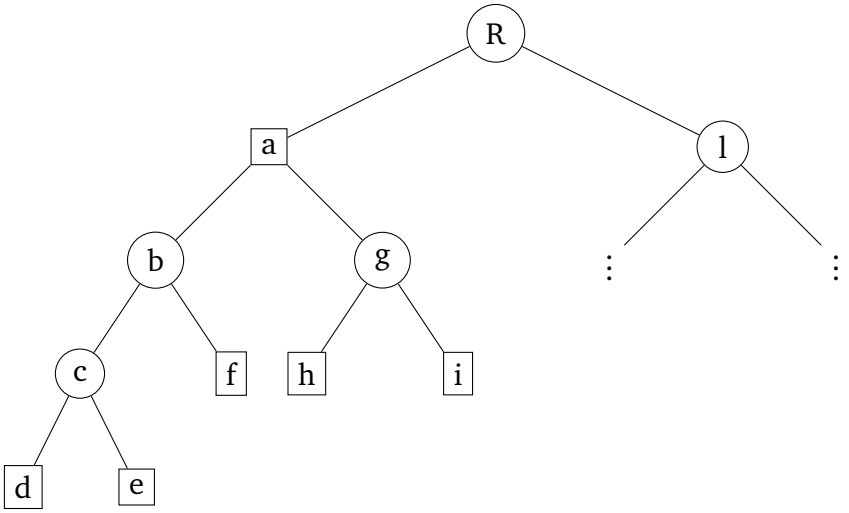


Figura 6.1: Esempio di struttura della tassonomia con nodi interni con esempi associati: la classe *a*, raffigurata come un rettangolo, può avere documenti associati.

di se stesso e gli esempi ad esso associati vengono trasmessi al figlio appena creato. In questo modo il classificatore posto nel nodo padre dovrà classificare in base agli esempi che stanno nel suo sottoalbero, presi quindi tra le classi *a'* (= a), *b* e *g*. Questo metodo rappresenta una trasformazione della tassonomia che non ne altera la sua semantica e che rende possibile l'ideazione di un algoritmo di *training* che agisce in modo ricorsivo.

6.2.3 Nuovo algoritmo di *training*

Per riuscire a classificare seguendo l'intera struttura della tassonomia è necessario creare un classificatore per ogni nodo interno. Per fare questo si è pensato di scorrere l'albero in profondità (*depth first*) e per ogni nodo interno si crea il classificatore associato. Di seguito viene descritto l'algoritmo più in dettaglio: *Tr* rappresenta il *training set* pre-elaborato e indicizzato.
Training(Tr)

1. Creo il classificatore di livello 1 sull'intero *training set* (*Tr*) e con le classi di primo livello;
2. Salvo il classificatore in una cartella prefissata;
3. Per ogni classe c_i figlia della radice della tassonomia
    1. Chiamo **DepthFirstTraining**(c_i, *Tr*);

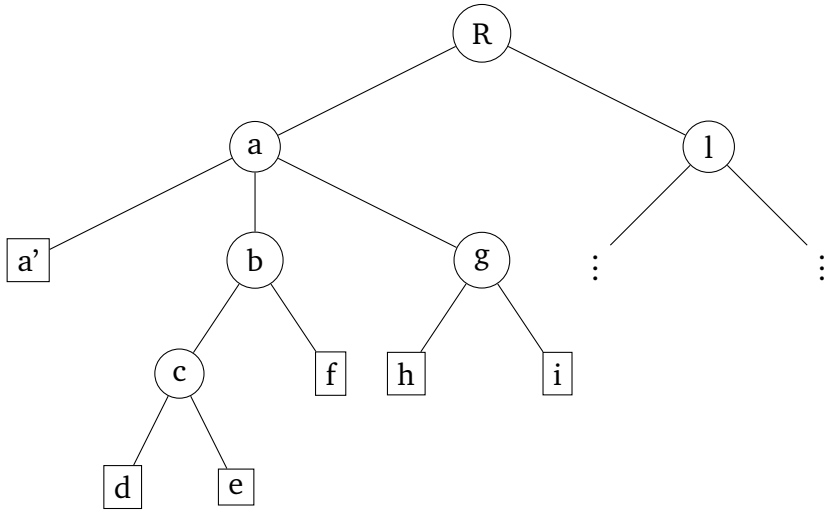


Figura 6.2: Esempio di struttura della tassonomia con nodi interni vuoti.

DepthFirstTraining(c, Tr)

1. Se c ha figli
    1. $Tr_c \leftarrow$ esempi di Tr che appartengono alla classe c o a classi figlie;
    2. $C \leftarrow$ insieme delle classi figlie di c;
    3. Se $\exists d \in Tr_c \mid \mathscr{F}(d) = c$
        1. $C \leftarrow C \cup c$;
    4. Creo il classificatore sul *dataset* Tr_c e con le classi in C;
    5. Salvo il classificatore in una cartella prefissata;
    6. Per ogni classe c_i figlia di c
        1. Chiamo **DepthFirstTraining**(c_i, Tr_c);

6.2.4 Nuovo algoritmo di classificazione

Per riuscire ad utilizzare tutti i classificatori creati dal nuovo metodo di *training* c'è bisogno di definire anche un nuovo algoritmo che effettua la classificazione di nuovi messaggi, sfruttando i classificatori creati. Di seguito viene presentato l'algoritmo: m è l'*e-mail* da classificare.

Classify(m)

1. $\widetilde{\mathscr{F}} \leftarrow$ classificatore di livello 1 generato dall'algoritmo di *training*;
2. $\vec{f} \leftarrow$ il messaggio m indicizzato;
3. Ritorna DeepClassify($\vec{f}$, $root$);

DeepClassify($\vec{f}$, c)

1. Se la classe c ha figli
    1. $\widetilde{\mathscr{F}} \leftarrow$ classificatore corrispondente alla classe c;
    2. Calcolo il *ranking* (R_0) del classificatore $\widetilde{\mathscr{F}}(\vec{f})$;
    3. Per ogni classe $c_i, i \in 1, 2, ..., n$ figlia di c
        1. $R_i \leftarrow$ DeepClassify(m, c_i);
    4. $\vec{R} \leftarrow [R_0, R_1, ..., R_n]$
    5. $R \leftarrow$ CalculateFullRanking($\vec{R}$, c)
    6. Ritorna R;

CalculateFullRanking($\vec{R}$, c)

1. Per ogni classe c_i figlia di c
    1. Per ogni valore p_j nel *ranking* R_i calcolo $p_j = p_0 * p_j$
2. Ritorno $\vec{R} \setminus R_0$;

CalculateFullRanking calcola la probabilità composta di una classe sulla base della probabilità della classe padre (anche nel caso la classe padre sia la classe stessa, vedi Sezione 6.2.2). Questo calcolo della probabilità composta è reso possibile anche con l'ausilio delle *SVM* poiché la libreria *LibSVM* offre la possibilità di ritornare la stima di probabilità delle classi, quindi l'*output* è dello stesso tipo di *NB*.

I test sull'intera tassonomia saranno eseguiti con *NB* poiché ha ottenuto buone prestazioni al primo livello e soprattutto non richiede un *tuning* dei parametri. Infatti, adottare *SVM* significherebbe rieseguire la calibrazione dei parametri per tutti i sotto classificatori.

6.2.5 Classificazione con *Naïve Bayes*

Applicando l'algoritmo descritto nella sezione 6.2.3, con *NB*, si ottengono le seguenti prestazioni:

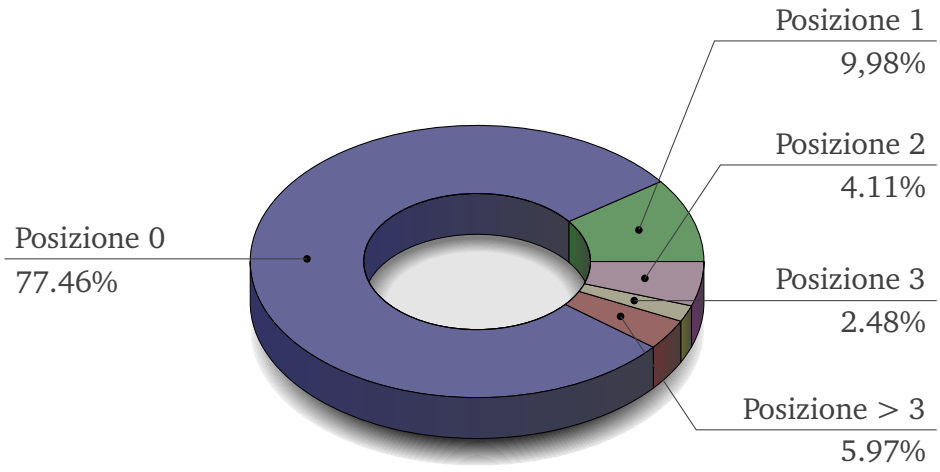


Figura 6.3: Prestazioni sull'intera tassonomia con *Naïve Bayes*.

Passando dal primo livello all'intera tassonomia si osserva un calo delle prestazioni di circa il 15%. Questo significa che avvengono diverse *misclassification* all'interno dei sottoalberi. Per capire meglio la situazione è necessario analizzare la matrice di confusione, dei diversi sottoalberi. In particolare si è notato che la classificazione del sottoalbero della classe 6 risulta difficile. Di seguito sono elencate le matrici di confusione (tabelle 6.15, 6.16 e 6.17) del sottoalbero 6 (in rosso sono evidenziati i casi di maggiore confusione).

	6.1	**6.2**	**6.3**	**6**
6.1	6.45	0	1,25	0
6.2	0	53.62	0	3.59
6.3	0	1.2	29.41	0
6	0	2,08	0	2,39

Tabella 6.15: Matrice di confusione del sottoalbero 6.

6.2.6 Il problema del sottoalbero 6.3.2

Dalla Tabella 6.17 appare evidente che c'è molta difficoltà nel distinguere le diverse classi del sottoalbero 6.3.2. In effetti, dopo un'attenta analisi del contenuto dei messaggi di queste categorie, si è notato che la maggior parte dei

	6.3.1	**6.3.2**
6.3.1	2.82	8.37
6.3.2	0.73	79.16

Tabella 6.16: Matrice di confusione del sottoalbero 6.3.

	6.3.2.1	**6.3.2.2**	**6.3.2**
6.3.2.1	19.63	1.85	8.08
6.3.2.2	6.24	23.21	14.32
6.3.2	9.82	5.31	11.55

Tabella 6.17: Matrice di confusione del sottoalbero 6.3.2.

termini è comune. Gli unici che, anche per un addetto esperto, riescono a discriminare queste categorie sono dei valori numerici che rappresentano le diverse versioni di uno stesso *software*. Per provare a risolvere il problema si sono effettuate alcune modifiche alla fase di *pre-processing*.

Ripetizione dei valori numerici

Un approccio più semplice e analogo a quello avuto per la gestione dell'oggetto (vedi Sezione 5.4.6) è di ripetere più volte i valori numerici all'interno del messaggio. La Figura 6.4 mostra le prestazioni del sistema ripetendo 5 volte i valori all'interno dei messaggi. Mentre la Tabella 6.20 mostra la matrice di confusione utilizzando questo approccio.

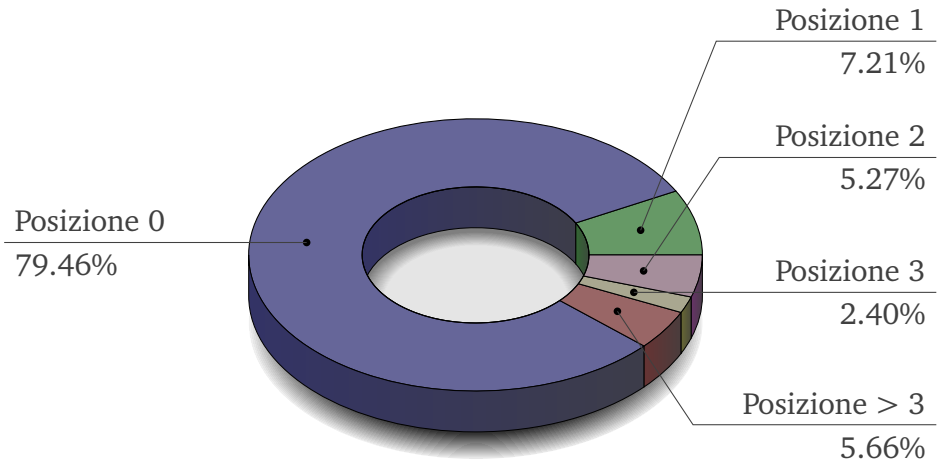


Figura 6.4: Prestazioni con la ripetizione dei valori numerici (5x).

	6.1	**6.2**	**6.3**	**6**
6.1	7.4	0	3.67	0
6.2	0	52	0	3.72
6.3	0	0	25.7	0
6	0	3.72	0	3.72

Tabella 6.18: Matrice di confusione del sottoalbero 6 dopo la modifica.

	6.3.1	**6.3.2**
6.3.1	6.68	0
6.3.2	0	93.32

Tabella 6.19: Matrice di confusione del sottoalbero 6.3 dopo la modifica.

Le tabelle 6.18, 6.19 e 6.20 evidenziano che la situazione è cambiata: nei sottoalberi 6 e 6.3 la confusione complessiva è diminuita. Lo stesso vale per 6.3.2 anche se solo parzialmente. La classe 6.3.2 rimane ancora molto incerta, con la differenza che ora viene classificata in modo errato solo con la classe 6.3.2.2 (vedi Tabella 6.20). Questo problema è principalmente dovuto al fatto che il nodo 6.3.2 contiene messaggi generici che riguardano simultaneamente tutte le sue sotto classi. La diminuzione degli errori su questo sottoalbero ha portato ad un aumento delle prestazioni: ora in posizione 0 viene raggiunto quasi l'80%.

Durante l'analisi dei messaggi del sottoalbero 6, oltre alle considerazioni fatte nelle sezioni precedenti, si è notato che esistevano diverse forme di uno stesso termine (o serie di termini) importanti ai fini della classificazione. Sostanzialmente queste diverse forme erano storpiature di termini più lunghi, dovuto probabilmente all'abitudine dei tecnici che scrivevano le *e-mail* o a particolari gerghi del settore. L'ideale in queste situazione sarebbe raggruppare questi termini che rappresentano uno stesso oggetto, in un termine unico riducendo il numero di *feature* e creandone di nuove con una più alta frequenza rispetto alle singole. A questo scopo si è pensato di introdurre una tecnica

	6.3.2.1	**6.3.2.2**	**6.3.2**
6.3.2.1	18.52	0	7.17
6.3.2.2	7.17	28.26	7.14
6.3.2	0	14.3	21.43

Tabella 6.20: Matrice di confusione del sottoalbero 6.3.2 dopo la modifica.

chiamata *Term clustering*.

6.3 *Term clustering*

In *Information Retrieval* il *Term Clustering* rappresenta un metodo per raggruppare termini simili tra loro secondo una certa metrica. L'idea alla base è quella di sostituire un insieme di termini che occorrono nel *corpus* con uno o più termini che li rappresenti. Questo raggruppamento di termini omogenei è detto *clustering*. In un certo senso quello che si effettua è un "or" ($\vee$) di più parole, ovvero:

$$w = w_1 \vee w_2 \vee ... \vee w_n$$

in sintesi, ogni occorrenza di w_i, $i \in 1, 2, ..., n$ viene sostituita con il termine (o sequenza di termini) w. Questa tecnica, oltre ad ottenere una riduzione dello spazio delle *feature*, aggiunge generalmente nuove *feature* che possono portare ad un miglioramento delle prestazioni del classificatore. L'idea è che queste nuove *feature* introdotte dal *clustering* potenzialmente hanno un contenuto informativo importante poiché raggruppano più termini singoli.

6.3.1 *Term clustering* manuale

Utilizzando il *term clustering* descritto nella Sezione 6.3 è possibile raggruppare questo insieme di termini e sostituirli con uno comune. Inizialmente questo processo è stato effettuato manualmente, ottenendo le prestazioni in Figura 6.5. La Tabella 6.26 mostra il numero di *cluster* individuati.

Cluster	**# Termini**	**Classe**
1	1	2
2	3	6.3.1
3	3	6.3
4	9	6.3.2.1
5	9	6.3.3.2
6	2	9
7	3	1.1
8	2	11
9	3	1.3

Tabella 6.21: *Cluster* individuati manualmente.

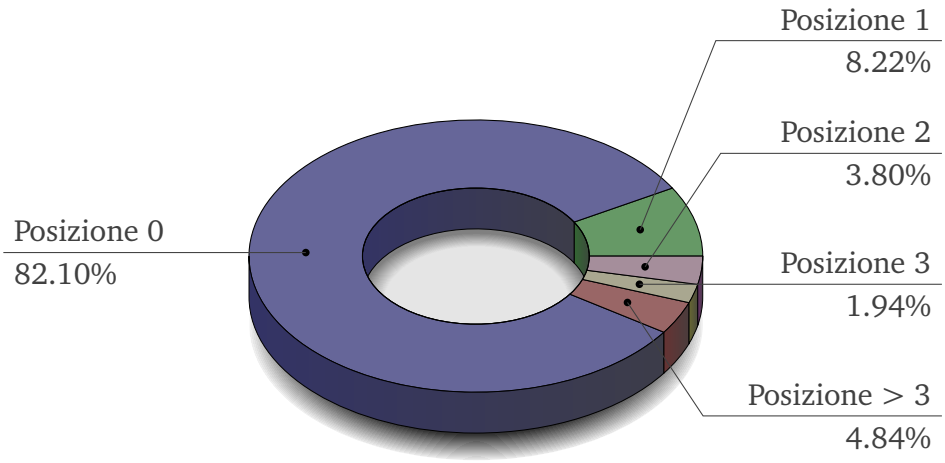


Figura 6.5: Prestazioni con l'adozione del *term clustering* manuale.

	6.1	**6.2**	**6.3**	**6**
6.1	7.46	0.42	1.20	0
6.2	0	50.13	0	2.81
6.3	0	1.61	30.14	0
6	0	2.4	0	2.8

Tabella 6.22: Matrice di confusione del sottoalbero 6 con *term clustering* manuale.

	6.3.1	**6.3.2**
6.3.1	2.26	5.32
6.3.2	1.57	85.42

Tabella 6.23: Matrice di confusione del sottoalbero 6.3 con *term clustering* manuale.

	6.3.2.1	**6.3.2.2**	**6.3.2**
6.3.2.1	28.48	2.65	0.93
6.3.2.2	14.3	25.49	0.93
6.3.2	4.95	6.68	15.57

Tabella 6.24: Matrice di confusione del sottoalbero 6.3.2 con *term clustering* manuale.

Le matrici di confusione 6.22, 6.23 e 6.24 mostrano che la situazione rispetto alla precedente non è cambiata di molto:

- 6 : la situazione è rimasta pressoché invariata;
- 6.3 : la confusione in questo sottoalbero è aumentata soprattutto nella classificazione della classe 6.3.1;
- 6.3.2 : le prestazioni sono migliorate leggermente e tendono ad equilibrare il calo avuto nell'albero padre.

Da questa analisi si può dedurre che le *performance* sono migliorate nel resto della tassonomia. In Figura 6.6 sono presentate le prestazioni al primo livello con il *term clustering* attivo.

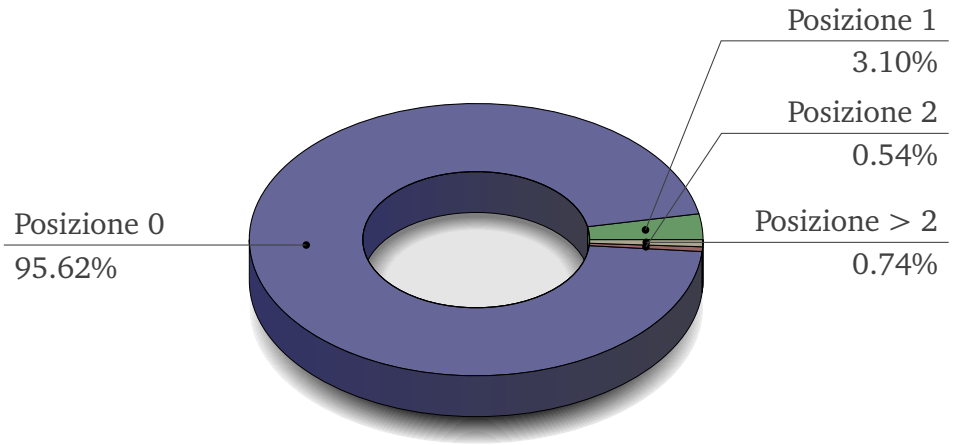


Figura 6.6: Prestazioni al primo livello con l'adozione del *term clustering* manuale.

6.3.2 *Term clustering* automatico

L'approccio manuale porta ad un beneficio ma richiede che la fase di ricerca e "clusterizzazione" venga ripetuta nel tempo da un umano. La situazione ideale sarebbe automatizzare la procedura in modo che non ci sia in futuro bisogno dell'intervento esterno. Per questo motivo è stato ideato un algoritmo che estrae in modo automatico i *cluster*: generalmente il *term clustering* cerca di raggruppare termini simili, secondo una certa metrica, per poi sostituirli con un nuovo termine che li rappresenta. In questa versione invece vengono raggruppati i termini che più si avvicinano ad un certo *set* di termini prefissato, nello specifico i nomi delle classi. Di seguito viene presentato lo pseudo codice

dell'algoritmo.

AutomaticTermClustering(m)

1. Per ogni $w_i \in \{w_0, w_1, ..., w_n\} \mid$ non è nel dizionario

    1. Se valgono le seguenti condizioni:
        - Il primo carattere di w_i coincide con il primo carattere di almeno uno dei termini che compone il nome di una classe $c \in \mathscr{C}$
        - w_i non coincide con uno dei termini che compone il nome della classe c;
        - $metrics(name(c), w_i) \geq \Omega$

    $\Rightarrow$ aggiungi w_i al *cluster* c;

Dove *name(c)* indica il nome della classe c ridotto in minuscolo e senza spazi, *metrics* è la metrica che definisce la similarità dei termini e Ω è la soglia minima per accettare il termine nel *cluster*. Se più classi contengono nel proprio *cluster* lo stesso termine w_i, questo verrà aggiunto alla classe che ha maggiore similarità secondo la metrica.

Vediamo un esempio: consideriamo come nome della classe *Microsoft Windows Server 2003* e come termine, una sua storpiatura, *win2003*.

$$\text{“win2003”} \rightarrow \text{‘}w\text{’}$$

$$\text{“Microsoft Windows Server 2003”} \rightarrow \{\text{‘}m\text{’}, \text{‘}w\text{’}, \text{‘}s\text{’}, \text{‘2’}\}$$

Quindi la prima parte della condizione *AND* è soddisfatta poiché: ‘w’ $\in \{\text{‘}m\text{’}, \text{‘}w\text{’}, \text{‘}s\text{’}, \text{‘2’}\}$. La seconda parte controlla se “win2003” è contenuto in “*Microsoft Windows Server 2003*” che è falso. La terza e ultima parte applica la metrica:

$$\text{metrics}(\text{“microsoftwindowsserver2003”}, \text{“win2003”}) = v$$

Se $v \geq \Omega$ allora “*win2003*” verrà sostituito con “*Microsoft Windows Server 2003*”, ovvero entra nel *cluster* corrispondente alla classe.

Scelta della metrica

Con *string metric* si intende una metrica che misura la similarità (o la distanza) tra due stringhe di testo. Esistono diverse metriche utilizzabili, ma nel contesto descritto nella sezione precedente molte di esse non garantirebbero buone prestazioni:

- Distanza di Hamming: la similarità tra le stringhe è calcolata sulla base del numero di sostituzioni elementari necessarie per trasformare una stringa nell'altra. Questa metrica penalizzerebbe troppo le stringhe con lunghezza molto diversa (si veda l'esempio della sezione precedente);

- Distanza di Levenshtein e Damerau-Levenshtein [10]: fanno sempre parte dei metodi che applicano il concetto di *edit distance* (come la distanza di Hamming) e quindi soffrono della stessa debolezza legata alla lunghezza delle stringhe;

- Distanza di Jaro-Winkler [24]: metrica adatta alla misurazione della similarità tra stringhe brevi, come ad esempio nomi.

Le metriche appena descritte sarebbero troppo penalizzanti in situazioni simili all'esempio della Sezione 6.3.2 e quindi necessiterebbero di una soglia bassa la quale rischierebbe di creare troppi falsi positivi.
Esistono però una serie di metriche sviluppate appositamente per gestire lunghe sequenze di caratteri e che tendono a non penalizzare troppo la differenza di lunghezza. Questi algoritmi sono detti di allineamento e sono comunemente utilizzati in bioinformatica per l'allineamento di sequenze di nucleotidi. Sono stati testati due di questi metodi: Needleman–Wunsch [35] e Smith–Waterman [15].

Di seguito sono presentati i risultati dei test: in rosso sono indicati i *cluster* che non sono stati individuati col metodo manuale. I risultati evidenziano che si ha maggiore affidabilità applicando Smith–Waterman, il quale porta ad un leggero aumento delle prestazioni anche se non quanto ottenuto con il metodo manuale. La soglia utilizzata è $\Omega = 0.5$, scelto su un *validation set* e successivamente testato su un *test set*, entrambi pari al 10% del *dataset*. In generale valori > 0.5 causavano una drastica diminuzione del numero di *cluster* vanificando quindi questo *step*, mentre valori < 0.5 erano troppo permissivi e il numero di falsi positivi era troppo elevato.

Cluster	# Termini	Classe
1	0	2
2	1	6.3.1
3	0	6.3
4	5	6.3.2.1
5	3	6.3.2.2
6	1	9
7	1	1.1
8	1	11
9	1	1.3
10	2	6.1
11	2	6.2
12	1	9.1

Tabella 6.25: *Cluster* individuati con Smith-Waterman, $\Omega = 0.5$ (*False Positive* $= 1$).

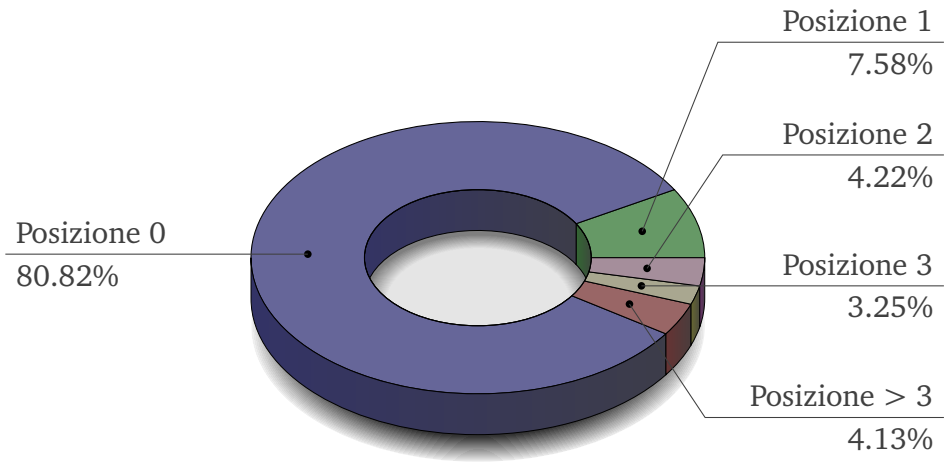


Figura 6.7: Prestazioni con l'adozione del *term clustering* automatico Smith-Waterman ($\Omega = 0.5$).

Diminuendo il valore di Ω (< 0.5) il numero dei *cluster* rimane invariato tuttavia aumentano il numero di falsi positivi. Con valori di Ω maggiori invece il numero dei cluster diminuisce causando una conseguente perdita in precisione.

Cluster	# Termini	Classe
1	0	2
2	1	6.3.1
3	0	6.3
4	7	6.3.2.1
5	7	6.3.2.2
6	1	9
7	1	1.1
8	1	11
9	1	1.3
10	0	6.1
11	4	6.2
12	1	9.1
13	5	4

Tabella 6.26: *Cluster* individuati con Needleman–Wunsch, $\Omega = 0.5$ (*False Positive* $= 43$).

Il numero di falsi positivi con questo algoritmo è troppo elevato e causa un degrado delle prestazioni. Utilizzando $\Omega > 0.5$ il numero dei *cluster* individuati diminuisce notevolmente e con esso anche il numero di falsi positivi. Le prestazioni però risultano inferiori rispetto a quelle ottenute con Smith-Waterman.

6.4 Aggiornamento del modello

Uno dei nuovi requisiti richiesti dall'azienda (FO_17) prevede che il modello si aggiorni ogni qualvolta classifica delle nuove *e-mail* prese dalla casella di posta elettronica. Il vecchio sistema è stato progettato per aggiornare il modello ogni notte avendolo disponibile solo il giorno successivo. Per poter soddisfare il requisito FO_17 è stato necessario effettuare delle modifiche al *thread* di *training* e di classificazione. La Figura 6.8 mostra i nuovi flussi dei *thread*.

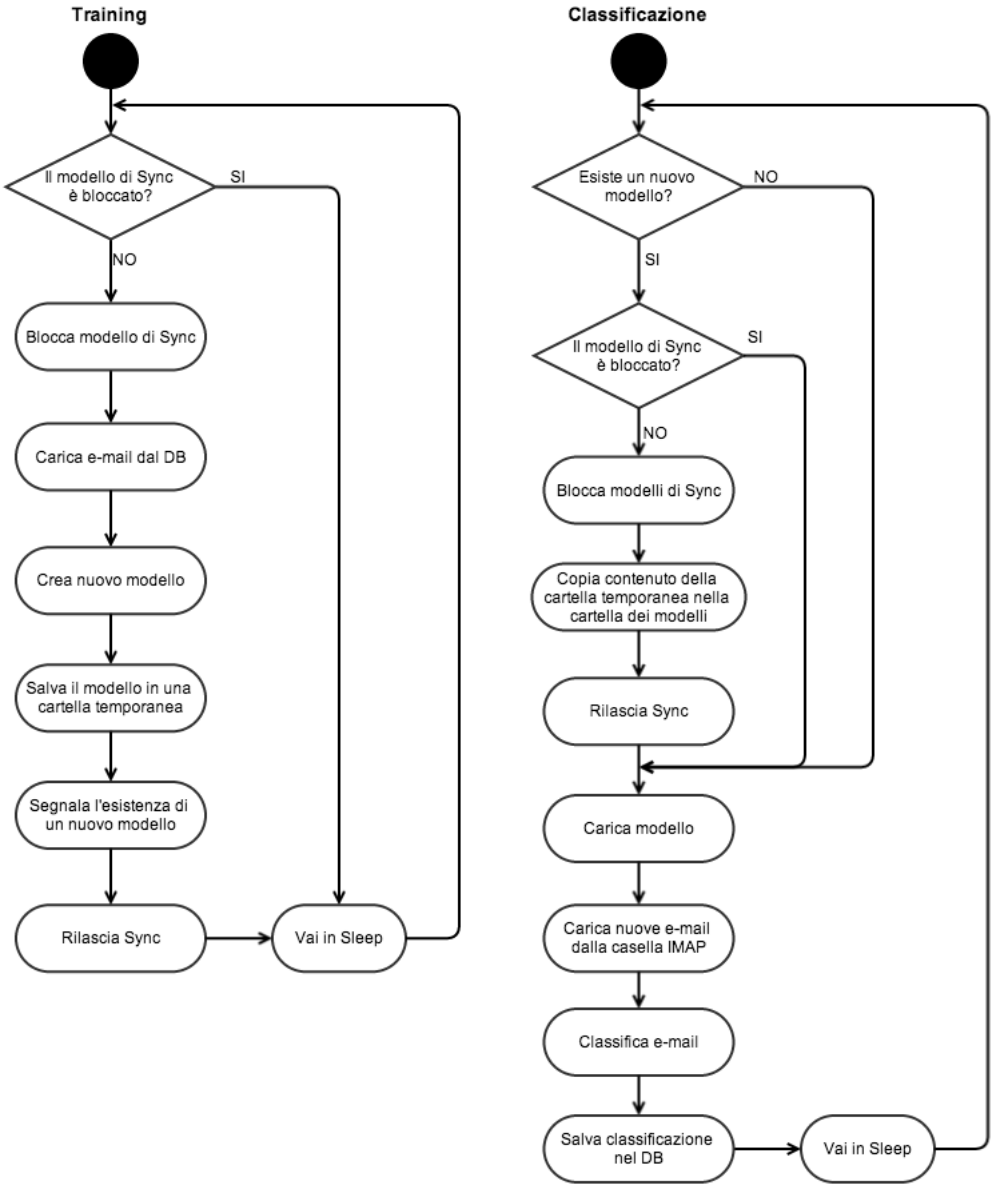


Figura 6.8: Diagramma di flusso dei nuovi *thread*.

6.5 Presentazione dei risultati e gestione dei *feedback*

Ogni volta che avviene una classificazione da parte del sistema, il messaggio classificato viene inserito nella tabella *ToBeConfirmed* del *database MySQL*. A questo punto un utente può visualizzare il risultato attraverso una pagina *web* dedicata, previa una registrazione al sistema di *feedback*. La pagina che presenta i risultati è resa disponibile da una *servlet* che gira sul *server* Tomcat, che deve essere installato in una macchina all'interno dell'azienda, in modo che i dati rimangano nella rete interna. La *servlet* offre i risultati della classificazione (prelevati dalla tabella *ToBeConfirmed*) in una tabella in cui nella prima colonna sono presenti gli oggetti delle *e-mail* classificate e nella seconda colonna la classificazione assegnata dal sistema. L'oggetto, all'interno della tabella, è un *link* che manda al testo completo del messaggio in modo che l'utente possa leggere il contenuto e quindi decidere la classificazione corretta. Per dare il proprio *feedback* l'addetto deve agire nel menu a tendina che mostra la classificazione proposta: se questa è corretta allora l'utente può semplicemente lasciare l'opzione selezionata, altrimenti sceglie quella più adatta. Il menu a tendina mostra le diverse opzioni nell'ordine indicato dal *ranking* ottenuto in fase di classificazione; in questo modo è probabile che la classificazione corretta si trovi nelle prime posizioni del menu diminuendo così il lavoro dell'addetto. Una volta selezionate le preferenze per ogni *e-mail* è sufficiente confermare. La *servlet*, infine, si occuperà di salvare questi messaggi confermati nella tabella *Confirmed* del DB, che rappresenta il *training set* per il prossimo ciclo di apprendimento.

6.5.1 Multiutenza

Essendo la pagina web pubblica, all'interno del sistema aziendale, è possibile che più utenti agiscano sulle stesse *e-mail* contemporaneamente. Questo fatto può portare a due situazioni particolari:

- Concordi: i due utenti assegnano per uno stesso messaggio la stessa classe;
- Discordi: gli utenti assegnano categorie diverse allo stesso messaggio.

Nel primo caso è sufficiente evitare che la copia della stessa *e-mail* venga aggiunta al DB. Per fare questo i messaggi all'interno della tabella *Confirmed* sono chiavi primarie, in questo modo la stessa *e-mail* con la stessa categoria confermata rappresenta effettivamente uno stesso oggetto, e viene quindi rifiutato

dal *DBMS*. Nel secondo caso, i due oggetti sono diversi e vengono quindi aggiunti al DB. La situazione così ottenuta è una copia di uno stesso messaggio per due categorie differenti: se due addetti umani hanno dato classificazione discordante significa che il messaggio (escludendo errori) è ambiguo e al suo interno contiene informazioni che riguardano contemporaneamente tutte e due le categorie. Il fatto che al prossimo ciclo di apprendimento, il messaggio venga utilizzato per alimentare due classi diverse è auspicabile poiché contiene informazioni utili per entrambe. Anche in questo caso quindi la situazione nel DB è corretta.

7

Riconoscimento automatico dei cambiamenti nella tassonomia

In questo capitolo verrà presentata la tecnica utilizzata per la gestione dei cambiamenti nella tassonomia nel corso del tempo. In particolare verrà posta grande attenzione al caso in cui alcuni nodi della tassonomia si stiano per specializzare causando la nascita di nuove sotto classi.

7.1 Classi obsolete

Con il trascorrere del tempo le tecnologie cambiano e con esse anche i sistemi che l'azienda offre ai propri clienti, questo porta inesorabilmente alla scomparsa di alcuni prodotti e come conseguenza si ha una drastica diminuzione del numero di messaggi che li riguardano. Se il numero di messaggi in un certo nodo (o sotto albero) della tassonomia comincia a essere irrisorio allora il nodo può essere definito obsoleto.
Per poter riconoscere la situazione appena descritta, il sistema deve tener traccia della distribuzione dei messaggi nel tempo, la quale verrà analizzata e indicato all'utente se e quali classi sono potenzialmente obsolete. Procedere alla cancellazione di queste categorie spetta all'addetto poiché una cancellazione prematura potrebbe comportare problemi nelle classificazioni future.

L'approccio adottato dal nuovo sistema è il seguente: ogni N messaggi viene calcolato il numero di *e-mail* di ogni classe negli ultimi M mesi. Se questo numero è inferiore ad una certa soglia S allora la classe viene segnalata come obsoleta. I parametri N, M, ed S vengono definiti dall'azienda che dispone di maggiori informazioni per poter dare una stima corretta riguardo alla distribuzione dei messaggi.

7.2 Specializzazione

L'utilizzo della tassonomia di classi agevola notevolmente il processo di ricerca e salvataggio delle informazioni aziendali: è molto più semplice gestire informazioni disposte su una struttura gerarchica piuttosto che una mera cartella non strutturata. Uno dei motivi principali è che il numero dei messaggi in ogni nodo dovrebbe essere contenuto. Questa situazione con il tempo potrebbe non essere più vera: è possibile che una certa classe si ritrovi un numero di *e-mail* troppo elevato per poter garantire una gestione efficiente. Il problema può essere risolto da un umano che periodicamente controlla le cartelle e tenta di ristrutturarle: questo compito però non è banale e richiede tempo. Per questo motivo l'azienda ha richiesto un *tool* che propone nuove specializzazioni, le quali possono essere confermate oppure rifiutate dagli addetti.

Il problema descritto in questa sezione può essere interpretato come un problema di individuazione di *cluster* in un insieme di dati. Esistono due macro categorie di algoritmi di *clustering*: una richiede sia definito a priori il numero di *cluster* da individuare (es. *k-Means, Expectation Maximization*), l'altra in cui il parametro non deve essere definito (es. *Hierarchical Clustering, DBSCAN, OPTICS*). In questa seconda categoria possono essere inseriti anche algoritmi basati su reti neurali come le *Self Organizing Map* e le *ART neural network*. Nel contesto della specializzazione tassonomica la categoria più adatta è la seconda descritta in questa sezione e in particolare verranno prese in considerazione le reti neurali *ART*.
Nonostante il sistema attuale sia di tipo *batch*, l'intenzione è di renderlo più simile possibile ad un sistema *online* (infatti il *training* del modello avviene ad ogni nuovo lotto *e-mail* classificate), e le reti *ART* sono note per essere molto veloce e adattive rispetto alla distribuzione dei dati in ingresso.

7.3 *Adaptive Resonance Theory* (*ART*)

La teoria della risonanza adattiva (*Adaptive Resonance Theory*, *ART*) è stata sviluppata da Carpenter e Grossberg [18] su come il cervello umano elabora le informazioni. Questa teoria è nata dalla necessità di risolvere il cosiddetto *Stability-Plasticity Dilemma*, ovvero:

Stability-Plasticity Dilemma. *Come può un sistema essere sufficientemente adattivo da gestire eventi significativi e allo stesso tempo essere abbastanza stabile da ignorare eventi irrilevanti?*

Sulla base della teoria *ART* Carpenter e Grossberg svilupparono una serie di modelli, supervisionati e non supervisionati, per gestire differenti tipi di input (*pattern*) [42]:

- *ART1* [18]: rete neurale con input di tipo binario (non supervisionato);
- *FuzzyART*: rete neurale con input di tipo *fuzzy* (non supervisionato);
- *ART2* [17]: rete neurale in grado di gestire input reali (non supervisionato);
- *ARTMAP* [23]: noto anche come *Predictive ART*, combina due modelli ART1 (o ART2) in una struttura supervisionata.

7.3.1 Architettura di base

L'architettura di base di una rete neurale a risonanza adattiva è costituita da tre gruppi di neuroni: un livello di *input* F_1, un livello di *output* F_2, che contiene le unità *cluster* e un meccanismo per il controllo della similarità dei *pattern* all'interno di uno stesso cluster (meccanismo di *Reset*). Il livello F_1 può essere suddiviso in due parti: una parte per la gestione dell'ingresso ($F_{1(a)}$) e una per interfacciarsi ($F_{1(b)}$) al livello F_2. Generalmente la prima parte si occupa di ricevere l'input (*ART1*), talvolta invece avviene anche una piccola fase di *processing* (*ART2*). La porzione di interfacciamento invece combina segnali di $F_{1(a)}$ con quelli del livello F_2.
Per controllare la similarità tra *pattern* che appartengono allo stesso *cluster* esistono due gruppi di connessioni, una per ogni unità in ingresso e una per ogni unità *cluster*. Ogni unità del livello $F_{1(b)}$ è connessa con ogni unità del livello F_2 da collegamenti pesati (*bottom up)*. Analogamente il livello F_2 è connesso alla porzione $F_{1(b)}$ da collegamenti pesati (*top down*). Il livello F_2 è detto competitivo: l'unità cluster con *input* maggiore è la candidata all'apprendimento del *pattern* in ingresso. Inizialmente ogni unità di F_2 è inattiva, ovvero non è in grado di apprendere. Per avere la capacità di apprendimento del *pattern* l'unità deve avere i propri pesi *top down* sufficientemente simili al vettore di *input*. La decisione è presa dall'unità di *Reset* sulla base delle unità in $F_{1(a)}$ e $F_{1(b)}$. Se un'unità non è in grado di apprendere viene inibita. La Figura 7.1 mostra l'architettura generale di una rete ART.

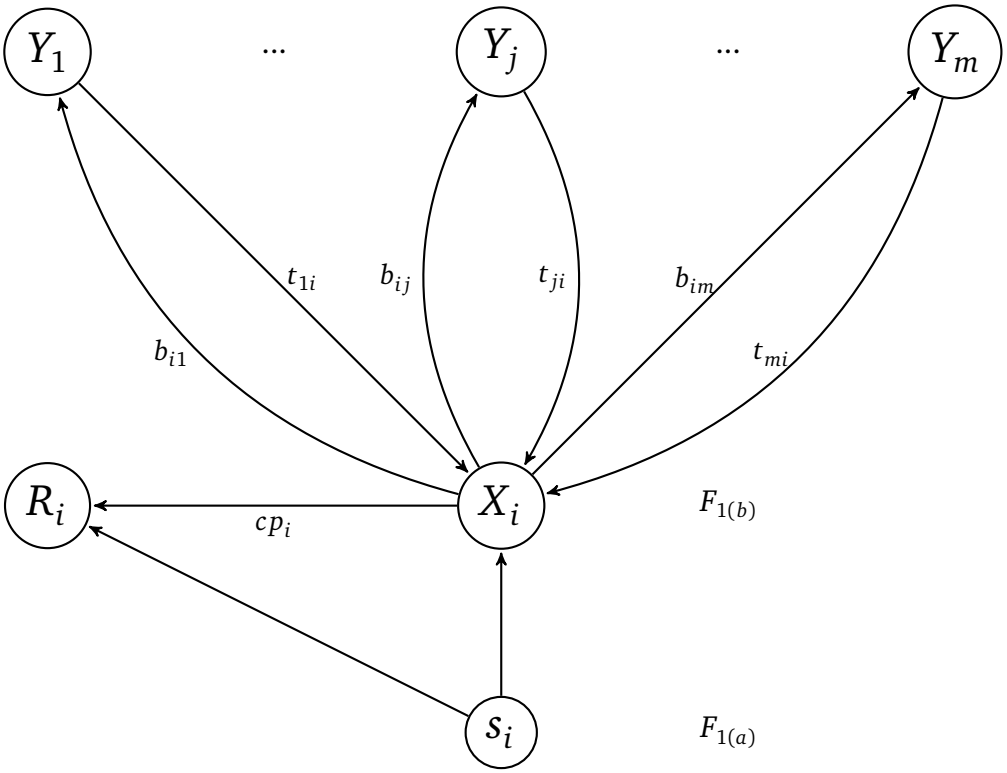


Figura 7.1: Architettura base di una *ART*: la variabile S_i rappresenta il *pattern* in ingresso (è raffigurata una sola unità, $i \in 1, ..., n$), R_i è il meccanismo di *Reset* e X_i è lo strato di interfacciamento con il livello F_2 che è rappresentato dalle variabili Y_j con $j \in 1, ..., m$. b_{ij} e t_{ji} sono rispettivamente i pesi *bottom up* e *top down*.

Questa architettura generale rappresenta la base per ogni modello ART, che in base alle proprie necessità, aggiunge unità di controllo necessarie per elaborare le informazioni in ingresso. Nella prossima sezione viene presentato il modello *ART2*, in grado di gestire *input* reali. Questo algoritmo verrà poi utilizzato per la fase di *clustering* utile alla risoluzione del problema descritto nella Sezione 7.2.

7.3.2 ART2

ART2 è stato progettato per riuscire a gestire vettori di ingresso a valori reali. Per questa ragione lo strato F_1 viene arricchito per accomodare *input pattern* reali: include una combinazione di normalizzazioni e rimozione di rumore.

Architettura generale

L'architettura tipica di una *ART2* [20] è illustrata in Figura 7.2. Lo strato F_1 consiste di sei tipi di unità (W, X, U, V, P e Q). Ogni unità è costituita da un vettore n-dimensionale, dove n è la lunghezza del vettore *pattern*. Esistono inoltre tre unità supplementari tra le unità $W \rightarrow X$, $V \rightarrow U$ e $P \rightarrow Q$ le quali calcolano le norme, rispettivamente dei vettori $\vec{w}$, $\vec{v}$ e $\vec{p}$ e mandano questo segnale inibitorio alle unità X, U e Q. I simboli posti nei collegamenti tra le unità del livello F_1, in Figura 7.2, indicano le trasformazioni che avvengono ai segnali (vettori) quando passano da un unità alla successiva. Mentre i simboli sui collegamenti tra le unità P dello strato F_1 e le unità Y di F_2 rappresentano i pesi che moltiplicano i segnali trasmessi attraverso quelle connessioni. Le funzioni del livello F_2 rimangono invariate rispetto all'architettura base delle reti ART: le unità competono in modalità *winner-take-all* per poter apprendere gli *input pattern* e vengono inibite se non sono abbastanza simili al vettore d'ingresso.

Algoritmo

Nella prima parte di questa sezione verrà data una descrizione dell'algoritmo di apprendimento, e nella seconda verrà fornito l'algoritmo passo per passo. La notazione utilizzata è la seguente:

- n numero di componenti del vettore di *input*;
- m numero massimo di *cluster*;
- b_{ij} pesi *bottom up*, da P_i a Y_j;
- t_{ji} pesi *top down*, da Y_j a P_i;
- ρ parametro di vigilanza;
- $||\vec{a}||$ norma del vettore a.

Un *learning trial* (presentazione di un nuovo *pattern*) consiste nell'elaborare il vettore d'ingresso $\vec{S} = (s_1, s_2, ..., s_n)$ fino a che non viene appreso da una *cluster unit* o viene rifiutato. Inizialmente, nel primo ciclo del *learning trial*, vengono

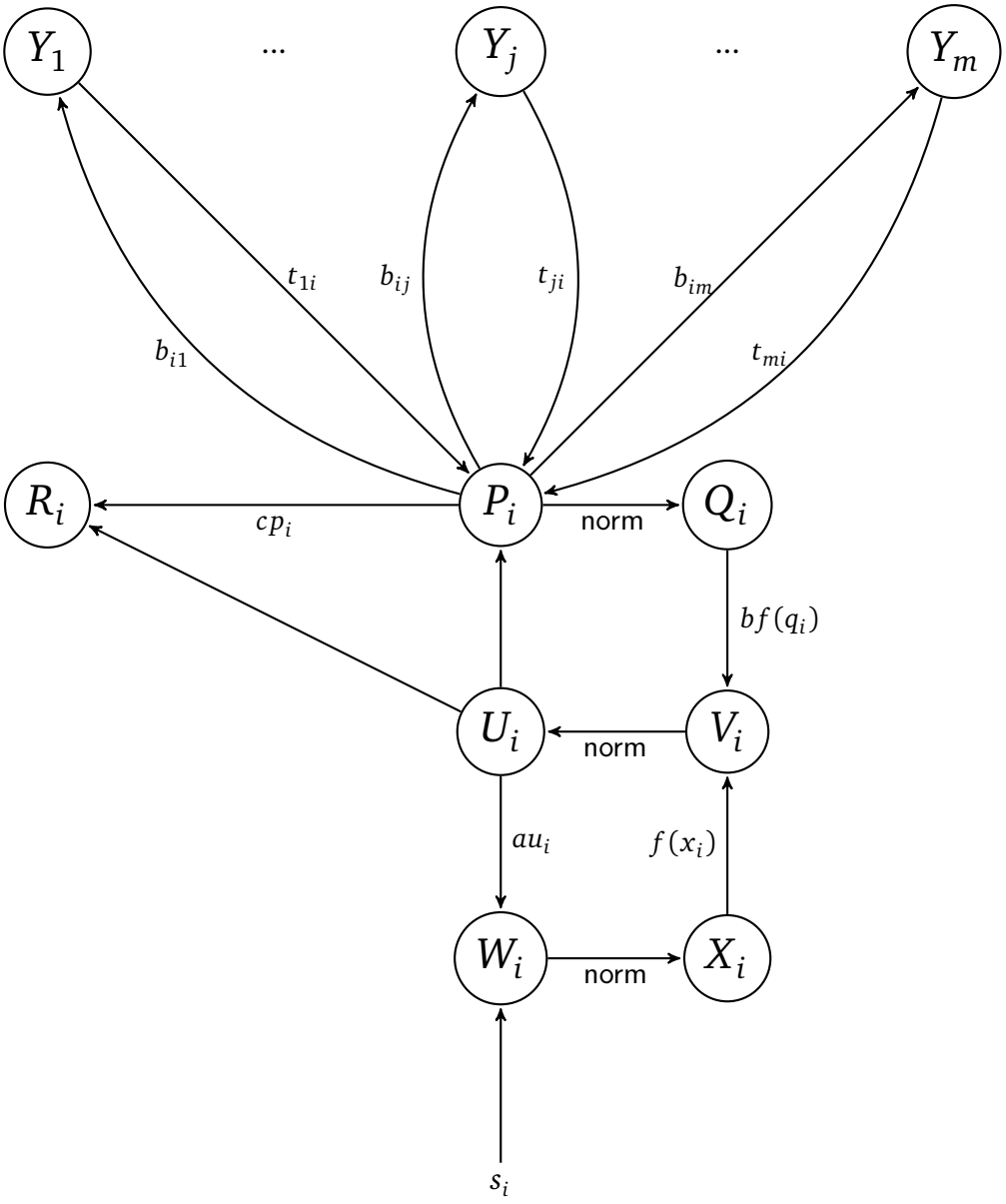


Figura 7.2: Architettura della rete neurale *ART2*: la variabile s_i rappresenta il *pattern* in ingresso (è raffigurata una sola unità, $i \in 1, ..., n$), R_i è il meccanismo di *Reset* e le variabili Y_j con $j \in 1, ..., m$ corrispondono al livello F_2 che è collegato al livello sottostante da b_{ij} e t_{ji} che sono rispettivamente i pesi *bottom up* e *top down*. Il livello F_1 è caratterizzato dall'insieme di variabili W_i, X_i, U_i, V_i, P_i e Q_i.

calcolati i valori delle unità U_i. Successivamente un segnale viene inviato alle unità corrispondenti W_i e P_i. Le unità in W_i sommano il segnale in ingresso da U_i con s_i, mentre P_i somma le U_i con il segnale *top down* se il corrispettivo Y_i dello strato F_2 è attivo. X_i e Q_i sono normalizzazioni rispettivamente di W_i e P_i. Queste due unità vengono poi manipolate da una funzione di attivazione f e sommate in V_i. Questo completa un ciclo di aggiornamento dello strato F_1. La funzione di attivazione è definita come:

$$f(x) = \begin{cases} x & \text{se } x \geq \theta \\ 0 & \text{se } x < \theta \end{cases}$$

grazie ad essa le attivazioni di P e Q tendono all'equilibrio dopo due cicli di aggiornamento del livello F_1. Lo scopo di questa funzione è di rimuovere il rumore dai dati in modo da aiutare la rete a raggiungere una struttura stabile di *cluster*. Una rete viene definita stabile quando il primo *cluster* scelto per ogni *pattern* viene accettato, o, in altre parole, non avvengono *reset*.
Quando il livello F_1 ha raggiunto l'equilibrio, le unità P inviano i loro segnali allo strato superiore F_2 dove avviene una competizione *winner-take-all* in modo da scegliere il *cluster* candidato. A questo punto viene attivato il meccanismo di *reset* che controlla se il candidato è sufficientemente simile al vettore *pattern* da poterlo apprendere, in caso contrario viene scartato e lo *unit cluster* inibito. Questo processo di scelta e controllo del candidato continua finché il *pattern* non viene appreso da un unità *cluster* oppure tutte le unità sono inibite, in quel caso il *pattern* viene ignorato. Per un candidato apprendere il *pattern* significa aggiornare i propri pesi *bottom up* e *top down*. Se ART è progettato per un apprendimento lento questo aggiornamento avviene una sola volta per ogni *learning trial*, nel caso di apprendimento veloce l' aggiornamento dei pesi avviene finché non è raggiunto l'equilibrio, tali iterazioni vengono dette epoche. Si può notare che l'apprendimento delle reti *ART* è incrementale e quindi è facilmente adattabile ad un contesto *on-line*.
Di seguito viene illustrato lo pseudo codice dell'algoritmo di apprendimento con ART2.

TrainingART2(m, n, *N_EP*, *N_IT*)

1. Inizializzazione dei parametri: $a, b, c, d, e, \theta, \rho, \alpha$;
2. Per *N_EP* volte fai
    1. Per ogni *input pattern*
        1. Aggiorna l'unità F_1

$$u_i = 0, \quad x_i = \frac{s_i}{||e + \vec{s}||},$$
$$w_i = s_i, \quad q_i = 0,$$
$$p_i = 0, \quad v_i = f(x_i).$$

2. Aggiorna F_1 una seconda volta

$$u_i = \frac{v_i}{e + ||\vec{v}||}$$
$$w_i = s_i + a u_i$$
$$p_i = u_i$$
$$x_i = \frac{w_i}{e + ||\vec{w}||}$$
$$q_i = \frac{p_i}{e + ||\vec{p}||}$$
$$v_i = f(x_i) + b f(q_i)$$

3. Calcola il segnale delle unità F_2

$$y_j = \sum_{i=1}^{n} b_{ij} p_i$$

4. Finché avviene il *reset*

   1. Trova Y_J con il segnale maggiore

   $$\exists J \text{ t.c. } y_J \geq y_j \text{ per } j = 1, ..., m$$

   2. Controllo *reset*

$$u_i = \frac{v_i}{e + ||\vec{v}||}$$
$$p_i = u_i + d\, t_{Ji}$$
$$r_i = \frac{u_i + c p_i}{e + ||\vec{u}|| + c||\vec{p}||}$$

Se $||\vec{r}|| < \rho - e \Rightarrow y_J = -1$, *reset* è vero, vai a *(2.4)*
Altrimenti $||\vec{r}|| \geq \rho - e \Rightarrow$ *reset* è falso

$$w_i = s_i + au_i$$
$$x_i = \frac{w_i}{e + ||\vec{w}||}$$
$$q_i = \frac{p_i}{e + ||\vec{p}||}$$
$$v_i = f(x_i) + bf(q_i)$$

3. Per N_IT volte fai (*risonanza*)
   1. Aggiorna i pesi dell'unità *cluster* vincitrice

$$t_{Ji} = \alpha d u_i + [1 + \alpha d(d-1)] t_{Ji}$$
$$b_{ij} = \alpha d u_i + [1 + \alpha d(d-1)] b_{iJ}$$

   2. Aggiorna lo strato F_1

$$u_i = \frac{v_i}{e + ||\vec{v}||}$$
$$w_i = s_i + au_i$$
$$p_i = u_i + d\, t_{Ji}$$
$$x_i = \frac{w_i}{e + ||\vec{w}||}$$
$$q_i = \frac{p_i}{e + ||\vec{p}||}$$
$$v_i = f(x_i) + bf(q_i)$$

Scelta dei parametri

In questa sezione viene descritto il ruolo dei parametri all'interno dell'algoritmo.

n numero delle unità di *input* (livello F_1);

m numero delle unità di *cluster* (livello F_2);

N_EP numero di epoche: volte in cui vengono eseguiti tutti i *learning trial*;

N_IT numero di iterazioni: volte per cui avviene il ciclo di risonanza;

a, b pesi utilizzati nello strato F_1. Con $a = 0$ o $b = 0$ la rete sarebbe molto instabile, mentre qualsiasi valore diverso da zero la renderebbe più stabile. La rete non è molto sensibile alle variazione di a e b, solitamente sono fissati a 10;

c parametro utilizzato nel controllo del *reset*. Valori piccoli di c producono maggiore efficacia del parametro di vigilanza, solitamente è fissato a 0.1;

d valore di attivazione dell'unità vincitrice. Da notare che i parametri d e c devono soddisfare la disequazione:

$$\frac{cd}{1-d} \leq 1$$

il valore comunemente utilizzato per d è 0.9;

e parametro che evita la divisione per 0 quando la norma del vettore è 0.

θ parametro per la riduzione del rumore, solitamente fissato a $1/\sqrt{n}$;

α parametro di apprendimento, detto *learning rate*. Un valore piccolo riduce la velocità di apprendimento, ma aiuta il raggiungimento dell'equilibrio anche in modalità *slow learning*;

ρ parametro di vigilanza. Assieme al valore iniziale dei pesi *bottom up*, questo parametro determina il numero di *cluster* che si formeranno. Teoricamente i valori ammissibili per ρ sono l'intervallo $[0, 1]$. Spesso tale parametro viene fissato ad un valore non inferiore a 0.7;

$t_{ji}(0)$ valore iniziale dei pesi *top down*: devono essere sufficientemente piccoli da garantire che non ci sia un *reset* al primo *pattern*. Tipicamente sono fissati tutti a 0;

$b_{ij}(0)$ valore iniziale dei pesi *bottom up*: devono essere scelti in modo che soddisfino la disequazione:

$$b_{ij}(0) \leq \frac{1}{(1-d)\sqrt{n}}$$

in modo da prevenire la possibilità che un nuovo vincitore venga scelto durante il periodo di risonanza. Valori alti di $b_{ij}(0)$ incoraggiano la rete a formare più *cluster*.

7.4 Applicazione di ART2

Massey [29] e Card et al. [12] hanno mostrato come le reti ART possano raggiungere buoni risultati nel *text clustering*. Nel contesto della specializzazione tassonomica, presentato nella Sezione 7.2, l'utilizzo dell'algoritmo ART deve essere accompagnato da alcune operazioni al contorno che tengono conto di due importanti considerazioni:

- *Cluster* troppo piccoli non possono essere considerati specializzazioni poiché non contengono abbastanza evidenze;
- *Cluster* troppo grandi non possono essere considerati specializzazioni poiché coprono gran parte della classe di appartenenza e quindi la specializzazione è la classe stessa;

Per i motivi appena descritti l'algoritmo che propone le specializzazioni è sviluppato come segue:

TaxSpecializationDetectionART2

1. Per ogni classe $c \in \mathscr{C}$ t.c. c contiene messaggi nel Tr
    1. $Tr_c \leftarrow$ messaggi di Tr classificati come c;
    2. $\vec{S} \leftarrow$ applicazione di **ART2**, dove gli *input pattern* sono i messaggi in Tr_c;
    3. Scarta dall'insieme dei *cluster* $\vec{S}$ quelli con un numero di messaggi $\leq \frac{|Tr_c|}{10}$
    4. Scarta dall'insieme dei *cluster* $\vec{S}$ quelli con un numero di messaggi $\geq \frac{9}{10}|Tr_c|$
    5. Per ogni $s \in \vec{S}$
        1. Estrai le 10 *feature* più presenti nei messaggi contenuti nel *cluster* s;
    6. Ritorna $\vec{S}$ e la lista di *feature* associata;

7.5 Test

In questa sezione verranno presentati i risultati dei test effettuati sull'algoritmo *ART2* descritto nelle sezioni precedenti. *ART2* verrà confrontato con due algoritmi di *clustering* molto noti:

- *k-Means*: è un algoritmo di *clustering* che tenta di partizionare un insieme di n oggetti in k sottoinsiemi con l'obiettivo di minimizzare la varianza *intra-cluster*. Formalmente, dato un insieme di oggetti $\langle x_0, x_1, ..., x_n \rangle$ dove ogni x_i è un vettore reale di dimensione d e un intero k, l'algoritmo mira a creare k *cluster* ($S = \{S_1, S_2, ..., S_k\}$) in cui sia minimizzata la *within-cluster sum of squares*, ovvero:

  $$\underset{S}{\operatorname{argmin}} \sum_{i=1}^{k} \sum_{x_j \in S_i} ||x_j - \mu_i||^2$$

  dove μ_i è la media dei punti di S_i;

- *Expectation Maximization* (*EM*): è un algoritmo iterativo che ha l'intento di trovare l'ipotesi *maximum likelihood* assumendo che i dati siano generati da più distribuzioni gaussiane. È caratterizzato da due *macro-step*: nel primo (*E: expectation*) vengono calcolate le probabilità che un dato appartenga ad una certa classe, nel secondo (*M: maximization*) vengono alterati i parametri che caratterizzano la distribuzione di ogni classe per massimizzare le probabilità stimate nello *step* precedente.

Questi due algoritmi, oltre a differire nell'approccio necessitano di *input* diversi: *k-Means* ha bisogno a priori del numero di *cluster* k da generare, mentre per *EM* tale parametro può essere omesso (come anche per *ART2*).

Per valutare questi metodi sono state utilizzate due metriche:

- *Purity (Pu)*: calcola la frazione di esempi della classe maggioritaria all'interno del *cluster*. La *purity* può assumere valori compresi tra 0 e 1 e da un'idea della purezza dei *cluster* creati. Tuttavia questa metrica non sempre ritorna valori attendibili soprattutto quando il numero di *cluster* è molto elevato. Infatti, se si assegna ogni singolo esempio ad un *cluster* differente la purezza avrà il valore massimo.

- *Rand Index (RI)*: equivale all'*accuracy* descritta nel Capitolo 2, infatti è definita come:

  $$RI = \frac{TP + TN}{TP + FP + TN + FN}$$

dove TP e FP sono rispettivamente i *true positive* ovvero le coppie di esempi della stessa classe che appartengono allo stesso *cluster* e i *false positive*, ovvero le coppie di esempi che stanno nello stesso *cluster* appartenendo però a classi diverse. In modo analogo sono definiti gli TN *true negative* e gli FN *false negative*.

Parametri utilizzati

Come per *SVM*, anche in questo caso la scelta dei parametri gioca un ruolo importante. Tuttavia effettuare una ricerca esaustiva sarebbe risultato troppo oneroso e quindi si è preferito effettuare una scelta ponderata basandoci sulle esigenze esposte nelle sezioni precedenti. Di seguito sono elencati i valori scelti per i parametri affiancati da una breve motivazione:

$\boldsymbol{m = 30}$ più che sufficiente a gestire la totalità dei casi presi in esame;

$\boldsymbol{N\_IT = 1}$ il ciclo di risonanza viene eseguito una singola volta, come per la maggior parte delle configurazioni *ART*;

$\boldsymbol{a = b = 10}$ valore standard, diverso da 0, utile alla stabilità della rete;

$\boldsymbol{c = 0.1}$ valore vicino allo 0 in modo da rendere più efficace il parametro di vigilanza;

$\boldsymbol{d = 0.9}$ valore valore standard che rispetta la disequazione $\frac{cd}{1-d} \leq 1$;

$\boldsymbol{e = 0.01}$ valore molto vicino allo zero che evita la divisione per 0 nella norma ma che non crea squilibrio nei calcoli;

$\boldsymbol{\alpha = 0.5}$ per un apprendimento equilibrato;

$\boldsymbol{\rho = 0.9}$ valore molto vicino a 1 che tende a creare un maggior numero di *cluster*.

I test sono stati eseguiti sui nodi interni della tassonomia, in modo da confrontare le proposte dei vari algoritmi rispetto alla reale struttura gerarchica delle classi. Di seguito sono riportati i risultati: si noti che il parametro k, *input* del metodo *k-Means*, è fissato al numero di classi in quel sottoalbero al primo livello (sono state tralasciate le classi con un numero di esempi inferiore a 20). Come si può notare dai risultati delle Tabelle 7.1, 7.2 e 7.3, le prestazioni dei 3 algoritmi sono comparabili e nella maggior parte dei casi si eguagliano. Da notare però che *KM*, come detto precedentemente, è avvantaggiato dal fatto che k viene definito correttamente a priori. Il test, contrassegnato con (*), sulla classe 0 (la radice della tassonomia) è stato eseguito fissando il numero

	Classe 0*			Classe 1			Classe 6		
Algoritmo	**# Cl**	**Pu**	**RI**	**# Cl**	**Pu**	**RI**	**# Cl**	**Pu**	**RI**
ART2	6/12	0.24	0.68	2/3	0.88	0.59	4/3	0.62	0.52
EM	12/12	0.24	0.56	3/3	0.88	0.65	2/3	0.62	0.49
KM	12/12	0.24	0.59	3/3	0.9	0.77	3/3	0.62	0.48

Tabella 7.1: Confronto tra *ART2*, *EM* e *KM*. *# Cl* indica il numero di *cluster* individuato dall'algoritmo rispetto al numero di classi effettive, *Pu* è la *Purity* e *RI* il *Rand Index*. (Parte 1/3)

	Classe 9			Classe 23			Classe 6.3		
Algoritmo	**# Cl**	**Pu**	**RI**	**# Cl**	**Pu**	**RI**	**# Cl**	**Pu**	**RI**
ART2	2/7	0.62	0.48	2/10	0.53	0.47	3/2	0.9	0.45
EM	4/7	0.62	0.51	2/10	0.55	0.34	3/2	0.9	0.52
KM	7/7	0.63	0.47	10/10	0.61	0.42	2/2	0.9	0.68

Tabella 7.2: Confronto tra *ART2*, *EM* e *KM*. *# Cl* indica il numero di *cluster* individuato dall'algoritmo rispetto al numero di classi effettive, *Pu* è la *Purity* e *RI* il *Rand Index*. (Parte 2/3)

	Classe 6.3.2			Classe 1.3		
Algoritmo	**# Cl**	**Pu**	**RI**	**# Cl**	**Pu**	**RI**
ART2	2/2	0.6	0.51	2/6	0.39	0.22
EM	4/2	0.55	0.49	1/6	0.39	0.22
KM	2/2	0.53	0.49	6/6	0.44	0.44

Tabella 7.3: Confronto tra *ART2*, *EM* e *KM*. *# Cl* indica il numero di *cluster* individuato dall'algoritmo rispetto al numero di classi effettive, *Pu* è la *Purity* e *RI* il *Rand Index*. (Parte 3/3)

di iterazioni di *EM* pari a 1 altrimenti il tempo di esecuzione sarebbe risultato troppo elevato.
Sostanzialmente *ART2* raggiunge, in questo contesto, risultati simili agli altri metodi considerati, con tempi di esecuzione molto inferiori ad *EM* ed avendo la proprietà di essere *online*, a differenza degli altri due algoritmi.

8

Conclusioni e sviluppi futuri

Il sistema realizzato nel corso dell'attività di tesi ha condotto al raggiungimento di risultati decisamente soddisfacenti. La conoscenza del dominio applicativo e le conseguenti considerazioni fatte sui messaggi di posta elettronica hanno portato ad un aumento delle prestazioni rispetto al sistema preesistente. Una delle modifiche che ha portato maggiore beneficio è stata l'introduzione del coefficiente moltiplicativo per le *feature* all'interno dell'oggetto. Al primo livello della tassonomia vengono collocate oltre il 95% delle *e-mail* in prima posizione e quasi il 99% tra le prime due. Passando all'intera gerarchia di classi le *performance* si riducono all'82%, il quale rappresenta comunque un ottimo risultato considerando che la tassonomia contiene oltre 60 categorie. In questa fase è stato fondamentale l'ausilio del *term clustering*, dapprima manuale e in seguito automatico.
I risultati appena esposti sono stati raggiunti utilizzando come algoritmo di classificazione *Naïve Bayes* anche se, come descritto nel Capitolo 6, con *SVM* le prestazioni al primo livello si sono dimostrate comparabili. L'ausilio delle *Support Vector Machine* però richiede una fase di calibrazione dei parametri molto costosa e quindi si è deciso di non utilizzare tale approccio nella classificazione gerarchica.
In progetti di questo tipo, che realizza un sistema di classificazione di documenti testuali, l'obiettivo principale è chiaramente garantire la più alta percentuale possibile di classificazioni corrette sul totale di documenti ma, come si è potuto notare dai capitoli precedenti, non rappresenta l'unico aspetto importante. Infatti, essendo i messaggi di posta elettronica, una fonte di conoscenza per gli addetti dell'azienda coinvolta, anche la gestione della gerarchia di classi, soprattutto in settori che sono in continua evoluzione, costituisce un tassello fondamentale per garantire ordine e quindi un aumento della produttività all'interno del sistema aziendale. Per questo motivo il lavoro di tesi si è in parte focalizzato sullo sviluppo di un sistema semi-automatico che propone agli utenti eventuali modifiche della tassonomia di classi. Si è deciso di utilizzare un algoritmo di *clustering* adattivo e *online*, *ART2*, in modo da garantire

velocità e adattabilità. I risultati hanno mostrato come questo approccio raggiunga prestazioni comparabili a tecniche ben più note e attuali come *k-Means* e *EM*, con il vantaggio però di essere *online*.

Oltre alle modifiche sull'algoritmo di classificazione, in tutte le sue fasi, è stato poi necessario rivedere il meccanismo di *feedback* da parte degli utenti, poiché l'approccio precedente, basato sullo scambio di file *XML*, non era più adottabile a causa del cambiamento del sistema *OTRS* aziendale. È stato quindi creata una *Servlet*, resa disponibile attraverso un *Web Server* posto all'interno dell'azienda, che pubblica i risultati delle classificazione in una pagina *web*. In questo modo, gli utenti, una volta collegati e autenticati al servizio, possono offrire il loro *feedback* che verrà utilizzato dal sistema di classificazione per aggiornare il proprio modello. A differenza del sistema preesistente, in cui la fase di apprendimento avveniva una sola volta al giorno (durante la notte), il nuovo sistema aggiorna il proprio modello ogniqualvolta uno o più *feedback* vengono salvati all'interno della base di dati.

8.1 Possibili sviluppi futuri

Il sistema ha soddisfatto la totalità dei requisiti obbligatori e gran parte dei requisiti desiderabili descritti nel Capitolo 4. Come detto nella sezione precedente a causa della modifica del sistema *OTRS* adottato dall'azienda è stata modificata la procedura di *feedback*. In futuro potrebbe essere opportuno reintegrare il classificatore all'interno di questo sistema, come fatto in precedenza, tramite lo scambio di file *XML*.
Per quanto riguarda la gestione della tassonomia, oltre al già presente meccanismo che si occupa di proporre modifiche alla struttura, si potrebbe introdurre un sistema che, sulla base dell'accettazione delle proposte (o parte di esse) offerte dal sistema, riorganizzi la tassonomia in modo automatico.
Infine, per aumentare la qualità del classificatore, si potrebbe usufruire di informazioni di contesto attualmente presente in azienda (come la configurazione della infrastruttura *hardware* e *software* detenuta dal cliente) per la preselezione di un insieme ristretto di classi o per l'utilizzo di dizionari specializzati.

Bibliografia

[1] R. Kirkby B. Pfahringer A. Bifet, G. Holmes. Data stram mining: A practical approach, Maggio 2011.

[2] F. Aiolli A. Kopliku, A. Sperduti. Automatic email classification. Tesi di laurea specialistica, 2008.

[3] R. Dale A. Lampert and C. Paris. Segmenting email message text into zones. Proceeding EMNLP '09 Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2, 2009.

[4] Wee-Keong Ng Aixin Sun, Ee-Peng Lim. Performance measurement framework for hierarchical text classification, 2003.

[5] P. Raghavan e H. Schutze C. D. Manning. Introduction to information retrieval, 2008.

[6] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[7] P. Clark and T. Niblett. The cn_2 induction algorithm. Machine Learning, 3(4):261283, 1989.

[8] W. W. Cohen. Learning rules that classify e-mail. In Proceedings of AAAI Spring Symposium on Machine Learning and Information Retrieval, 1996.

[9] M. Sahami D. Koller. Hierarchically classifying documents using very few words. Proc. of the 14th Int. Conf. on Machine Learning, 1997.

[10] F.J. Damerau. A technique for computer detection and correction of spelling errors, Marzo 1964.

[11] Craig Stanfill e David Waltz. Toward memory-based reasoning. Communications of the ACM, v.29 n.12, p.1213-1228, Dicembre 1986.

[12] N. Vlajic e H.C. Card. Categorizing web pages using modified art. Electrical and Computer Engineering, Vol. 1, Maggio 1998.

[13] E. Giacoletto e K. Aberer. Automatic expansion of manual email classifications based on text analysis. In Proceedings of ODBASE'2003, the 2nd International Conference on Ontologies, Databases, and Applications of Semantics, pp.785 802, 2003.

[14] D. D. Lewis e K.A. Knowles. Threading electronic mail: A preliminary study. Information Processing and Management 33:2,209-217, 1997.

[15] T. F. Smith e M. S. Waterman. Identification of common molecular subsequences. Journal of molecular biology, Vol. 147, No. 1, Marzo 1981.

[16] Terry R. Payne e Peter Edwards. Interface agents that learn: An investigation of learning issues in a mail agent interface, 1995.

[17] G. A. Carpenter e S. Grossberg. Art 2: self-organization of stable category recognition codes for analog input patterns. Applied Optics, Vol. 26, Dicembre 1987.

[18] G. A. Carpenter e S. Grossberg. A massively parallel architecture for a self-organizing neural pattern recognition machine. Computer Vision, Graphics, and Image Processing, Vol. 37, Gennaio 1987.

[19] S. Kiritchenko e S. Matwin. Email classification with co-training. In Proceedings of the 2001 Conference of the Centre for Advanced Studies on Collaborative Research, 2001.

[20] L. Fausett. Foundamental of neural networks: architectures, algorithms and applications, Dicembre 1993.

[21] D. Wu e V. N. Vapnik. H. Drucker. Support vector machines for spam categorization. IEEE Transactions on Neural Networks, Vol. 10, No. 5, Settembre 1999.

[22] J. Del Campo-Avila e R. Morales-Bueno e Albert Bifet J. M. Carmona, M. Baena-Garcia. Gnusmail: Open framework for on-line email classification. ECAI 2010, 2010.

[23] G. A. Carpenter e S. Grossberg J. Reynolds. Art 2: a self-organizing neural network architecture for fast supervised learning and pattern recognition. Artificial Neural Networks, 1991.

[24] M.A. Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. Journal of the American Statistical Association 84, 1989.

[25] K. Popat T. Hofmann K. Toutanova, F. Chen. Text classification in a hierarchical mixture model for small training sets, 2001.

[26] A. Sperduti L. De Nardo. Un sistema di classificazione di posta elettronica. Tesi di laurea triennale, 2009.

[27] A. Sperduti M. Astegno. Studio di tecniche di apprendimento automatico per l'analisi e la classificazione di messaggi di posta elettronica. Tesi di laurea specialistica, 2011.

[28] K. Kita M. Sasaki. Rule-based text categorization using hierarchical categories. Proc. of the IEEE Int. Conf. on Systems, Man, and Cybernetics, 1998.

[29] L. Massey. On the quality of art1 text clustering. Neural Networks 16, 2003.

[30] A. Sperduti Mattia Tomasoni. System for the automatic classification of e-mail. Tesi di laurea triennale, 2007.

[31] David Heckerman e Eric Horvitz Mehran Sahami, Susan Dumais. A bayesian approach to filtering junk e-mail, 1998.

[32] J. Provost. Naive-bayes vs. rule-learning in classification of email. Technical Report AITR-99-284, University of Texas at Austin, Artificial Intelligence Lab, 1999.

[33] J. Rennie. ifile: An application of machine learning to e-mail filtering. In Proceedings of KDD'2000 Workshop on Text Mining, 2000.

[34] Andrew Mccallum e Gary Huang Ron Bekkerman. Automatic categorization of email into folders: Benchmark experiments on enron and sri corpora, 2004.

[35] C. D. Wunsch S. B. Needleman. A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology, Vol.48, Marzo 1970.

[36] R. Schiaffino A. Kershenbaum S. D'Alessio, K. Murray. The effect of using hierarchical classifiers in text categorization. Proc. of the 6th Int. Conf. Recherche d'Information Assistee par Ordinateur, 2000.

[37] S. Matwin e S. Abu-Hakima S. Kiritchenko. Email classification with temporal features. In M. Klopotek, S. Wierzchon, and K. Trojanowski, editors, Intelligent Information Processing and Web Mining, 2004.

[38] Nakatani Shuyo. Language detection library. `http://www.slideshare.net/shuyo/language-detection-library-for-java`, Marzo 2010.

[39] H. Chen S.T. Dumais. Hierarchical classification of web content. Proc. of the 23rd ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR), 2000.

[40] Blaine Nelson e Karl Chen e Anthony D.Joseph Steve Martin, Anil Sewan. Analyzing behaviorial features for email classification. Berkeley, CA: University of California at Berkeley, 2005.

[41] Konstantin Tretyakov. Machine learning techniques in spam filtering. Institute of Computer Science, University of Tartu, 2004.

[42] L.W. Freriks P.J.M. Cluitmans M.J. van Gils. The adaptive resonance theory network: (clustering-) behaviour in relation with brainstem auditory evoked potential patterns, Novembre 1992.

[43] T.W. Finin Y. Labrou. Yahoo! as an ontology: Using yahoo! categories to describe documents. Proc. of the 8th Int. Conf. on Information Knowledge Management, 1999.