Recommendation system basato su social network

Roberto Da Canal

13 marzo 2012

Indice

Capitolo 1

Introduzione al sistema Milkrain

1.1 Il problema della ricerca di contenuti nel web

Oggigiorno le nuove forme di comunicazione, come internet e telefonia mobile, sono utilizzate abitualmente da una gran quantità di persone e gli utilizzi che esse ne fanno sono tra i più disparati. Per rendersi conto di quanto queste forme di comunicazione siano utilizzate si può leggere alcune informazioni riassuntive che l' ITU ha pubblicato in data 7 agosto 2011 [1]. Di queste citiamo due frasi molto significative :

> At the end of 2010 – while more than 2.6 billion people worldwide still lacked access to toilets or other forms of improved sanitation – there were almost 4 billion mobile cellular subscriptions in the developing world.

e

> Global Internet penetration in 2010 (30%) was higher than global fixed (16%) or mobile (12%) telephone penetration in the year 2000.

Il grande utilizzo delle nuove forme di comunicazione presenta certamente grandi vantaggi per gli utilizzatori finali, ma allo stesso tempo porta numerose sfide per gli sviluppatori che devono cercare di far funzionare al meglio l'enorme potenziale che la tecnologia ha da offrirci. Il lavoro svolto in questa tesi si inserisce proprio in una di queste difficoltà : la ricerca di informazioni rilevanti nel web.

[1] http://www.itu.int/net/pressoffice/stats/2011/03/index.aspx

Questo problema si è presentato già alla nascita del www, nel 1991, quando il campo delle informazioni era ristretto alle sole pubblicazioni scientifiche, e con il passare degli anni, l'aumento dell' utenza e della quantità di contenuti, il problema ha acquisito sempre maggior rilevanza. Numerosissimi sforzi sono stati svolti e molte tecniche e approcci sono stati presentati e utilizzati. Il più famoso di essi è senza dubbio l'algoritmo PageRank che ha portato alla creazione di Google, il più grande motore di ricerca al mondo e il cui fatturato supera i 29 miliardi di dollari. Nonostante i numeri neppure Google riesce a soddisfare appieno i propri utenti : ad esempio, nella scrittura di questa stessa pagina si è utilizzato Google per la ricerca dei dati relativi all'utilizzo di internet e alla query "internet statistics" il motore di ricerca ha suggerito come primo risultato la risorsa "http://www.internetworldstats.com" , ma come si fa a sapere se questa fonte è adatta per scrivere una tesi ? E' sicura e attendibile ? Google sa l' importanza della ricerca per lo scrittore di questa tesi ? La risposta a queste domande non si può sapere e certamente non è facile da individuare : il fatto che Google è il motore di ricerca più utilizzato non vuol dire che esso sia infallibile e difatti si è optato per una fonte ritenuta più autorevole. Tra tutti i concetti e le metodologie proposte nella letteratura vengono riportate nelle sezioni seguenti quelle più significative per il progetto Milkrain.

1.2 Recommendation System

Un Recommendation System (RS) è un sistema che propone all'utente delle risorse, o più in generale *item*, cercando di massimizzare il grado di interesse dell'utente negli item proposti. Ad esempio molti siti web di e-commerce (es ebay, amazon) propongono automaticamente prodotti aggiuntivi che l'utente non ha ricercato ma che potrebbero interessarlo; similarmente, siti di condivisione di contenuti, come Youtube o Last.fm, suggeriscono ulteriori contenuti non ricercati direttamente dall'utente.

Un RS può utilizzare diversi approcci per migliorare la sua efficacia, i due più affermati sono :

- *Content Based* : basano la loro strategia di ricerca suggerendo item simili a quelli che l'utente ha precedentemente giudicato interessanti ;
- *Social Based* : basano la loro strategia di ricerca sull' esperienza degli altri utenti simili all'utente attuale.

Nelle definizioni precedenti sono presenti due fattori importanti : la similitudine tra utenti e la similitudine tra item. Ogni RS può utilizzare la tecnica che preferisce per implementare una misura di similitudine, ma si deve osservare che questo compito comporta sempre, da parte del RS, l'acquisizione di dati da parte del RS che riguardano l'utente e la storia

dell'utente nell' utilizzo del sistema. Sempre mantenendo molto generale la descrizione dei RS si è osservato che le due tipologie presentano le seguenti caratteristiche :

- I sistemi Content-based hanno il vantaggio che un item può essere presentato a un utente anche se non è mai stato visionato da altri utenti, metre lo svantaggio principale si ha con gli item che non sono simili agli item giudicati positivamente dall'utente ma che lo potrebbero interessare ugualmente. Inoltre questi sistemi presentano un periodo di avviamento in cui non hanno a disposizione dati sull'utente.

- I sistemi Social-based hanno caratteristiche quasi opposte ai precedenti. Un item viene presentato a un utente solo quando altri utenti lo hanno valutato positivamente e possono essere suggeriti anche item inaspettati ma comunque interessanti. Inoltre un nuovo utente fin da subito può sfruttare il lavoro svolto dagli altri utenti senza dover aspettare un periodo iniziale. Lo svantaggio sta nel fatto che l'utente deve valutare numerosi item prima che il sistema diventi efficace.

Nell' ultimo periodo si stanno sviluppando anche i Semantic Recommender System che prevedono l'utilizzo di particolari tecniche per dare un significato ai dati grezzi del web. Solitamente queste tecniche consistono nell'utilizzo di tag, ovvero di parole chiave che specificano il significato del dato a cui si riferiscono e che rendono accessibile questa informazione anche ai software e non solo all'uomo. I tag sono inseriti dagli utenti stessi, che sono in grado di capire il significato esatto della risorsa, agevolando quindi il lavoro dei RS. L'utilizzo dei tag però pone anche delle difficoltà, come ad esempio gli errori di tipografia nella specifica di un tag, la limitatezza dei tag utilizzabili e la soggettività con cui i tag vengono utilizzati.

Infine un concetto molto importante nello sviluppo dei RS è la fiducia o *trust*. Essa ricopre un ruolo fondamentale in diversi RS e può essere utilizzata per migliorare l'efficacia del sistema. L' utilizzo principale della fiducia nei RS è quello di analizzare le correlazioni tra vari utenti : se l'utente X ha un alto grado di fiducia rispetto all'utente Y allora gli item ritenuti positivi da Y dovrebbero essere interessanti anche per l'utente X. Come vedremo anche il sistema Milkrain utilizza il concetto di fiducia, ma non solo tra utenti.

1.3 Milkrain

Il progetto Milkrain vuole essere un nuovo RS per contenuti web e di tipo social-based. L'idea principale sviluppata nel progetto è quella di migliorare le ricerche web effettuate mediante i motori di ricerca usuali grazie alla cooperazione degli utenti e all'analisi dei loro feedback. Il progetto è stato

avviato con il lavoro svolto dall'Ing. Stefano Benvenuti in [3]. Nel seguito viene riportato una sintesi del funzionamento del sistema sviluppato da Benvenuti rimandando al lavoro originale per i dettagli.

1.3.1 Utilizzo e moduli principali

Come detto un punto di forza di Milkrain è quella di sfruttare la collaborazione degli utenti. A questo proposito gli utenti sono organizzati in un grafo che modella i legami di fiducia esistenti tra di essi. In particolare la fiducia di un utente X nei confronti di un utente Y viene specificata non da un unico valore, ma bensì da un insieme di valori : uno per ogni possibile categoria, o *topic*, prevista dal sistema.

Nel normale utilizzo del sistema si hanno in sequenza le seguenti fasi :

1. l' utente sottopone una query al sistema ;
2. il sistema recupera una serie di risorse da datasource esterni e una lista di topics da associare alla query dell'utente ;
3. le risorse recuperate vengono ordinate in base alle valutazioni espresse precedentemente da altri utenti ;
4. le risorse riordinate vengono presentate all'utente il quale può dare un giudizio a zero, una, o più risorse mediante un sistema a 5 valori di giudizio ;
5. in caso di valutazioni il sistema esegue un aggiornamento del database .

L'applicazione è stata sviluppata in modo modulare e le varie fasi di funzionamento sono di competenza dei rispettivi moduli. Non si riporta la descrizione esaustiva di tutti i moduli, perchè disponibile in [3], ma si riporta comunque sotto forma di pseudocodice solamente le fasi più importanti del funzionamento del sistema originale.

Modulo principale

Algorithm 1 Pseudocodice del modulo principale $Milkrain$

Require: $user$, $query$
  Sia LR una lista vuota di risorse e LT una lista vuota di topic
  LR,$LT \leftarrow$ Classifier($query$)
  return Combine($user$,LR,LT)

Questo modulo delinea la risposta del sistema a una query richiamando gli altri moduli.

Modulo Classifier

Algorithm 2 Pseudocodice del modulo $Classifier$

Require: *query*
  $resources$, $labels$ $\leftarrow$ interroga data source
  $dynTopics \leftarrow \emptyset$
  for all *label* in $resources.labels$ **do**
    $tmpTopics \leftarrow$ StaticClassifier($label$)
    $dynTopics \leftarrow dynTopics \cup tmpTopics$
    for all *topic* in $tmpTopics$ **do**
      $dynTopics(topic) += 1$ /*Salva un contatore per ogni topic*/
    end for
  end for
  $stcTopics \leftarrow$ StaticClassifier($queryfiltrata$)
  $allTopics \leftarrow \emptyset$
  if $\emptyset \not\subseteq stsTopics \subseteq dynTopics$ **then**
    $allTopics \leftarrow stsTopics$
  else if $dynTopics \neq \emptyset$ **then**
    $allTopics \leftarrow \{topic$ in $dynTopics$ con contatore massimale $\}$
    if $\emptyset \neq stcTopics$ **then**
      $allTopics \leftarrow stcTopics$
    else
      Se $queryfiltrata$ ha una lunghezza inferiore a 4
      salva nel database la corrispondeza
      tra *query* e i topic in $dynTopics$
    end if
  else
    $frbTopics \leftarrow$ AskFreebase($query$)
    $allTopics \leftarrow allTopics \cup frbTopics$
    Se $queryfiltrata$ ha una lunghezza inferiore a 4
    salva nel database la corrispondeza
    tra *query* e i topic in $frbTopics$
  end if
  return $resources$ **and** $\{t \in allTopics : t$ è un topic permesso$\}$

Il Classifier svolge due importanti compiti : recuperare le risorse dai data source e estrapolare i topic adatti. Nell' implementazione originale il Classifier è composto da due sotto-moduli : lo Static-Classifier e il Dynamic-Classifier. Il primo analizza una query estrapolando le singole parole, le coppie di parole consecutive e le triplette di parole consecutive e controllandone la presenza nel database. Il secondo invece si basa sull'algoritmo *Lingo*, che effettua un'analisi delle risorse basandosi sul modello Vector Space Model e utilizzando la decomposizione ai valori singolari per ottenere un model-

lo dei concetti espressi nelle risorse in forma matriciale. Poi, considerando le parole, le coppie e le triplette di parole consecutive nella query come pseudo-risorse trova quelle più significative confrontandole con le colonne della matrice precedentemente calcolata.

Modulo Combine

Algorithm 3 Pseudocodice del modulo *Combine*

Require: $user$, LR, LT
  elimina in LR le risorse che compaiono più volte
  $\alpha \leftarrow \sum_{t \in LT}$ getAlphaFromDB($user$, $topic$) / $|topics|$
  $\beta \leftarrow \sum_{t \in LT}$ getBetaFromDB($user$, $topic$) / $|topics|$
  for all $resource \in LR$ **do**
    $Ratings \leftarrow \emptyset$
    for all $topic \in LT$ **do**
      $resource.topics \leftarrow resource.topics \cup topic$
      $neighbours \leftarrow$ getNeighboursFromDB($user$, $topic$)
      for all $neighbour \in neighbours$ che ha votato $resource$ **do**
        $rating \leftarrow rating \cup$ getRatingFromDB($neighbour$, $resource$, $topic$)
        $trust \leftarrow$ getTrustFromDB($user$, $neighbour$, $topic$)
        $ratings \leftarrow ratings \cup (rating, trust)$
      end for
    end for
    if $|ratings| > 0$ **then**
      $resource.rating \leftarrow$ mediaPesata($ratings$)
      $resource.ranking \leftarrow \alpha * resource.initialRanking + \beta * resource.rating$
    end if
  end for
  return LR ordinata secondo i valori di ranking delle risorse

Come appare chiaro dal listato 3 il Combine ha l' importate compito di ordinare le risorse tenedo conto di due fattori principali : la posizione originale nei motori di ricerca e la valutazione delle risorse da parte di altri utenti. A questi due fattori vengono affiancati anche i valori di fiducia che l'utente ha rispetto ai motori di ricerca (parametro α) e rispetto alla rete di utenti (parametro β). Anche i valori di fiducia dell'utente nei confronti dei motori di ricerca e nei confronti della rete di utenti sono specifici per ogni topic.

Modulo Feedback

Attualmente il feedback di una risorsa consiste in una valutazione numerica intera compresa tra 1 e 5. Il modulo traduce la valutazione in una

posizione di classifica, ad esempio alla valutazione 5 associa la prima posizione, e confronta questa posizione con i valori *resource.initialRanking* e *resource.rating* : se la posizione è più vicina a *resource.initialRanking* allora α viene incrementato e β decrementato, altrimenti si ha l'aggiornamento opposto. Successivamente e con una tecnica simile vengono aggiornati i valori di fiducia dell'utente rispetto agli amici che hanno valutato la risorsa.

Interfaccia web

Per quanto riguarda l'interfaccia web essa è stata realizzata mediante il linguaggio Php, a differenza di tutti gli altri moduli che erano realizzanti in Java.

1.3.2 Lavoro svolto durante la tesi

Introduciamo il lavoro svolto durante la tesi mediante due tabelle (1.1 1.2) : la prima espone l'organizzazione del progetto sviluppato in [3] mentre la seconda espone il progetto attuale.

<table>
<tr><td>Classifier</td><td>Combine</td><td>Feedback</td><td>Interfaccia Web</td></tr>
<tr><td colspan="3">JVM</td><td>Apache2</td></tr>
<tr><td colspan="4">OS</td></tr>
</table>

Tabella 1.1: Struttura progetto Milkrain originale

Il progetto originale si compone effettivamente di due parti, il core che comprende i moduli Combine, Classifier e Feedback interamente scritta in Java e una seconda parte scritta in Php che serve da interfaccia per gli utenti. Le due parti sono slegate, nel senso che quando l'utente richiede una particolare funzionalità del sistema (come ad es. la ricerca di una query, o l'aggiornamento del proprio profilo) il server web Apache2 richiama un' apposita applicazione Java, facente parte dell'intero core, che si occupa di portare a termine il compito richiesto.

Il lavoro di tesi svolto ha modificato il progetto che ora assume l'organizzazione riportata nella tabella 1.2. La prima modifica apportata consiste nella sostituzione della vecchia interfaccia web con una nuova interfaccia realizzata in Scala mediante il framework Lift. L'adozione del framework è giustificata dal fatto che questi strumenti sono sviluppati appositamente per facilitare e velocizzare lo sviluppo di siti web, inoltre Scala garantisce una miglior interazione con il core di Milkrain, come si vedrà da alcuni test effettuati. Il secondo passo è stato quello di adottare il sistema di virtualizzazione VirtualBox al fine di rendere l'intero progetto facilmente spostabile da una macchina fisica ad un'altra. Infine è stato costruito un nuovo modulo classifier basato su support vector machine per rispondere alle difficoltà

di classificazione di query composte da numerose parole. Il nuovo classifier risolve questo problema e raggiunge una precisione di classificazione migliore.

Interfaccia Web LIFT		
Classifier	Combine	Feedback
JVM		
OS Guest		
Virtual Box		
OS		

Tabella 1.2: Struttura progetto Milkrain attuale

Capitolo 2

Server Milkrain

In questo capitolo vengono presentati gli strumenti software utilizzati sia per facilitare lo sviluppo del progetto, sia dal progetto stesso che sono stati introdotti durante il lavoro di questa tesi. E' importante sottolineare che Milkrain utilizza Cassandra come sistema database, ma non si riportano informazioni a riguardo in quanto l'utilizzo del database è rimasto invariato dalla tesi di Benvanuti.

2.1 VirtualBox

VirtualBox è un software di virtualizzazione con licenza freeware, da noi utilizzato. Grazie ad esso è possibile costruire una macchina virtuale, detta anche macchina guest in contrapposizione alla macchina reale detta macchina host, su cui poter lavorare ottenendo alcuni benefici :

1. se necessario effettuare un backup è possibile copiare l'intera macchina virtuale, rendendo molto semplice la ripresa del lavoro in caso di guasti anche gravi ;

2. se si vuole cambiare la macchina fisica, perchè ad esempio si necessita di un hardware migliore, basta copiare la macchina virtuale evitando la reinstallazione di tutti gli strumenti utilizzati da Milkrain.

In particolare il secondo punto troverà applicazione quando il progetto verrà aperto al pubblico : in questa occasione Milkrain verrà ospitato in un server molto probabilmente virtuale messo a disposizione da una azienda di hosting.

Nella seguente sezione viene riportata la procedura per creare una macchina virtuale con sistema operativo Ubuntu Server su una macchina fisica anch'essa con Ubuntu Server.

Crezione della macchina virtuale

Per la creazione della macchina virtuale sono stati eseguiti i seguenti comandi :

- registrazione nuova macchina virtuale con sistema operativo guest Ubuntu :
  `VBoxManage createvm --name VMMilkrain --basefolder /home/milkrain/vm/VMMilkrain --ostype Ubuntu --register`
- impostazione memoria ram, lettore dvd, interfaccia di rete :
  `VBoxManage modifyvm VMMilkrain --memory 1024 --acpi on --boot1 dvd --nic1 bridged --bridgeadapter1 eth2 --macaddress1 ############`
- creazione hard disk virtuale :
  `VBoxManage createhd --filename /home/milkrain/vm/VMMilkrain/VMMilkrain.vdi --size 20000`
- attacco hard disk virtuale a macchina virtuale :
  `VBoxManage storagectl VMMilkrain --name IDE Controller --add ide`
- `VBoxManage storageattach VMMilkrain --storagectl IDE Controller --port 0 --device 0 --type hdd --medium /home/milkrain/vm/VMMilkrain/VMMilkrain.vdi`
- monto l'immagine di installazione del so guest nel lettore dvd della macchina virtuale :
  `VBoxManage storageattach VMMilkrain --storagectl IDE Controller --port 0 --device 1 --type dvddrive --medium /home/milkrain/vm/ubuntu-11.10-server-i386.iso`
- avvio macchina virtuale :
  `VBoxHeadless --startvm VMMilkrain &`

Dopo aver eseguito questi comandi si accede alla macchina virtuale tramite il programma `rdesktop`, anche da remoto, e si prosegue con l'installazione del sistema operativo sulla macchina virtuale. Conclusa tale operazione è possibile spegnere la macchina virtuale, rimuovere il lettore dvd e riavviare la macchina virtuale :

- spegnimento macchina virtuale :
  `VBoxManage controlvm VMMilkrain poweroff`
- rimozione lettore dvd :
  `VBoxManage storageattach VMMilkrain --storagectl IDE Controller --port 0 --device 1 --medium none`

- riavvio macchina virtuale :
  `VBoxHeadless --startvm VMMilkrain &`

Di particolare importanza è la creazione dell'interfaccia di rete per la macchina virtuale specificata nella seconda linea di comando tramite le opzioni `-nic1 bridged --bridgeadapter1 eth2 --macaddress1 ############` . In questo caso si è adottata la modalità "bridged" la quale fa apparire la macchina virtuale nella rete della macchina reale come se effettivamente fosse collegata a tale rete, utilizzando l'interfaccia `eth2` della macchina host e utilizzando il mac address specificato. Questa modalità è molto pratica, in quanto la macchina virtuale appare come una macchina fisica collegata alla rete e quindi accessibile mediante software di accesso remoto, come ssh. Inoltre se nella rete è presente un server dhcp, la macchina virtuale acquisterà da esso un indirizzo ip in modo automatico.

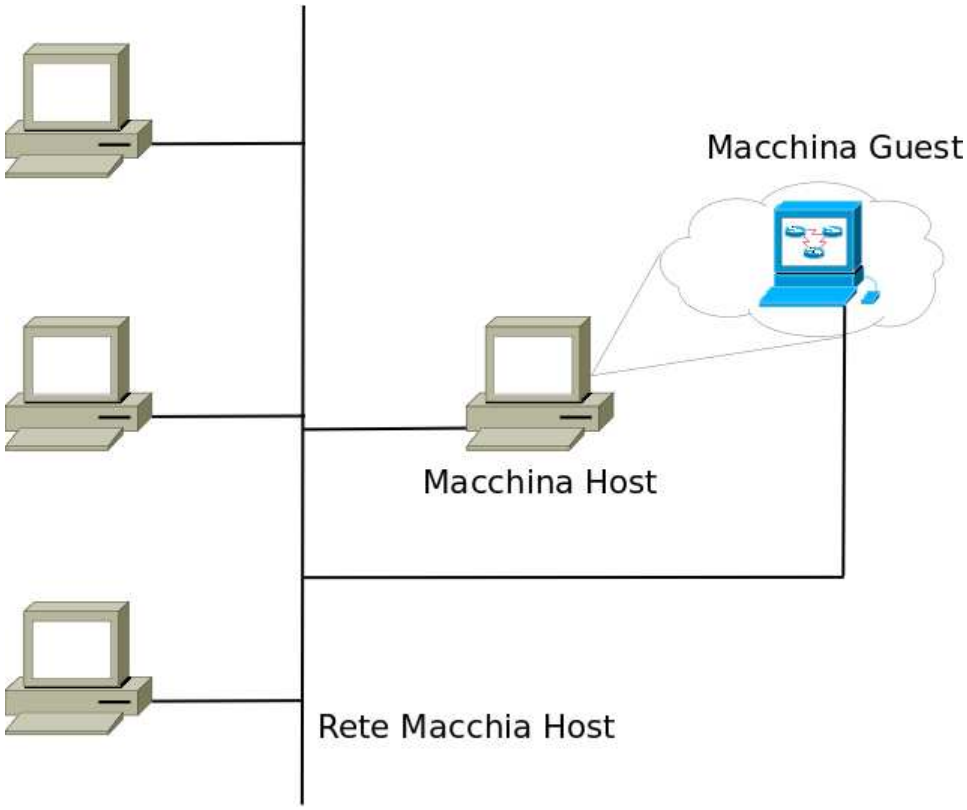


Figura 2.1: Macchina virtuale collegata in modalità `bridged`.

2.2 Jetty e Apache

L'adozione del framework Lift, discussa più avanti nella tesi, ha implicato l'utilizzo di due ulteriori strumenti : un web server che sia anche un servlet container e l'utilizzo di Maven come build manager dell'intero progetto.

Jetty e SSL/TLS

Jetty è il web server utilizzato da Milkrain, anch'esso disponibile liberamente come VirtualBox. Non sono state necessarie particolari procedure per il suo utilizzo e l'installazione è stata particolarmente semplice. L'unico passo che merita di essere riportato è l'adozione dell'accesso di tipo https alle pagine web da parte degli utenti. Questo metodo di accesso prevede l'utilizzo di certificati e protocolli di crittografia durante la fruizione delle pagine web in modo da ottenere almeno i seguenti vantaggi :

- viene assicurata l'identità del server : in questo modo l'utente è sicuro di accedere a pagine web di Milkrain e non a pagine web fasulle che si fingono il servizio richiesto ;
- lo scambio dei dati tra utente e server viene crittografato in modo che non sia interpretabile da eventuali intercettatori.

In questo modo si vuole fornire un servizio che sia sicuro e che garantisca la privacy dell'utente. In particolare il primo punto si basa sull'emissione di certificati firmati da Certification Autorities (CA) riconosciuti a livello globale o da CA appartenenti a una catena di CA con in testa un CA riconosciuto. Per MilkRain, invece, si utilizzerà un certificato autofirmato, senza quindi richiedere l'intervento di una CA esterna, rinunciando in parte al primo servizio offerto da SSL/TLS : nella sostanza dei fatti, quando un utente si connetterà per la prima volta al sito web il suo browser richiederà l'aggiunta di una eccezione di sicurezza. Per poter accedere a questi vantaggi, e quindi mettere a disposizione l'accesso https, si è scelto di utilizzare OpenSSL, ovvero una libreria (anch'essa libera) che implementa i protocolli SSL e TLS. Tali protocolli si interpongono tra il protocollo TCP e il protocollo HTTP e creano un canale di comunicazione criptato tra il client e il server. L'adozione di OpenSSL prevede i seguenti passi :

- creazione di un coppia di chiavi RSA ;
- creazione di un certificato ;
- configurazione del server Jetty.

Si riportano i vari passi [1] :

[1] http://wiki.eclipse.org/Jetty/Howto/Configure_SSL

- creazione di un file casuale :
  `dd if=/dev/urandom of=casuale bs=1M count=1` ;
- creazione chiavi RSA :
  `openssl genrsa -des3 -rand casuale -out jetty.key` ;
- creazione certificato :
  `openssl req -new -x509 -key jetty.key -out jetty.crt` ;
- unione certificato e chiavi in un file PKCS12 :
  ```
  openssl pkcs12 -inkey jetty.key -in jetty.crt -export -out
  jetty.pkcs12
  keytool -importkeystore -srckeystore jetty.pkcs12 -srcstoretype
  PKCS12 -destkeystore jettykeystore ;
  ```
- opzionalmente offuscare le password utilizzate nei punti precedenti :
  ```
  java -cp lib/jetty-http-8.0.0.M3.jar:lib/jetty-util-8.0.0.M3.jar
  org.eclipse.jetty.http.security.Password passworddaoffuscare
  ;
  ```
- modifica file configurazione Jetty aggiungendo :
  ```
  <Call name=‘‘addConnector’’>
  <Arg>
  <New class=‘‘org.eclipse.jetty.server.ssl.SslSelectChannelConnector’’>
  <Arg>
  <New class=‘‘org.eclipse.jetty.http.ssl.SslContextFactory’’>
  <Set name=‘‘keyStore’’>/home/.../ssl/jettykeystore</Set>
  <Set name=‘‘keyStorePassword’’>jettykeystorepassword</Set>
  <Set name=‘‘keyManagerPassword’’>jettypkcs12password</Set>
  <Set name=‘‘trustStore’’>/home/.../ssl/jettykeystore</Set>
  <Set name=‘‘trustStorePassword’’>jettykeystorepassword</Set>
  </New>
  </Arg>
  <Set name=‘‘port’’>8443</Set>
  <Set name=‘‘maxIdleTime’’>30000</Set>
  </New>
  </Arg>
  </Call> ;
  ```

Jetty e Milkrain

Prima di avviare il server Jetty è necessario compilare il progetto e inserire la servlet ottenuta nell'apposita directory prevista dal server web. Come già anticipato il progetto Milkrain, adottando il framework Lift (di cui si parlerà nel prossimo capitolo) viene sviluppato e gestito dal build-manager Maven il quale ha il compito di gestire tutte le dipendenze a librerie esterne

presenti nel progetto. Anche l'installazione di Maven non presenta difficoltà (in quanto presente nei repository Ubuntu). Tutte le dipendenze sono inserite modificando il file `pom.xml` del progetto Milkrain : al momento della compilazione sarà compito di Maven ricercare nei suoi repository remoti le dipendenze specificate e renderle disponibili al compilatore `javac`, senza dover cercare e scaricare tutte le librerie a mano. A onor del vero, alcune librerie devono essere aggiunte localmente, in un repository locale, perchè non ancora disponibili nei repository remoti di Maven, ma sono comunque un numero esiguo.

I comandi Maven principalmente utilizzati sono i seguenti :

- compilazione progetto :
  `mvn compile` ;
- avvio server jetty :
  `mvn jetty:run -Djetty.port=####` ;
- creazione servlet :
  `mvn package` .

Si noti che il secondo comando è utile in fase di sviluppo, perchè è un modo veloce per vedere il lavoro fatto, ma è di gran lunga preferibile utilizzare l'ultimo comando al fine di avviare il server, in quanto in questo modo si può modificare il file di configurazione. Per inserire la servlet ottenuta è sufficente copiare il relativo file `milkrain-1.0.war` nella cartella di jetty `webapps` e creare il file `/contexts/milkrain-1.0.xml` (sempre nelle sottocartelle di Jetty) con il seguente contenuto :

```
<?xml version=‘‘1.0’’ encoding=‘‘ISO-8859-1’’?>
<!DOCTYPE Configure PUBLIC ‘‘-//Mort Bay Consulting//DTD Configure//EN’’
‘‘http://jetty.mortbay.org/configure.dtd’’>
<Configure class=‘‘org.eclipse.jetty.webapp.WebAppContext’’>
<Set name=‘‘contextPath’’>/</Set>
<Set name=‘‘war’’> <SystemProperty name=‘‘jetty.home’’ default=‘‘.’’/>
/webapps/milkrain-1.0.war </Set>
</Configure> .
```

Per avviare il server Jetty è sufficiente il comando `java -jar start.jar -Djetty.port=####`.

Apache

Anche se abbiamo detto che il framework utilizzato ha reso necessario l'utilizzo di un servlet container (Jetty nel nostro caso) si è reso necessario l'utilizzo di un ulteriore web server : il più conosciuto ed utilizzato Apache2 [2] . Infatti lo sviluppo di MilkRain, oltre ai contenuti riportati in questa tesi,

[2] http://httpd.apache.org/

ha richiesto l'accesso ai dati presenti in vari social network, come Facebook e Twitter, e tale accesso è stato effettuato tramite pagine php, elaborate per l'appunto dal server Apache. Ovviamente l'accesso di tipo https è stato esteso anche agli accessi alle pagine emanate dal server Apache.

2.3 Controllo versione e Wiki

Al fine di rendere lo sviluppo del progetto un lavoro su cui possano interagire più persone sono stati adottati due strumenti : Subversion come software di controllo versione e MediaWiki come motore wiki. Il primo rende possibile il lavoro di gruppo sullo stesso progetto mantenendo diverse versioni di esso e segnalando in modo opportuno gli utenti in caso di modifiche contemporanee sugli stessi file. Per fare questo il progetto viene gestito in un repository apposito presente nel server al quale i vari utenti possono attingere per avere un copia sempre aggiornata del progetto e per poter salvare le modifiche che essi vogliono apportare.

Il secondo, invece, permette la creazione di un sito web in stile wikipedia dove ognuno può inserire nuove pagine su Milkrain per annotare informazioni, osservazioni e dati che possono ritornare utili in futuro (come ad esempio i passi necessari alla prima installazione del progetto).

Capitolo 3

Implementazione Front End

In questo capitolo si riportano le fasi e le soluzioni adottate nella costruzione del front end del progetto Milkrain, ovverosia di un sito web utilizzabile da un pubblico non esperto per la fruizione dei servizi offerti dal sistema.

3.1 Scelta tecnologie implementative

Come primo passo dell'implementazione si è dovuto scegliere quale linguaggio utilizzare per la costruzione del sito web. La scelta non è affatto semplice in quanto esistono varie possibilità molto utilizzate e affermate. Ad esempio, per tutti i linguaggi web più noti è possibile trovare almeno un sito di successo che lo utilizza, come Microsoft utilizza la tecnologia .NET, Ruby viene utilizzato da Twitter, PHP da Facebook e Java da una serie di compagnie finanziarie.

Per cercare di indirizzare la ricerca di un linguaggio a noi opportuno si è osservato che il core di Milkrain è sviluppato in Java e quindi si potrebbe pensare di previlegiare questo linguaggio rispetto agli altri. Una caratteristica fondamentale che distingue Java nello sviluppo web è il fatto che esso è *s*tatically typed mentre molti altri linguaggi web (linguaggi dinamici) sono *d*ynamically typed. Questo fatto ha alcune ripercussioni tra le quali :

- Java dovrebbe offrire prestazioni, sopratutto in termini di velocità di esecuzione, migliori rispetto agli altri linguaggi [1] ;
- i linguaggi dinamici sono piú semplici da imparare e rendono piú veloce lo sviluppo di applicazioni web.

Il secondo passo, non strettamente necessario ma che noi abbiamo intrapreso, consiste nella scelta e utilizzo di un framework web, ovverosia di

[1] In internet si possono trovare diverse comparazioni tra le performance dei vari linguaggi di programmazione, ad esempio : http://shootout.alioth.debian.org/u32/which-programming-languages-are-fastest.php

una serie di strumenti che facilitano e velocizzano lo sviluppo di un sito web. Se la scelta del linguaggio si è rivelata la prima difficoltà, la scelta del framework è altrettanto ardua. Infatti, per ogni tecnologia esistono una infinità di framework basati su di essa e non esiste un metodo efficace di comparazione tra i vari framework.

In conclusione abbiamo adottato il linguaggio Scala [2] e il framework Lift [3], l'unico web framework disponibile per questo giovane linguaggio. La scelta di Scala rispetto a Java non è stata semplice, in quanto non si avevano esperienze dirette dell'utilizzo delle rispettive tecnologie, ma alla fine è stato scelto Scala per le numerose carattesistiche innovative riportate dai suoi sviluppatori e per la convinzione maturata leggendo pagine sparse nel web che la suite Java Enterprise non sia proprio semplice da utilizzare e che richieda un certo tempo per essere sufficentemente assimilata. Inoltre Scala è un linguaggio Java compliant, ovverosia il compilatore Scala produce programmi che vengono eseguiti dalla Java Virtual Machine al pari di un qualsiasi programma Java. Questo fatto porta al grande vantaggio di poter richiamare classi e metodi Java da programmi Scala (e viceversa), assicurandoci che ritroveremo in esso tutte le potenzialità offerte dal mondo Java.

Nella seguente sezione vengono introdotte le principali caratteristiche del linguaggio.

3.1.1 Scala

Riportiamo due citazioni che vengono utilizzate per descrivere in modo conciso il linguaggio. Esse sono :

> It is a statically typed, mixed-paradigm, JVM language with a succinct, elegant, and flexible syntax, a sophisticated type system, and idioms that promote scalability from small, interpreted scripts to large, sophisticated applications

e

> The name Scala is a contraction of the words scalable language

Vediamo punto per punto la prima definizione.

statically typed Scala, come Java, richiede che il tipo di dato a cui una variabile fa riferimento non possa cambiare durante l'esecuzione del programma : se una variabile fa riferimento ad un oggetto di tipo A, in futuro potrà riferire solo oggetti di tipo A o derivati da A ;

[2] www.scala-lang.org
[3] http://liftweb.net/

mixed-paradigm Scala offre l'interconnessione di due paradigmi di programmazione : *Object Oriented Programming* (OOP) e *Functional Programming* (FP). Sebbene negli ultimi tempi l'approccio OOP è stato di gran lunga più utilizzato non ci sono motivi per non lasciare al programmatore la scelta di quale paradigma utilizzare. Inoltre Scala offre un modo elegante di utilizzare contemporaneamente i due paradigmi per sfruttare il meglio di entrambi.

JVM language Scala è un linguaggio Java-compliant, ovverosia può essere eseguito dalla JVM in quanto il compilatore Scala produce Byte Code. Questo fatto comporta che un programma Scala può utilizzare una classe Java (quindi Scala può attingere da tutte la Api di Java) e viceversa. Inoltre è in fase di sviluppo un compilatore Scala per la piattaforma .NET .

Succint, elegant and flexible synatx Scala offre numerose tecniche per rendere il proprio codice succinto e meno verboso rispetto a Java, rendendo il linguaggio sotto questo punto di vista paragonabile ai linguaggi dinamici. Una delle tecniche più importanti è la *Type Inference* che rende le definizioni di metodi e variabili molto concise. Un'altra tecnica consiste nel permettere caratteri non alfa numerici nei nomi dei metodi : in tal modo, grazie anche a del zucchero sintattico, è possibile avere dei metodi che vengono utilizzati come se fossero degli operatori.

sophisticated type system Scala estende e migliora i tipi di dato disponibili in Java : un esempio sono le *Trait* che possono essere pensate come interfacce Java in cui è possibile inserire del codice [4].

Infine è doverosa una nota sulle origini del linguaggio. Il suo sviluppo è cominciato nel 2001 per opera di Martin Odersky durante il suo dottorato (seguito dal professor Niklaus Wirth padre dei linguaggi Pascal, Oberon e Modula-2 per i quali vinse il premio Turing), il quale ha collaborato anche allo sviluppo del compilatore Java e dei *Generics* sempre di Java.

3.1.2 Lift

Anche se la scelta del framework è stata di fatto implicata dalla scelta del linguaggio Scala, riportiamo i punti fondamentali per quanto riguarda il suo utilizzo.

Il primo vantaggio che lo sviluppatore si accorge nell'utilizzare Lift è la separazione netta fra la logica dell'applicazione e la sua visualizzazione.

[4] Se volessi definire la trait "Confrontabile" con i metodi " < " , " = " , " ! " , " > " , " <= " , " >= " potrei definire direttamente nella trait tutti i metodi in funzione dei soli " < " , " = "

Questo aspetto viene implementato nel framework attraverso l'utilizzo di speciali tag xml inseriti nelle pagine web : in questo modo lo sviluppatore crea la struttura di una pagina web utilizzando l'usale html (o l'xhtml) delineando quindi l'aspetto del sito agli occhi dell'utenza, ma specifica la logica del programma in file separati mediante codice Scala. Questi file di codice Scala vengono "inseriti" nelle pagine web proprio grazie agli appositi tag xml.

Il secondo vantaggio di cui si può beneficiare è la disponibilità di numerose funzioni messe a disposizione dal framework per facilitare la creazione di form, senza doversi occupare di gestire la comunicazione dei dati in arrivo, senza gestire le difficoltà di chiamate ajax (nel caso si voglia utilizzare dei form che non necessitano l'aggiornamento dell'intera pagine web) e sopratutto ottenendo form sicuri senza dover ricorrere a tecniche particolari di sicurezza (come il controllo dell'input ottenuto da un utente contro possibili attacchi sql-injection).

A titolo di esempio vengono riportati le parti di codice che riguardano il form di login a Milkrain. Nella pagina html di accesso il form viene inserito mediante il tag xml :

```
<lift:LoginForm.createForm form=‘‘POST’’>
<div> <label style=‘‘float:left;width:60px;margin-left:10px’’> Username
</label> <entry:username /> </div>
<div> <label style=‘‘float:left;width:60px;margin-left:10px’’> Password
</label> <entry:password /> </div>
<div style=‘‘margin-top:20px’’> <entry:submit /> </div>
</lift:LoginForm.createForm>
```

Come si può vedere in questo frammento di codice non si specifica nulla a riguardo delle operazioni che si devono effettuare al login di un utente, ma si specificano solo quali sono le componenti del form (come le labels) e il loro aspetto. Il tag `lift:LoginForm.createForm` richiama il codice scala seguente che crea effettivamente il form e specifica quali siano le operazioni da effettuare a fronte di una richiesta di login :

```
 bind(‘‘entry’’, xhtml, ‘‘username’’ -> SHtml.text( usrName.toString,
incoming => usrName.set(incoming)), ‘‘password’’ -> SHtml.password( usrPassword.toStrin
incoming => usrPassword.set(incoming)), ‘‘submit’’ -> SHtml.submit( ‘‘Login’’,
check))
```

Come è facilmente intuibile, ad esempio, il codice specifica che il tag xml `<entry:username />` deve essere sostituito da un campo di testo che, quando il form viene inviato, richiama la funzione `incoming => usrName.set(incoming)` per aggiornare il valore di una variabile. Grazie a questo piccolo esempio si può vedere come i due vantaggi sopra esposti siano effettivamante raggiunti dall'utilizzo del framework.

3.1.3 Struttura e funzionalità del sito web

Attualmente il sito web supporta le seguenti funzionalità :

- iscrizione utenti
- ricerca risorse web
- raccolta feedback espliciti
- gestione degli amici e dei topics in comune
- analisi dati da social network

3.2 Analisi e stress test

In questa sezione si vuole investigare sulle risorse che il nuovo sistema (server Jetty e framework Lift) necessita ed avere un minimo di confronto rispetto alla precedente implementazione (server Apache e PHP). Durante i vari test si sono utilizzati i seguenti programmi :

- VisualVM : fornisce l' utilizzo CPU e memoria di programmi Java ;
- top : fornisce l'utilizzo di CPU e memoria di applicazioni in generale ;
- JMeter : permette di simulare numerose richieste concorrenti a una pagina web.

3.2.1 Analisi server Jetty

Utilizzando JMeter si sono simulate diverse situazioni di stress modificando di volta in volta il numero di richieste alla pagina iniziale, ma lasciando sempre uguale a 60 secondi l' intervallo di tempo in cui le richieste vengono effettuate. Si riportano in tabella i risultati ottenuti :

Nella tabella e nelle tabelle successive i tempi sono misurati in millisecondi e più precisamente i loro significati sono i seguenti :

- tempo medio : tempo medio di risposta del server ;
- tempo minimo : tempo minimo di risposta del server ;
- tempo massimo : tempo massimo di risposta del server ;
- deviazione standard : deviazione standard dei tempi di risposta ;
- richieste al secondo : numero di pagine/richieste servite al secondo ;
- KB al secondo : traffico generato in uscita dal server verso i client.

N° richieste al secondo	Tempo medio	Tempo minimo	Tempo massimo	Deviazione standard	Richieste al secondo	KB secondo
10	6	3	113	7,64	10	25,25
20	7	3	131	9,93	20	50,37
40	6	3	100	6,24	39,7	100,22
80	68	3	3107	391,96	76,9	194,24
160	145	2	4937	598,57	135,8	342,76
320	6214	3	56977	6965,66	132,4	334,19
240	1829	2	95721	10244,72	85,7	216,32
200	1957	4	26687	2705,02	109,4	276,08

Tabella 3.1: Stress test server Jetty

Le prove consistono nella semplice richiesta della pagina iniziale, in questo modo viene analizzato solo il comportamento del server, tralasciando il core di Milkrain, il database e i vari datasource : questo approccio è utile perché tali componenti sono comunque comuni alle due implementazioni del sito. Dalla tabella si vede che il server regge bene fino a 160 richieste al secondo (9600 al minuto), oltre questa soglia il tempo medio di risposta cresce nettamente passando dall'ordine dei decimi di secondo all'ordine dei secondi. Tutti questi test sono stati effettuati utilizzando solo JMeter e non anche VisualVM per avere dei dati il più veritieri possibile.

Individuata la soglia di traffico massimo se ne è analizzato il comportamento in termini di memoria utilizzata e percentuale di CPU richiesta attraverso VisualVM (oltre a JMeter). Di seguito i dati ottenuti dai due programmi :

N° richieste al secondo	Tempo medio	Tempo minimo	Tempo massimo	Deviazione standard	Richieste al secondo	KB secondo
160	649	4	8611	1419,85	101,4	255,99

Tabella 3.2: Analisi soglia di traffico server Jetty

Da questa immagine le osservazioni interessanti che si possono trarre sono :

- memoria utilizzata inferiore a 500 MB ;
- la memoria utilizzata cresce linearmente a test concluso, solo dopo alcuni secondi si ha un netto abbassamento (non si vede in figura) ;
- la CPU viene utilizzata maggiormente nelle prima metà del test (forse dopo intervengono meccanismi di caching) ;

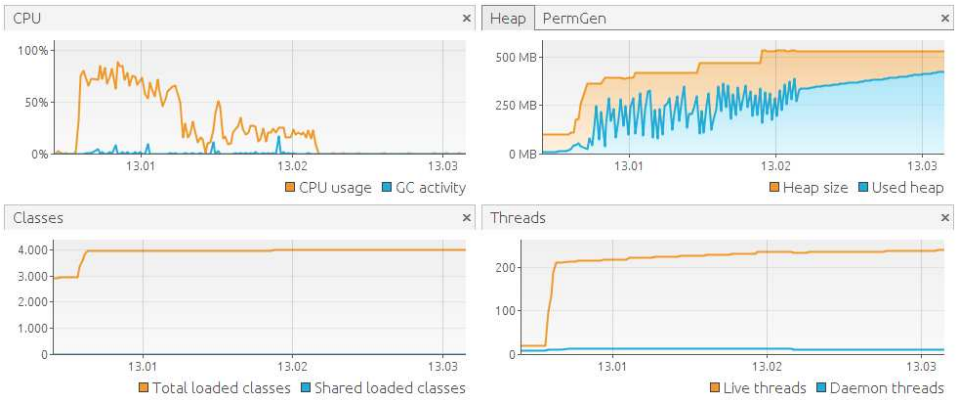


Figura 3.1: Analisi grafica soglia di traffico server Jetty

- I tempi restituiti da JMeter sono peggiori rispetto all'analoga prova effettuata precedentemente, segno che VisualVM ha introdotto dei rallentamenti nel server.

Al fine di analizzare il sito in toto sono state effettuate 10 richieste concorrenti a una pagina che riceve come parametro HTTP GET una query e restituisce le risorse trovate dal core di Milkrain : in realtà questa pagina è stata aggiunta all'applicazione con il solo scopo di rendere più semplice il test in quanto non si deve effettuare il login per accedere ad essa. Si riportano i risultati ottenuti da JMeter (senza VisualVM) e i risultati ottenuti da VisualVM (in una seconda prova):

Tempi ottenuti	5050	4604	2489	4506	3986	4473	4682	4002	4620	4351
Media	4276,3									

Tabella 3.3: Tempi di risposta a 10 query concorrenti server Jetty

Vale la pena ricordare che Milkrain fa un uso abbastanza pesante della rete appoggiandosi a datasource esterni per recuperare risorse a lui utili. Quindi i tempi in tabella non vanno presi come un buon indicatore per le performance del sistema perché essi dipendono da fattori esterni come le condizioni della rete e il carico presente sui datasource. Comunque tali tempi sono per noi importanti per effettuare una comparazione con gli altri sistemi (Apache e Milkrain da solo) in quanto i test sono stati eseguiti lo stesso giorno, sulla stessa macchina e sulla stessa rete.

Alcune osservazioni :

- il carico sulla CPU non è molto elevato (50% circa [5]) ;

[5] La macchina utilizzata ha un processore con quattro core a 2.27 GHz.

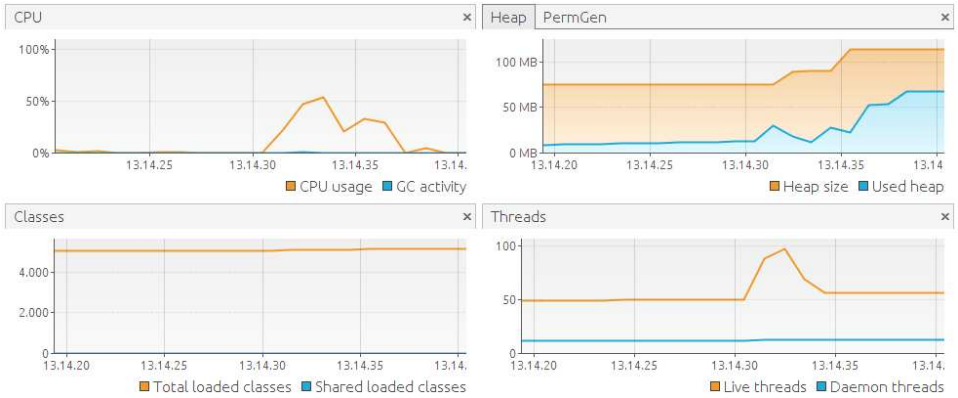


Figura 3.2: Analisi grafica test 10 query concorrenti server Jetty

- l' utilizzo di memoria cresce fino a circa 72 MB ;
- la memoria occupata permane fino a quando non interviene il garbage collector (circa 8-10 sec dopo la fine del test, non si vede in figura) mentre i thread vengono liberati istantaneamente.

3.2.2 Analisi server Apache

Al fine di confrontare i due server (Apache2 e Jetty) è stato rieseguito il test di richiesta della pagina iniziale da parte di numerosi thread, partendo da 9600 thread che era la soglia massima gestita da Jetty. In tabella i risultati ottenuti :

N° richieste al secondo	Tempo medio	Tempo minimo	Tempo massimo	Deviazione standard	Richieste al secondo	KB secondo
160	130	3	3102	546,34	135,2	341,4
320	48	0	4022	371,69	214	365,92
640	29	0	4709	311,48	294,3	501,01

Tabella 3.4: Stress test server Apache

Si osserva immediatamente che Apache (al di là del fatto che la pagina è di 1744 B mentre per sito Lift era di 2572 B) supporta bene carichi di traffico maggiori rispetto alla soglia individuata per Jetty, questo potrebbe derivare dal fatto che comunque Jetty è un programma Java e anche dal fatto che il framework Lift introduce una serie di misure per migliorare la sicurezza e l'affidabilità del sito.

Ora passiamo al secondo tipo di test, ovverosia alle 10 richieste di query contemporanee. I tempi di risposta ottenuti tramite JMeter sono i seguenti :

Tempi ottenuti	12278	13301	12241	12235	12984	13016	12266	12606	13107	12253
Media	12628,7									

Tabella 3.5: Tempi di risposta a 10 query concorrenti server Apache

Questi tempi sono circa 3 volte maggiori rispetto ai tempi ottenuti con Jetty. Inoltre il consumo di CPU e di memoria (ottenuti tramite il comando top) sono risultati praticamente nulli. La motivazione di ciò sta nel fatto che per ogni query ricevuta il sistema PHP/Apache2 effettua una chiamata di sistema che lancia un nuovo processo Milkrain che gestisce la singola query. Molto probabilmente questo fatto è la causa dei tempi di risposta così lunghi. Quindi se da un lato Apache2 risulta essere un server più "snello" rispetto a Jetty, l' adozione di quest'ultimo risulta più efficace nella gestione di query concorrenti e più in generale nell' eseguire funzionalità che necessitano il richiamo di metodi Java.

3.2.3 Analisi applicazione Milkrain

Per completare le comparazioni e effettuate sembra opportuno avere anche un valore della memoria consumata dall'applicazione Milkrain, per ciò si è effettuato il test di 10 query concorrenti anche per il sistema Milkrain. Ecco i risultati ottenuti da VisualVM :

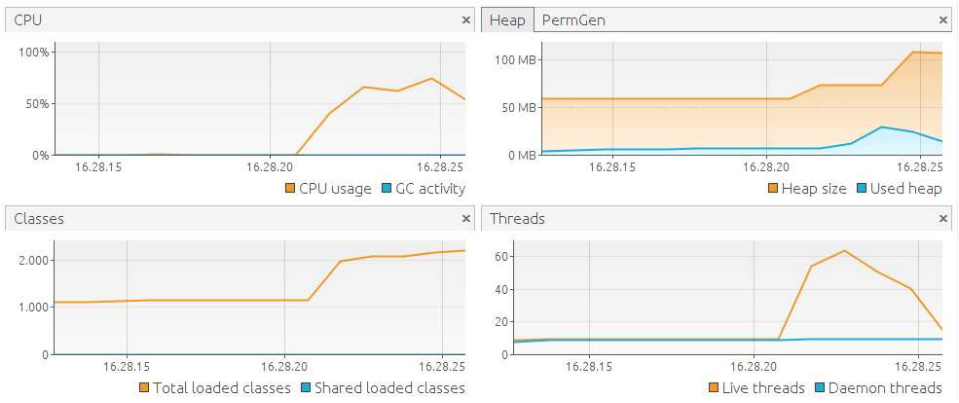


Figura 3.3: Analisi grafica test 10 query concorrenti applicazione Milkrain

Dall'immagine si vede che il picco massimo di memoria è di circa 32 MB, da cui si deduce che il server Jetty (per questo tipo di test) implica un notevole aggravio nella memoria utilizzata.

3.2.4 Tuning applicazioni Java

Dopo aver eseguito i precedenti test si deducono i seguenti fatti :

- a fronte di 10 query il server consuma circa 72 MB di memoria ;
- la percentuale di CPU utilizzata dal Garbage Collector (GC) è praticamente nulla ;
- i tempi di risposta a una query sono nell'ordine dei secondi e sensibili anche alle condizioni della rete.

Da queste osservazioni sembra fattibile l'innalzamento delle attività di garbage collection al fine di abbassare l'utilizzo di memoria massimo da parte del server. Leggendo la documentazione di Jetty risulta chiaro che il server non permette la configurazione di parametri al fine di modificarne le performance, ma l'utente in cerca di tali informazioni viene rimandato a una pagina del sito di Oracle in cui si introducono alcune impostazioni effettuabili su una applicazione Java. Tali impostazioni sono molto numerose e appetibili per un pubblico esperto, per i nostri fini le impostazioni che sembrano più interessanti sono quelle che riguardano i GCs e l'heap dell'applicazione.

Nelle più recenti versioni di Java non esiste un unico GC, ma ne vengono offerti almeno 4 [6]:

- *Serial Collector* (SC)
- *Parallel Collector* (PC)
- *Parallel Compacting Collector* (PCC)
- *Concurrent Mark-Sweep Collector* (CMS)

Attualmente la scelta del GC viene effettuata automaticamente al lancio della JVM ipotizzando che il tipo di applicazione, e quindi il GC che essa necessita, sia legata al tipo di macchina in cui viene eseguita : se, ad esempio, la macchina dispone di 2 o più processori e più di 2 GB di memoria ram la macchina viene classificata come server. Oltre alla scelta del GC, all'avvio dell'applicazione vengono scelti anche altri parametri che riguardano la gestione dell'heap e il GC utilizzato. Attraverso l'impostazione di alcuni flag da riga di comando è possibile modificare le scelte effettuate dal sistema per migliorarne le performance finali.

Attraverso l'uso del flag *-XX:+PrintCommandLineFlags* è possibile vedere le scelte che vengono effettuate in modo automatico al lancio del server. Nel nosrto specifico caso, in cui la macchina dispone di una CPU a 4 core e

[6] alcune note sui GC utilizzati dalla JVM si possono trovare in appendice

4 GB di memoria ram, viene scelto il parallel collector impostandone a 4 il numero di thread, mentre la taglia dell'heap inizialmente vale circa 60 MB e la taglia massima vale circa 955 MB.

Visto che la macchina dispone di 4 core e che la percentuale di CPU utilizzata dal server non è troppo elevata si è deciso di testare il CMS, ma lasciando alla JVM la scelta di ulteriori parametri. Quest'ultimi si sono rivelati essere i seguenti :

- dimensione della young generation compresa tra 20 e 83 MB ;
- dimensione tenured generation iniziale è di 62 MB ;
- dimensione tenured generation 7 volte la dimensione della young generation ;
- un oggetto deve sopravvivere al massimo a 4 young generation prima di essere collocato nella tenured generation ;

Utilizzando il CMS il consumo massimo di memoria a fronte di 10 query contemporanee scende a meno di 30 MB. Inoltre risultano assenti le major collection, ma si hanno solamente delle minor collection e tutte di durata molto breve (centesimi di secondo).

Per investigare ulteriormente il comportamento del server e del CMS si sono effettuati ulteriori test per cercare di capire come cresce la quantità di memoria utilizzata al crescere del numero di query effettuate contemporaneamente. I test consistono nel sottoporre il server a un numero crescente di query e rilevare il consumo di memoria, i dati ottenuti sono riportati nella figura 3.4.

Come si può vedere dal grafico il consumo di memoria cresce in modo lineare, intervallando tratti di non crescita, con il numero di query contemporanee a cui è sottoposto il server.

Sicuramente la scelta dei parametri per questo tipo di configurazioni è un compito molto difficile e che si dovrebbe fare quando si dispongono di dati relativi a una sessione di lavoro vera del server, oltre al fatto che la configurazione ottimale dipende anche dall'hardware che ospita l'applicazione. Infatti bisogna capire che tipo di problemi deve affrontare il server: numerose richieste ma distribuite uniformemente nel tempo o poche richieste concentrate in momenti caldi del giorno, è più problematico il throughput o le altre altre metriche, ecc. E' comunque confortevole vedere che il solo cambiamento del GC comporta un uso minore della memoria e questo fa sperare che un miglioramento aggiuntivo sia possibile attraverso l'impostazione di flag aggiuntivi.

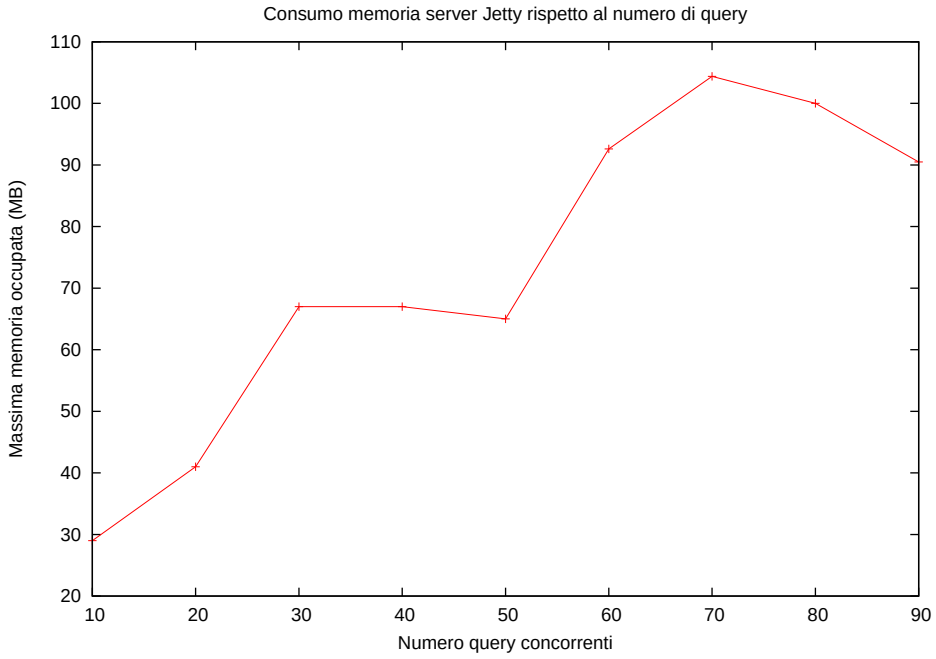


Figura 3.4: Consumo memoria rispetto al numero di query

Capitolo 4

Nuovo Classifier

Come visto nella parte introduttiva al progetto Milkrain il Classifier ha come obbiettivo quello di associare uno o più topic a una query. Questo compito è di vitale importanza perchè solo le risorse classificate beneficiano dei feedback degli utenti : un feedback, infatti, non è associato a un risorsa ma bensì a una coppia (risorsa, topic). Questo comporta che una risorsa può essere valutata positivamente per il topic t_1 ma non per i topic t_2 e t_3. L'importanza del Classifier deriva anche dal fatto che i valori di trust tra utenti sono specificati per ogni topic.

4.1 Classifier precedente

Questo modulo fonda il proprio funzionamento su tre componenti :

- l'utilizzo di un algoritmo di estrazione di parole chiave dai risultati ottenuti dai motori di ricerca ;
- l'appoggio al servizio offerto da Freebase nella classificazione di query ;
- il salvataggio di informazioni utili per la classificazione di query future nel database .

Il primo punto viene implementato grazie a una parte dell'algoritmo Lingo ([13]) : questo algoritmo, originariamente è un algoritmo di clusterizzazione di risultati ottenuti da motori di ricerca, ma per quanto riguarda Milkrain ne è stata utilizzata solo la prima parte che dovrebbe estrapolare le parole / frasi più significative presenti nei risultati. Benvenuti ha chiamato questo algoritmo *Modified Lingo*. Freebase invece è una base di metadati forniti principalmente dai propri utenti con l'obbiettivo di descrivere e contenere tutte le informazioni conosciute, infatti al momento del suo avvio è stato descritto e introdotto con la seguente frase : "an open shared database

of the world's knowledge". Per Milkrain quindi è un mezzo molto potente per la classificazione di una query ed infatti veniva utilizzato come un oracolo nel caso in cui la classificazione della query non poteva essere effettuata utilizzando solo gli altri due punti. Infine quando veniva utilizzato Freebase venivavno salvate nel database alcune associazioni del tipo parola chiave - categoria per cercare di migliorare la classificazione delle query future e soprattutto limitare sempre più la dipenza da Freebase. Da notare che i topic utilizzati dal classifier erano gli stessi utilizzati da Freebase : a rigore Freebase utilizza topic gerarchici mentre Milkrain taglia tali topic ai soli primi due livelli.

Nel corso della tesi ci si è accorti che molte volte il classifier non dava risposte e si ha avuto la sensazione che alcuni topic venivano ritornati molto (troppo) frequentemente rispetto ad altri. Con alcuni semplici esperimenti, che consistevano nel classificare alcune centinaia di query, si è infatti rilevato che le query composte da una o due parole vengono quasi sempre classificate mentre le query con tre o più parole sono sempre meno classificate al crescere del numero delle parole costituenti la query. Già solamente da questo piccolo test che certifica la difficoltà nel classificare query con tre o più parole si è deciso di provare a sviluppare un nuovo classifier.

4.2 Lavori correlati

Ricercando in bibliografia (e in particolare in [5] [6] [7]) si trovano riferimenti a numerosi tentativi di risoluzione al problema e dagli articoli visionati si possono trarre alcune osservazioni di base.

I classificatori proposti possono essere di tipo supervisionato o non supervisionato. I primi dovrebbero riuscire a garantire delle performance migliori ma recuperare dataset adeguati è un compito molto difficile. Contrariamente i classificatori non supervisionati presentano performance lievemente inferiori e non necessitano di dataset di costruzione. Questo però non vuol assolutamente dire che per la costruzione di classificatori non supervisionati non siano necessarie ingenti quantità di dati. Un tipo di classificatori che gioca un ruolo importante nella classificazione di web query sono i classificatori synonym-based che lavorano grazie a delle associazioni parola-query precomputate. Questi classificatori hanno presentano buoni livelli di precisione (si veda la definizione alla fine di questa sezione) ma bassi livelli di recupero. Infine in [6] si studia la possibilità di combinare due classificatori diversi per ottenerne uno ibrido.

Un'altra osservazione comune sta nell' affermare che la singola query non porta con sé informazione sufficiente al fine della classificazione. Per cercare informazione aggiunta si adotta quasi sempre la rete e in particolare i motori di ricerca : considerare i risultati ottenuti da quest'ultimi è praticamente un passo obbligatorio. Oltre ai motori di ricerca tradizionali vengono utiliz-

zati anche i così detti "Directory Search" ovvero dei motori di ricerca che associano ai risultati anche delle categorie (solitamente composte da centinaia di migliaia di topic possibili). Un'altro strumento utilizzato, sopratutto per migliorare i classificatori synonym-based, sono strumenti software per il suggerimento di sinonimi.

Infine la mancanza di data set liberi con cui poter costruire un classificatore supervisionato. L'unico di tali data set disponibile sembra essere quello relativo alla 2005 ACM Knowledge Discovery and Data Mining competition. In questa competizione veniva fornito un data set di 800000 query da categorizzare su 67 categorie ma con solo 111 query già categorizzate da persone. Oggi sono disponibili un totale di 800 query categorizzate manualmente, quelle che sono state utilizzate per valutare i lavori proposti.

In molti casi le valutazioni dei classificatori vengono effettuate mediante le metriche :

1. $A = \sum_i$ # di query classificate correttamente con il topic t_i
2. $B = \sum_i$ # di query classificate con il topic t_i
3. $C = \sum_i$ # di query classificabili con il topic t_i
4. *precisione* $\mathbf{P} = \frac{A}{B}$
5. *richiamo* $\mathbf{R} = \frac{A}{C}$
6. $\mathbf{F1} = \frac{2PR}{P+R}$

Come si può vedere nelle sezioni successive i classificatori attuali non fanno miracoli : già riuscire ad avere valori di precisione pari a 0.5 può essere considerato un buon risultato.

Di seguito vengono sintetizzati i lavori svolti in [5] [6] [7] per dare un'idea di quali sono le risorse e le tecniche utilizzabili.

Unsupervised Query Categorization using Automatically-Built Concept Graphs

In questo articolo ([5]) gli autori evidenziano il fatto che gli approcci di tipo supervisionato hanno il grande difetto di richiedere data set per costruire il classificatore : tali data set dovrebbero essere molto corposi per essere rappresentativi di tutte le query possibili e la costruzione manuale di tali data set richiede sforzi non trascurabili. In più la costruzione di un dataset solitamente copre solo le query di una particolare lingua, quindi se si volesse coprire più nazioni in modo ottimale si dovrebbe costruire un dataset per ognuno di essi. Inoltre gli autori sostengono che i classificatori synonym based sono semplicemente troppo costosi da costruire. Quindi viene proposto

un approccio non supervisionato basato su una semplice metrica :

$$xref(t,t') = \frac{1}{|D_t|} \sum_{d \in Dt} \mathbf{1}_{[t' \in d]}$$

dove t e t' sono dei termini mentre D_t rappresenta l'insieme dei top N documenti restituiti da un motore di ricerca in risposta a t (N è un parametro). Utilizzando log di query a motori di ricerca e recuperando i risultati associati alle query è possibile costruire il context-graph : un grafo in cui un nodo rappresenta un concetto (di solito una parola o frase) e i lati tra i nodi sono pesati dalla metrica $xref$. Il grafo viene reso aciclico eliminando i collegamenti di valore più basso in ogni ciclo e si impone la presenza di un solo lato tra due nodi : se $xref(t,t') > xref(t',t)$ allora si mantiene sono il lato $t \rightarrow t'$. Le categorie (topic) che si vogliono utilizzare sono inserite nel grafo aggiungendo un nodo che le rappresenti e aggiungendo lati di valore 1 che partono dalle parole significative della categoria (descrittori) alla categoria stessa. Una parte dei descrittori vengono trovati analizzando le risorse in D_t, ad esempio estrapolando i concetti chiave utilizzando algoritmi sismili a Modified Lingo, mentre altri vengono trovati dalla procedura di inserimento della categoria all'interno del grafo. La categorizzazione di una query richiede l'estrapolazione di concetti dalle risorse ritornate dal motore di ricerca e la ricerca delle categorie all'interno del concept-graph partendo dai nodi che rappresentano i concetti trovati.

Query Enrichment for Web-query Classification

Il lavoro svolto in [6] è davvero impressionante per quanto riguarda la mole di lavoro svolto. Infatti gli autori non hanno sviluppato un solo classifier ma bensì quattro classifier base e ulteriori classifier ottenuti combinando tali classifier.

I quattro classifier base sono a loro volta suddivisi in due tipologie. Un primo classifier è basato su support vector machine, mentre gli altri sono di tipo synonym based. Per quanto riguarda il classifier basato su svm si sa solamente che è stato costruito a partire dai dati ottenibili dall' Open Directory Project (un sito di tipo Directory Search) che sono liberi e accessibili a tutti e classifica una query classificando le pagine che vengono ritornate dai motori di ricerca come risposta a quella query. I rimanenti tre classifier sono invece di tipo synonym based che utilizzano la classificazione effettuata mediante directory search esterni per ottenere una categoria finale. Il mapping tra le categorie proprie dei directory search e le categorie finali viene effettuato valutando il matching delle singole parole costituenti le catogorie. Per aumentare il numero di categorie su cui poter effettuare questa traduzione sono stati utilizzati anche i plurali e i sinonimi delle categorie in gioco. I tre classifier di tipo synonym based si differenziano esclusivamente sul di-

rectory search utilizzato : Google Directory Search , Looksmart e Lemur (un motore di ricerca costruito a partire dai dati ODP).

Infine vengono proposti ulteriori classificatori ibridi ottenuti mediante combinazioni lineari dei risultati dei classificatori base. La scelta dei pesi da dare ai singoli classificatori base ha portato a quattro classificatori ibridi :

- EDP : i pesi sono specificati per ogni categoria finale in base alla precisione che i classificatori base hanno ottenuto in un validation set ;
- EDF : come EDP ma considerando la misura F1 ;
- EP : viene assegnato un solo peso ad ogni singolo classifier base in proporzione al valore di precisione ottenuto in un validation set ;
- EF : come EP ma considerando la misura F1 ;
- EN : viene assegnato un peso uguale a tutti i classifier.

Nell'articolo vengono comparati i vari classifier commentando i grafici degli andamenti delle misure di precisione, richiamo e F1 dei classificatori base e dei classificatori EDP e EN al variare del numero di pagine web considerate nella classificazione di una query e al variare del numero di catogorie restituite. Da questi esperimenti risulta chiaro come 40 pagine presentino una soglia importante per tutti i classificatori testati, mentre il numero di query ottimale sembra essere 3 ma non in modo così chiaro ed evidente come per il numero di pagine. Si deve comunque notare come i classificatori ibridi proposti in questo articolo richiedano l'utilizzo sia di motori di ricerca sia di directory search.

Automatic Subject Categorization of Query Terms for Filtering Sensitive Queries in Multimedia Search

Anche il lavoro presente in [7] si fonda su una metrica sviluppata appositamente. In particolare siano C l'insieme di categorie e $V = \{w_1, w_2, ..., w_n\}$ un insieme di query pre categorizzate. La categorizzazione di una nuova query t viene effettuata mediante la funzione di ranking :

$$R(t,c) = \sum_{w \in W_{t,c}} \frac{f_w}{max_{k \in W_t} f_k} log(n_w + 1)$$

dove W_t indica i termini di V che sono presenti nelle risorse ritornate da un motore di ricerca in risposta alla query t (insieme indicato con D_t) e $W_{t,c}$ indica i termini corrispondenti alla categoria c di W_t. Infine f_w è il numero di occorrenze del termine w in D_t e n_w indica il numero di documenti in D_t che contengono il termine w.

La categorizzazione di una query mediante questo approccio sembra essere abbastanza semplice, pultroppo il passo molto più impegnativo è la classificazione e la scelta dei termini in V.

Confronto delle prestazioni

Nella tabella 4.1 si riportano i risultati ottenuti nei lavori precedentemente esposti.

Articolo	**Precisione**	**Misura F1**	**Richiamo**	Top-1	Top-5
[5]	0.614	0.460	0.368	-	-
[6]	0.424	0.444	-	-	-
[7]	-	-	-	49.83%	78.64%

Tabella 4.1: Confronto dei risultati

I valori Top-n indicano la percentuale di casi in cui le prime n categorie trovate per una certa query contengono la query adatta. I lavori [5] e [6] sono direttamente confrontabili perchè il validation set utilizzato è quello della KDD Cup Competition. Come si può vedere dai risultati riportati nella tabella la classificazione di query è un problema molto difficile. Infatti sia i valori di precisione che i valori F1 non sono molto elevati, ed inoltre tutti e tre i lavori citati riportano sforzi notevoli nel lavoro effettuato, soprattutto per la creazione di un dataset su cui poter lavorare. Di seguito vengono riportati i dati utilizzati per costruire i modelli di classificazione citati :

- [5] : concept-graph di circa $2,3$ milioni di concetti e circa $6,74$ milioni di lati ;
- [6] : 1540000 richieste a ODP per costruire il data set per il classificatore supervisionato, più tutti gli sforzi fatti per migliorare il classificatore synonym-based. Si ricorda inoltre che la classificazione di una query utilizza sia le risorse provenienti da motori di ricerca sia da directory search ;
- [7] : circa 18000 termini sono stati classificati manualmente.

In tutti e tre i casi gli sforzi compiuti per ottenere dati sufficienti sono davvero notevoli.

Tentativo di miglioramento classifier

Il primo tentativo per migliorare il classifier puntava a riutilizzare i tre punti fondamentali su cui si basava il classifier precedente. Le novità introdotte erano :

1. nel database non venivano più salvate le associazioni query - topic ma venivano salvate le associazioni parola chiave - topic , dove le parole chiave venivano estratte mediante l'algoritmo Modified Lingo ;

2. le parole chiave salvate nel database venivano salvate in forma stemmificata per facilitarne il riuso ;

3. per velocizzare il popolamento del database era stato avviato un programma indipendente che analizzava le query più frequenti presenti nel file di log del motore di ricerca di AOL.

Quindi le due differenze principali con il classifier precedente sono il tentativo di slegare i topic dalle singole query, e cercare di classificare una query mediante le parole chiave ottenute da essa. Pultroppo questo classifier ha mostrato fin da subito la sua limitatezza (infatti non è stato sottoposto a test, come i classifier che verranno presentati successivamente) : alcuni topic venivano ritornati troppo frequentemente. Visto che questo classifier e il precedente presentavano alcuni problemi in comune si è deciso di cambiare completamente approccio.

4.3 Classifier basato su Support Vector Machine

Dopo un certo periodo di ricerca di risorse utili per la crezione di un classifier, come disponibilità di dataset e/o servizi di directory search attualmente disponibili, si è realizzato che le uniche risorse direttamente disponibili erano Freebase e ODP. Il primo è accessibile mediante chiamate alle sue api, mentre il secondo mette a disposizione un file di dump aggiornato periodicamente e contenente tutte le sue risorse. Per il classificatore basato su svm si è utilizzato solo il file di dump di ODP seguendo i passi riportati di seguito .

Preprocessing

Il file di dump è un file xml che rappresenta i vari collegamenti tra le categorie, strutturate in livelli, e tutte le risorse presenti nel database di ODP. Queste informazioni sono molto utili per la costruzione di un dataset per due motivi : sono in numero elevato, circa 1,5 milioni di risorse, e sono state inserite e catalogate da persone. Le risorse sono costituite da una url, una descrizione e un titolo (in modo molto simile a come sono visualizzati i risultati su una pagina di un motore di ricerca) e ad ognuna di esse è associata un'unica categoria. Come detto le categorie sono strutturate a livelli, ma per il progetto Milkrain non è necessario l'utilizzo di una così ricca tassonomia e si è quindi deciso di tagliare le categorie ODP considerando solo i primi due livelli.

Il passo successivo è stato quello di rimuove tutte le parole non significative, come congiunzioni e simboli che non sono lettere, e applicare uno

stemmer alle risorse trovate. Un algoritmo di stemmificazione ha il compito di ridurre/semplificare le parole in stemmi in modo tale che parole molto simili siano ridotte allo stesso stemma. Per esempio le parole "stemmer", "stemming" e "stemmed" dovrebbero essere ridotte tutte allo stesso stemma "stem". Si noti che lo stemma di una parola non è la radice morfologica della parola ma solo un suo particolare prefisso. Esistono numerosi stemmer e di varie tipologie (basati su tabelle di ricerca, basati su regole, utilizzanti metodi statistici, etc.) ma una soluzione molto utilizzata e che non richiede strutture dati particolari è lo stemmer di Porter che è stato proposto nel 1980 ed è stato continuamente perfezionato. Secondo Wikipedia questo stemmer è di fatto l'algoritmo standard per la stemmificazione di parole inglesi.

Analisi e rappresentazione delle risorse

Le risorse così ottenute sono state analizzate per estrapolare le parole che sono più frequenti all'interno della collezione. Dall'analisi emerge che 15130 parole (o meglio stemmi) coprono l' 80 % delle singole parole presenti nelle risorse ODP, quindi nelle fasi successive si utilizzeranno le 20000 parole più frequenti per rappresentare una risorsa. Tale rappresentazione è stata effettuata costruendo la matrice A dove ogni riga rappresenta una risorsa e ogni colonna rappresenta una parola. Quindi l'elemento in posizione (i, j) rappresenta il legame tra l'i-esimo documento e la j-esima parola e nel nostro caso si è utilizzato lo schema tf-idf, normalizzando il tutto dividendo per l'elemento di valore massimo $+0.00001$. Infine le righe della matrice ottenuta e le categorie ODP tagliate al secondo livello costituiscono il training set per la costruzione della support vector machine. Per motivi di limitatezza delle memoria fisica disponibile non si sono usate tutte le risorse presenti nel file di dump ma solo un massimo di 100 risorse per ogni categoria.

Creazione Svm e classificazione di query

La creazione della svm è stata effettuate mediante le librerie *Weka* e *LibSVM* utilizzando sia una svm lineare con soft margin sia una svm non lineare con kernel rbf. Per entrambe le tipologie si sono effettuati vari test per la ricerca dei parametri ottimali utilizzando parte delle risorse non utilizzate nel dataset per la validazione dei risultati. Pultroppo alcuni topic non sono rappresentati nel validation set perchè presentavano poche risorse associate e tutte queste risorse erano state già utilizzate nel training set. I risultati ottenuti sono abbastanza simili, ma la svm lineare si è dimostrata più precisa classificando correttamente 20347 su 21498 risorse.

Ora che si dispone della svm per classificare una query si deve procedere nel seguente modo :

1. sottoporre la query ai data source e recupere le risorse ;

2. rappresentare tali risorse mediante lo schema tf-idf e normalizzando ;

3. classificare le risorse mediante la svm ;

4. scegliere i n topic di maggioranza ottenuti.

Nell'attuale implementazione vengono classificate tutte le risorse ritornate dai motori di ricerca, mentre il numero di topic restituito massimo è pari a 3 , come suggerito dagli esperimenti effettuati in [6].

Classificatore basato su svm utilizzante topic Freebase

Il classificatore costrutito finora, che chiameremo semplicemente SVM, utilizza i topic propri di ODP, mentre il classificatore originale era basato sui topic di Freebase. In un primo momento, ancor prima di adottare i topic di ODP, si era tentato di costruire un classificatore basato su support vector machine ma utilizzante i topic Freebase : la costruzione di tale classificatore è uguale a quanto riportato nelle sottosezioni precedenti ma con l'aggiunta di un passo importante nella creazione del dataset. Infatti occorre mappare ogni categoria di ODP in una categoria di Freebase. Per fare questo si sono utilizzati i topic ODP in forma originale e la seguente procedura :

1. stemmificare ogni topic ODP $t_{ODP} =< c_1, c_2, \cdots, c_n >$ ottenendo $\hat{t_{ODP}} =< \hat{c_1}, \hat{c_2}, \cdots, \hat{c_n} >$ e ogni topic Freebase in modo analogo ;

2. verificare per ogni topic ODP $\hat{t_{ODP}} =< \hat{c_1}, \hat{c_2}, \cdots, \hat{c_n} >$ se esiste un topic Freebase $\hat{t_F}$ tale $\hat{t_F} = \hat{c_j}$ per un qualche $1 \leq j \leq n$. In caso affermativo $\hat{t_{ODP}}$ viene mappato in $\hat{t_F}$.

Come è intuibile non tutti i topic ODP trovano un corrispondente Freebase e non viene garantita la totale correttezza della funzione di mapping ottenuta. Indicheremo questo classificatore con la sigla SVM-FB.

Prestazioni

Il nuovo e il precedente classifier sono stati testati per verificare se c'è stato un passo avanti nella classificazione di query o se gli sforzi effettuati sono stati inutili. Il test consiste nel classificare 100 query prese casualmente [1] con entrambi i classificatori e poi chiedere a 7 persone quante categorie ritornate sono appropriate. Si riportano in tabella i rusultati ottenuti :

L'ultima colonna risporta i valori di precisione considerando solo il primo topic ottenuto dal classificatore SVM. I valori della prima riga sono stati calcolati nel seguente modo :

[1] da un file di log di AOL http://gregsadetsky.com/aol-data/

	Benvenuti	**SVM-FB**	**SVM**	**SVM-1**
Precisione media per query	0.254	0.460	0.579	0.767
Precisione canonica	0.347	0.399	0.575	0.767

Tabella 4.2: Confronto prestazioni classifier

1. per ogni query viene calcolato il valor medio della precisione ottenuta secondo i 7 valutatori ;

2. viene calcolata la media dei valori ottenuti al passo precedente.

mentre i valori della seconda riga sono stati calcolati nel seguente modo:

1. per ogni valutatore viene calcolato il valore di precisione ottenuto dal classificatore ;

2. viene calcolata la media dei valori ottenuti al passo precedente.

Va detto che in letteratura viene riportato il valore di precisione calcolato come nella seconda riga della tabella, ma nel nostro caso abbiamo calcolato anche la precisione media per query per sottolineare il fatto che il vecchio classifier ha un comportamento peggiore dovuto al fatto che esso non ritorna alcun topic per 31 query. Come si può ben vedere i risultati di precisione ottenuti sono ben maggiori rispetto al classifier precedente e sono confrontabili con quelli dei lavori [5] [6] [7] . Inoltre risulta che il classificatore SVM-FB ha un precisione inferiore al classificatore SVM : questo fatto è da imputarsi molto probabilmente al mapping effettuato dei topic ODP nei corrispondenti topic Freebase che non è stato fatto manualmente ma da una procedura automatica. Infine confrontando le ultime due colonne si ritrova un'osservarzione fatta in [6] : all'aumentare del numero di categorie ritornate la precisione diminuisce. Pultroppo non si è riusciti a calcolare il recupero dei vari classificatori, sopratutto per la mancanza di persone volenterose che si prestassero a trovare per ogni query tutti i topic, sia di Freebase che ODP, adatti per descrivere la query.

Capitolo 5

Conclusioni

Il lavoro svolto durante l'attività di tesi ha portato sostanziali benefici al progetto Milkrain, seguendo e migliorando il modello proposto dall' Ing. Stefano Benvenuti e lavorando anche al fianco dell' Ing. Luca Bianco ([4]) per inserire la componente sociale data dalle informazioni che si possono prelevare dai vari social network sugli utenti di Milkrain.

Il progetto ora dispone di una macchina server dove possono lavorare più sviluppatori e che rende accessibile il servizio Milkrain ad eventuali utilizzatori e/o tester. In particolare l'utilizzo del progetto è garantito da un sito web sviluppato grazie al framework Lift : sebbene l'interfaccia sia molto semplice e non proprio accattivante, essa rende accessibili tutte le funzionalità offerte dal core di Milkrain.

L'adozione del linguaggio Scala per lo sviluppo front-end ha garantito tempi di risposta inferiori rispetto alla precedente interfaccia web, che era stata scritta in Php, per tutte le pagine che richiamano codice Java. Questo fatto è molto importante visto che la maggior parte delle funzionalità del sito fanno parte del core di Milkrain (sviluppato in Java). Un discorso ancora aperto è il consumo di memoria medio per query del server Jetty. Infatti, si è visto che lasciando alla JVM la scelta del garbage collector da utilizzare il server Jetty consuma molta più memoria rispetto alla sola applicazione Milkrain, ma si è anche visto che la situazione migliora in modo evidente imponendo l'utilizzo del CMS. Quindi si può presumere che quando il progetto verrà ospitato da un hardware multi processore, che renda efficace l'utilizzo di quest'ultimo garbage collector, l'aggravio di memoria introdotto dal server venga ridotto notevolmente.

Infine, per quanto riguarda il classificatore si può affermare che sono stati fatti dei passi in avanti grazie a parte del lavoro svolto durante questa tesi, ma si può altrettanto augurare la continuazione degli sforzi per affrontare il problema. Come dimostrato dai test effettuati il nuovo classificatore è caratterizzato da una precisione maggiore rispetto alla precisione raggiunta dal classificatore precedente, ed inoltre si avvicina ai valori di precisione

raggiunti dai lavori presenti in letteratura. Il nuovo classificatore risolve anche il problema delle query non classificate, grazie al fatto che esso lavora sulle risorse recuperate dai data sources e non direttamente sulla query. Il lavoro su questo modulo però non può essere considerato definitivo, in quanto è stato limitato soprattutto dalla mancanza di tempo e di risorse. Credo che sarebbe utile dedicare a questo problema un'intera tesi, in modo che si possa implementare un'altro classificatore (come ad esempio quello descritto in [5]) e confrontarlo con quello attuale. Un' altra possibilità da affrontare è quella di unire il nuovo classificatore con quello esistente per avere un classificatore ibrido.

Appendice A

Garbage Collector generazionali

In questa sezione vengono riassunti i principi più importanti in materia di Garbage Collector in ambiente Java. Tutte le informazioni che seguono si possono trovare in [10] e in [11].

A.1 Introduzione

Un oggetto viene considerato *garbage* quando non è più possibile accedere ad esso dal programma in esecuzione. Questi oggetti quindi risiedono nella memoria senza possibilità di riutilizzo e da qui nasce la necessità di rimuoverli rendendo nuovamente disponibile la memoria che essi occupano.

Dalla versione 1.2 Java include più algoritmi per il trattamento di tali oggetti che vengono utilizzati e combinati mediante la tecnica *generational collection* : gli oggetti sono suddivisi in *generazioni* e ogni generazione viene analizzata da un algoritmo in modo indipendente dalle altre. Questa strategia si fonda su numerose osservazioni empiriche effettuate negli anni al fine di rendere minimo il lavoro svolto dal GC. La più importate di queste osservazioni è l'assunzione che molti oggetti rimangono accessibili per brevi periodi di tempo, mentre solo una piccola parte degli oggetti ha dei tempi di vita più lunghi (*weak generational hypothesis*).

E' stata proprio questa ipotesi a portare gli sviluppatori Java a suddividere gli oggetti in 2 generazioni : *young generation* e *tenured generation* . Si può pensare alle due generazioni come a dei contenitori di oggetti con delle dimensioni opportune : quando una generazione satura interviene il GC che cerca di rimuovere quanti più oggetti possibile. Se la generazione saturata è la young generation si parla di *minor collection* e in questo caso il GC interviene esclusivamente su di essa, mentre si parla di *major collection* (o *full collection*) quando viene saturata la tenured generation. Se il

GC utilizzato prevede la compattazione della memoria essa viene eseguita in modo indipendente per le due generazioni.

Siccome le due generazioni ospitano oggetti di natura diversa esse presentano caratteristiche diverse. In particolare la young generation risulta essere molto più dinamica in quanto molti nuovi oggetti vengono continuamente allocati in essa dal programma e molti di essi muoio direttamente nella young generation, prima di essere promossi. Quindi questa generazione solitamente richiede delle collection molto più frequentemente rispetto alla tenured generation, la quale risulta più statica in quanto pochi oggetti riescono a farne parte. Per quanto riguarda le dimensioni delle due regioni, si ha che la young generation è di solito più piccola della tenured generation : questo comporta che le minor collection siano si più frequenti, ma anche molto rapide rispetto alle major collection.

Se la weak generation hypothesis fosse vera si avrebbe che la maggior parte degli oggetti verrebbero rimossi durante le minor collection, mentre le major collection diventerebbero poco frequenti, ottenendo sostanziali benefici rispetto a un GC naive che scansiona periodicamente tutti gli oggetti in memoria.

A.2 Young Generation e Fast Allocation

La young generation è a sua volta suddivisa in tre aree : *Eden* in cui vengono inseriti i nuovi oggetti appena allocati e due *Survivor Spaces* di cui uno, `from` , contiene oggetti sopravvissuti ad almeno una collection e `to` che rimane vuoto fino alla prossima collection (i ruoli di `from` e `to` vengono invertiti dopo una collection).

L'allocazione di nuovi oggetti deve essere efficiente e sopratutto thread-safe, per questo gli sviluppatori Java hanno adottato la semplice tecnica *bump-the-pointer* combinandola con la tecnica *Thread-Local Allocation Buffer* (TLAB) : ad ogni thread viene riservata una parte della young generation in cui poter allocare nuovi oggetti e un semplice puntatore al primo spazio di memoria libero. Solo raramente un thread esausta il suo TLAB dovendo richiederne uno aggiuntivo e utilizzando tecniche di sincronizzazione per non occupare aree di memoria già riservate.

Quando avviene una minor collection viene eseguito un apposito algoritmo di garbage collection, dipendente dal GC adottato, ottimizzato esclusivamente per la young generation.

A.3 Tenured Generation

La tenured generation non è ulteriormente suddivisa, ma quando essa satura il GC interviene sull'intero heap. Usualmente in questa generazione

sono presenti molti più oggetti rispetto alla young generation e le sue dimensioni vengono aggiornate molto più raramente rispetto a quelle della young generation. Quindi le major collection sono più onerose rispetto alle minor collection, ma sono anche meno frequenti. Un algoritmo di garbage collection per questa generazione punta ad essere efficiente in spazio (perchè la tenured generation ha dimensioni maggiori e contiene usualmente poco garbage), mentre gli algoritmi per le minor collection puntano ad essere il più veloci possibile vista la frequenza con cui esse si presentano.

A.4 Misure di prestazione per Garbage Collection e primi flag

Nell'analisi dei GC i seguenti indici sono spesso utilizzati :

- *Throughput* indica la percentuale del tempo che non viene spesa in attività di garbage collection ;
- *Pauses* consistono in intervalli di tempo in cui l'applicazione sembra morta a causa di attività di garbage collection ;
- *Footprint* è il working set di un processo ;
- *Promptness* è il tempo che passa da quando un oggetto diventa garbage a quando la memoria che esso occupa viene liberata.

Si deve notare che applicazioni diverse richiedono il miglioramento di metriche diverse : ad esempio le pauses sono di solito ben mascherate dai ritardi introdotti dalla rete in applicazioni lato server e quindi in questi ambiti si presta attenzione più al throughput che alle pauses.

Un fattore molto importante che influenza le precedenti metriche è la grandezza delle generazioni. In generale avere una piccola young generation implica pauses brevi ma un abbassamento del throughput, mentre una grande young generation implica un throughput più alto ma peggiora tutte le altre metriche.

Alcuni flag che permettono di settare parametri riguardanti le generazioni e l'heap sono :

- *-Xmx* : spazio massimo occupato dall'heap ;
- *-Xms* : grandezza iniziale dell'heap ;
- *-XX:MinHeapFreeRatio* : percentuale minima di spazio libero in una generazione ;
- *-XX:MaxHeapFreeRatio* : percentuale massima di spazio libero in una generazione.

In particolare gli ultimi due settaggi consentono di influenzare le attività di garbage collection : se la percentuale di spazio libero in una generazione non è compresa tra i due valori allora la generazione viene aumentata o rimpicciolita in modo che il valore rientri nell'intervallo specificato. Poter specificare il massimo spazio occupato dall'heap aiuta a prevenire il numero di page faults e fenomeni di trashing.

Due regole empiriche riguardanti l'uso di applicazioni su macchine server sono :

- se non ci sono problemi con le pauses dare più memoria possibile all'heap ;
- aumentare la memoria disponibile all'applicazione all'aumentare del numero di processori.

A.5 Configurazione delle generazioni

Come già detto la grandezza delle generazioni è un fattore molto importante, inoltre si deve tener conto che nel caso di heap con taglia massima fissata, l'aumento della memoria affidata a una generazione provoca una riduzione delle memoria disponibile all'altra generazione. Quindi la scelta dipende molto dal tipo di applicazione e in particolare dalla distribuzione dei tempi di vita degli oggetti allocati.

Il parametri concernenti le dimensioni delle generazioni sono :

- *-XX:NewRatio=<n> : impone che la tenured generation sia n volte maggiore rispetto alla young generation ;*
- *-XX:NewSize=<n> e -XX:MaxNewSize=<m> impongono che la grandezza della young generation sia sempre compresa tra n e m.*

Anche per questo tipo di settaggio c'è un procedimento empirico molto utilizzato :

- fissare la grandezza massima dell'heap ;
- provare diverse dimensioni delle young generation ;
- essere sicuri che la tenured generation sia sufficientemente grande per contenere tutti gli oggetti che non muoio mai e con un 20 % di margine in più.

A.6 Quale Garbage Collector scegliere

Vediamo le caratteristiche salienti dei GC disponibili :

- Serial Collector : utilizza un unico thread ed è quindi consigliato per hardware con un singolo processore, ma anche per macchine a più processori se l'applicazione in questione utilizza poca memoria (meno di 100 MB approssimativamente) in quanto non comporta overhead dovuti alla comunicazione fra thread. Per specificare il suo utilizzo viene fornito il flag *-XX:+UseSerialGC* ;

- Parallel Collector : a differenza del precedente esegue le minor collection in modo parallelo [1] ed è quindi consigliato per hardware a più processori e applicazioni con grandi carichi di memoria. La gestione concorrente delle minor collection dovrebbe aumentare il throughput dell'applicazione. Per specificare l'utilizzo del parallel collector si deve utilizzare il flag *-XX:+UseParallelGC* ;

- Parallel Compacting Collector : aumenta il parallelismo del PC impiegando un algoritmo parallelo anche per le tenured collection. Per specificare il suo utilizzo viene utilizzata l'opzione *-XX:+UseParallelOldGC* ;

- *Concurrent Mark-Sweep Collector* : esegue gran parte del lavoro in modo concorrente all'applicazione riducendo così le pauses a discapito del throughput. Anche questo GC è consigliato per applicazioni che necessitano di molta memoria e che dispongono di più processori, il suo utilizzo viene specificato mediante il flag *-XX:+UseConcMarkSweepGC*.

A.7 Serial Collector

Il SC utilizza un unico thread per effettuare il proprio lavoro, fermando completamente l'intera applicazione. Durante una minor collection gli oggetti non garbage presenti in `eden` vengono copiati in to mentre gli oggetti non garbage in `from` vengono copiati in `to` oppure vengono passati alla tenured generation in base al numero di collection che hanno già trascorso nella young generation. Se lo spazio in `to` non premette la copiatura in esso di un oggetto, tale oggetto viene automaticamente promosso alla tenured generation. In questo modo gli unici oggetti ancora raggiungibili nella young generation si trovano in `to` e gli spazi `eden` e `from` possono essere

[1] Nell'ambito garbage collection la denominazione "parallelo" si riferisce al fatto che il lavoro viene svolto da più thread che fermano l'applicazione, mentre "concorrente" indica che le operazioni di garbage collection vengono eseguite in modo concorrente all'applicazione, senza provocarne il blocco.

dichiarati liberi. Durante la prossima minor collection i ruoli di `to` e `from` saranno scambiati.

La tenured generation viene ripulita tramite l'algoritmo *mark-sweep-compact collection algorithm* che esegue nell'ordine i seguenti compiti : marca gli oggetti vivi, libera lo spazio occupato dal garbage, compatta la generazione spostando tutti gli oggetti all'inizio di essa (e permettendo l'uso della tecnica bump-the-pointer per allocazioni future).

A.8 Parallel Collector

Il SC non può sfruttare le eventuali CPU presenti che rimangono inutilizzate durante la sua esecuzione : il PC risponde a questa carenza utilizzando più thread per le operazioni di garbage collection. In particolare le minor collection utilizzano una versione parallela dell'algoritmo utilizzato dal SC riducendo così l'overhead introdotto e aumentando il throughput dell'applicazione. Si noti che anche questa versione dell'algoritmo necessita il blocco dell'applicazione. La parallelizzazione invece non è stata introdotta per la ripulitura della tenured generation. Questo fatto deve essere tenuto in considerazione nei casi in cui si deve evitare pauses troppo lunghe in quanto le major collection potrebbero richiedere molto tempo.

Per il PC si può specificare il valore desiderato di alcune metriche di prestazione che l'algoritmo tenterà di soddisfare con la seguente priorità :

- pauses ;
- throughput ;
- footprint .

Appena conclusa un collection il PC raccoglie dei dati in modo da costruirsi un statistica interna per stimare le metriche, quindi aumenta o restringe le due generation cercando di migliorare le prestazioni di garbage collection fino a raggiungere gli obbiettivi richiesti. L'aggiustamento delle generation avviene in modo incrementale, ovverosia aumentando o diminuendo la generazione di una certa percentuale della sua dimensione attuale.

Le impostazioni disponibili per il PC sono :

- *-XX:MaxGCPauseMillis=<n>* indica che si desiderano pauses di n millisecondi o meno. L'uso di questa opzione può causare un abbassamento del throughput e non sempre il traguardo specificato può essere soddisfatto ;
- *-XX:GCTimeRatio=<n>* specifica che la frazione di tempo spesa in attività di garbage collection sia di 1 / (1+n) . Il valore di default è 99, ovverosia solo l'1% del tempo viene dedicato ad attività di garbage collection ;

- *-XX:YoungGenerationSizeIncrement=<y>* indica l'incremento percentuale della young generation (default 20 %) ;
- *-XX:TenuredGenerationSizeIncrement=<t>* indica l'incremento percentuale della tenured generation (default 20 %) ;
- *-XX:AdaptiveSizeDecrementScaleFactor=<d>* impone un restringimento della generazione pari al fattore di aumento diviso d (default 4) ;
- *-XX:DefaultInitialRAMFraction=<n>* serve per specificare la frazione delle memoria RAM disponibile che inizialmente viene affidata all'applicazione ;
- *-XX:DefaultMaxRAMFraction=<m>* specifica la frazione massima di memoria che l'applicazione dovrebbe utilizzare.

In particolare le ultime due opzioni vengono così interpretate :

- taglia iniziale dell'heap = memory / DefaultInitialRAMFraction ;
- taglia massima dell' heap = min(memory / DefaultMaxRamFraction , 1 GB) .

A.9 Parallel Compacting Collector

Il PCC introduce la ripulitura della tenured generation (oltre alla young generation) in modo parallelo. Il lavoro svolto consiste di tre fasi :

1. Mark Phase. La generazione viene logicamente suddivisa in *regioni* di taglia fissa e gli oggetti raggiungibili dal codice dell'applicazione vengono analizzati in modo parallelo da più thread. Quando un oggetto viene trovato, ed è quindi raggiungibile, alcune informazioni di esso (come collocazione ed estensione) vengono memorizzate in una tabella associata alla regione che lo contiene ;
2. Summary Phase. Grazie alle compattazioni della generazione effettuate nelle precedenti collection molto probabilmente la parte sinistra (o inferiore) della generazione consiste di regioni contenenti molti oggetti ancora raggiungibili e pochi oggetti garbage. Quindi ripulire tali regioni non comporta grandi benefici ed infatti esse non verranno prese in considerazione dal PCC. Con un approccio gredy il PCC cerca la prima regione da sinistra che ha un certo grado di garbage in essa e suddivide logicamente la generazione in due aree : la *dense prefix* che contiene regioni con molti oggetti raggiungibili e il resto della generazione che viene ripulita e compattata regione per regione. Inoltre

vengono aggiornati dei puntatori che indicano dove comincia lo spazio libero delle regioni appena compattate. Anche se un approccio parallelo è possibile, questa fase viene implementata in modo seriale in quanto il suo peso è molto inferiore rispetto al peso delle altre due fasi.

3. Compaction Phase. Utilizzando le informazioni raccolte nelle precedenti fasi, vengono riempite alcune regioni ottenendo un heap densamente occupato all' inizio seguito da un spazio di memoria completamente libero. Anche questa fase è implementata in modo parallelo.

Questo collector non viene consigliato in sistemi con molte applicazioni avviate in quanto la probabilità di utilizzare tutti i core della CPU per sufficenti periodi di tempo è abbastanza bassa. In questi casi ci sono due possibilità : utilizzare un GC differente oppure abbassare il numero di thread attraverso l'opzione *-XX:ParallelGCThread*.

Comcurrent Mark-Sweep Collector

L'ultimo GC cerca di migliorare le pauses rendendo concorrente, al programma in esecuzione, gran parte del lavoro svolto per le major collection : i thread dell'applicazione sono fermati solo in situazioni particolari, come ad esempio quando la tenured generation satura o quando si deve allocare un oggetto senza disporre di sufficiente memoria. La situazione in cui il GC non riesce a completare una collection in modo concorrente viene detta *concurrent mode failure* . In questo modo il programma viene interrotto a causa di una major collection per intervalli di tempo minori rispetto ai precedenti GC a fronte di un abbassamento del throughput complessivo dell'applicazione. Va da se che il CMS dà risultati migliori all'aumentare del numero dei processori.

Senza entrare troppo nel dettaglio il CMS esegue il proprio lavoro sulla tenured generation alternando fasi di inattività con fasi di lavoro. Una fase attiva, o *collection cycle*, consiste nei seguenti passi :

1. ferma tutti i thread dell'applicazione per costruire il grafo degli oggetti direttamente raggiungibili dal programma e da oggetti della young generation (*initial mark pause*) ;

2. in modo concorrente all'esecuzione dell'applicazione espande il grafo degli oggetti raggiungibili utilizzando 1 o più processori ;

3. ferma tutti i processi dell'applicazione per aggiornare il grafo degli oggetti raggiungibili : alcune modifiche possono essere dovute alla concorrente esecuzione del programma nella seconda fase (*remark pause*) ;

4. in modo concorrente libera gli oggetti non più raggiungibili, modifica le dimensioni dell'heap e prepara le strutture dati per il successivo collection cycle utilizzando 1 processore.

Tra le due pauses quella più impegnativa è la remark pause : per questo essa viene eseguita da più thread.

Una caratteristiche che contraddistingue il CMS è il fatto che esso non esegue una compattazione della tenured generation, ma lascia gli oggetti dove sono. L' inserimento di nuovi oggetti in questa generazione non avviene più tramite la tecnica bump-the-pointer ma vengono mantenute una serie di liste agli spazi liberi nella generazione : ne consegue un certo overhead delle minor collection dovuto alla maggior complessità nello spostare oggetti nella tenured generation.

Un obbiettivo che il CMS cerca di soddisfare è quello di schedulare le major collection in modo che esse termino prima che la tenured generation diventi satura, in modo da evitare un concurrent mode failure. Esistono varie euristiche basate sui dati raccolti durante l'esecuzione del programma che cercano di prevedere quanto tempo occorre per effettuare il prossimo collection cycle e quando avverrà la prossima saturazione della tenured generation. Un altro evento che fa iniziare un collection cycle consiste nello sforamento di una certa soglia percentuale della memoria occupata nella tenured generation : tale soglia può essere impostata attraverso il flag *-XX:CMSInitiatingOccupancyFraction=<n>*.

Al fine di limitare l'influenza negativa del GC sul throughput dell'applicazione è possibile abilitare la modalità *incremental mode* (*-XX:+CMSIncrementalMode)* in cui il lavoro concorrente eseguito dal CMS viene suddiviso in jobs, evitando di occupare uno o più core esclusivamente dal GC. Esistono molti flag che si possono utilizzare per manipolare il funzionamento del GC in incremental mode.

A.10 Garbage-First Garbage Collector (G1)

Anche se è ancora in fase di sviluppo è possibile utilizzare questo nuovo soft real-time GC, che dovrebbe sostituire il CMS nelle versioni future di Java. La memoria è suddivisa in *regions* di ugual grandezza e a loro volta suddivise in *card* da 512 byte. Il lavoro svolto da G1 è suddiviso in tre stadi :

1. *Remembered Set Maintenance* . Ad ogni region viene associato il corrispondente Remembered Set (RS) che consiste dell'insieme delle regioni che contengono riferimenti ad oggetti della regione in esame ;

2. *Concurrent Marking* . G1 ricerca tutti gli oggetti ancora raggiungibili presenti nelle regioni e mantiene due bitmap per avere un report dello

stato dell'heap attuale e dell'heap precedentemente alla collection in corso. Questo stadio è a sua volta suddiviso in tre fasi. Inizialmente l'intero heap viene scansionato e viene aggiornata la bitmap corrente in base agli oggetti non garbage. Quando l' heap è utilizzato sopra una certa soglia percentuale avviene la seconda fase di re-marking per catturare tutti gli aggiornamenti che avvengono tra la prima e la seconda fase. In fine le regioni vengono ordinate in base al costo necessario per la loro ripulitura in modo che le regioni che vengono ripulite più efficientemente siano le prime a essere liberate ;

3. *Collection* . Le regioni che possono essere ripulite efficientemente vengono ora processate : gli oggetti ancora raggiungibili in esse vengono copiati e compattati in altre parti dell' heap e le regioni sono dunque disponibili per nuove allocazioni.

L'utilizzo del GC G1 è possibile grazie ai due flag *-XX:+UnlockExperimentalVMOptions* e *-XX:+UseG1GC* , mentre i settaggi disponibili sono i seguenti :

- *-XX:MaxGCPauseMillis=<n>* indica la durata massima delle pauses in millisecondi ;
- *-XX:GCPauseIntervalMillis=<n>* indica l'intervallo minimo tra una pause e la successiva ;
- *-XX:+G1YoungGenSize=<n>* impone la grandezza della young generation .

Appendice B

Analisi di collezioni di documenti

In questo capitolo vengono presentati alcuni tra gli strumenti più utilizzati per la classificazione di testi e in modo particolare come si possa rappresentare una collezione di documenti per poter eseguire delle analisi sulla somiglianza tra i documenti in essa contenuti.

B.1 Vector Space Model e Singular Value Decomposition

Indichiamo con $D = \{d_1, \cdots, d_n\}$ l'insieme di documenti da analizzare definiti su un dizionario $T = \{t_1, \cdots, t_m\}$ di termini. In questo modello l'elemento fondamentale è la matrice $A = [a_{i,j}]$ il cui compito è quello di rappresentare i documenti della collezione. In particolare l'elemento a_{ij} rappresenta il legame esistente tra il termine t_i e il documento d_j. Esistono varie possibilità per la scelta di come rappresentare tale legame :

1. *Binary weighting* : $a_{ij} = 1$ se t_i è presente nel documento d_j, 0 altrimenti ;
2. *Term frequency weighting* : $a_{ij} = tf_{ij}$ dove tf_{ij} indica la frazione dei ternimi uguali a t_i nel documento d_j ;
3. *Term frequency inverse document frequency* : $a_{ij} = tf_{ij} * idf_i$ dove $idf_i = log(\ ||D|| \ / \ ||\ d : t_i \in d\ ||\ )$.

Come è facile intuire le metriche sono state presentate in ordine di efficacia nel rappresentare i legami termine - documento. Infatti il secondo schema migliora il primo perchè assegna pesi maggiori ai termini che occorrono più frequentemente in un documento, mentre il terzo migliora il secondo limitando i pesi per quei termini che sono molto frequenti in tutti

i documenti e che quindi non sono specifici di un documento in particolare. Usualmente i valori della matrice A vengono normalizzati.

Nel seguito indicheremo con t_i sia il termine i-esimo che il vettore che esprime il legame tra il termine i-esimo e i documenti della collezione D, analogamente indicheremo con d_j anche il vettore associato al documento j-esimo, quindi le colonne e le righe della matrice A hanno i seguenti significati :

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} = \begin{bmatrix} \mathbf{d_1} & \mathbf{d_2} & \cdots & \mathbf{d_m} \end{bmatrix} = \begin{bmatrix} \mathbf{t_1}^T \\ \mathbf{t_2}^T \\ \vdots \\ \mathbf{t_m}^T \end{bmatrix} \tag{B.1}$$

Le colonne della matrice possono essere interpretate come i vettori di uno spazio vettoriale che descrive la collezione di documenti. Quindi si può utilizzare questa interpretazione per confrontare due documenti semplicemente calcolando il prodotto interno tra i rispettivi vettori $d_i^T \cdot d_j$ e similmente si può procedere per i termini. Tutti questi prodotti possono essere ragruppati nelle matrici :

1. AA^T il cui elemento di posizione (i, j) corrisponde alla similitudine tra i termini t_i e t_j ;
2. A^TA il cui elemento di posizione (i, j) corrisponde alla similitudine tra i documenti d_i e d_j .

Fondamentale teorema per l'analisi di collezioni di documenti mediante Vector Space Model è il seguente :

Teorema B.1.1 (Singular Value Docomposition)**.** *Una matrice $m \times n$ a valori reali M può essere sempre fattorizzata nella forma $M = U\Sigma V^T$ dove :*

1. *U e V^T sono matrici quadrate unitarie [1], le colonne di U rappresentano i vettori singolari sinistri, mentre le colonne di V rappresentano i vettori singolari destri di M;*
2. *Σ è una matrice diagonale rettangolare $diag(\sigma_1, \sigma_2, ...)$ dove $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq ...$ sono i valori singolari di M.*

La scomposizione ai valori singolari è legata agli autovalori e agli autovettori delle matrice originale seconodo la seguente proposizione :

[1] Una matrice reale U è unitaria se UU^T è la matrice indentità. Inoltre una matrice reale unitaria è anche una matrice ortogonale.

Proposizione B.1.2. *Sia $M = U\Sigma V^T$ la decomposizione ai valori singolari delle matrice M. Allora :*

1. *i vettori singolari sinistri di M sono gli autovettori di MM^T ;*
2. *i vettori singolari destri di M sono gli autovettori di M^TM ;*
3. *i valori singolari non nulli di M sono le radici quadrate degli autovalori non nulli della matrice MM^T (e della matrice M^TM).*

Applichiamo quindi la scomposizione ai valori singolari alla matrice A (supponendo, senza perdita di generalità $m > n$) ottenendo :

$$A = U\Sigma V^T = \begin{bmatrix} \mathbf{u_1} & \mathbf{u_2} & \cdots & \mathbf{u_n} \end{bmatrix} \cdot \begin{bmatrix} \sigma_\mathbf{1} & & & \\ & \ddots & & \\ & & & \sigma_\mathbf{n} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{v_1^T} \\ \mathbf{v_2^T} \\ \vdots \\ \mathbf{v_m^T} \end{bmatrix} \tag{B.2}$$

e indichiamo con $\hat{t_i}^T$ le righe di U e con $\hat{d_j}$ le colonne di V :

$$A = U\Sigma V^T = \begin{bmatrix} \hat{\mathbf{t_1}}^T \\ \hat{\mathbf{t_2}}^T \\ \vdots \\ \hat{\mathbf{t_m}}^T \end{bmatrix} \cdot \begin{bmatrix} \sigma_\mathbf{1} & & & \\ & \ddots & & \\ & & \sigma_\mathbf{n} & \\ & & & \end{bmatrix} \cdot \begin{bmatrix} \hat{\mathbf{d_1}} & \hat{\mathbf{d_2}} & \cdots & \hat{\mathbf{d_n}} \end{bmatrix} \tag{B.3}$$

Come si può vedere l'unica parte della matrice U che influenza un vettore termine t_i^T è il vettore riga $\hat{t_i}^T$, mentre l'unica parte di V che influenza un vettore documento d_j è il vettore colonna $\hat{d_n}$. Quindi per confrontare la somiglianza tra i due documenti d_i e d_j si può procedere nel seguente modo (H_i indica l'i-esima riga della matrice H , mentre H^j indica la j-esima colonna della matrice H):

$$\begin{aligned} d_i^T \cdot d_j &= A^TA(i,j) = (U\Sigma V^T)^T(U\Sigma V^T)(i,j) \\ &= (V\Sigma^TU^T)(U\Sigma V^T)(i,j) = (V\Sigma^T\Sigma V^T)(i,j) \\ &= (V\Sigma^T)_i(\Sigma V^T)^j = (v_i\Sigma^T)(\Sigma V^T)^j \\ &= (\hat{d_i}^T\Sigma^T)(\Sigma\hat{d_j}) = (\Sigma\hat{d_i})(\Sigma\hat{d_j}) \end{aligned} \tag{B.4}$$

La stessa tecnica si può adoperare per trovare il documento più pertinente data una query. Per rappresentare la query si utilizza un vettore q come se la query fosse un nuovo documento della collezione, prima però si deve rappresentare il vettore q nell'adeguato spazio vettoriale :

$$d_j = U\Sigma\hat{d_j} \Rightarrow \hat{d_j} = \Sigma^{-1}U^Td_j \Rightarrow \hat{q} = \Sigma^{-1}U^Tq \tag{B.5}$$

B.2 Latent Semantic Analysis

Il vector space model raprresenta un primo strumento con cui analizzare una collezione di documenti ma soffre di alcune limitazioni :

- due documenti possono trattare gli stessi argomenti anche se hanno pochissimi termini in comune : tali documenti non vengono riconosciuti come simili dal vector space model ;
- il significato di un termine può dipendere dal contesto, mentre il vector space model tratta tutte le parole come se fossero indipendenti.

La tecnica Latent Semantic Analysis cerca di ridurre queste limitazioni riducendo la dimensione della matrice A. L'ipotesi che viene fatta è che esista uno spazio nascosto che modella i concetti dei documenti nella collezione e la riduzione del rango della matrice sia un modo per "avvicinarsi" a tale spazio. Tale riduzione viene effettuata annullando tutti i valori della diagonale nella matrice Σ tranne i primi k :

$$A_k = U_k \Sigma_k V_k^T = \begin{bmatrix} \hat{\mathbf{t_1}}^T \\ \hat{\mathbf{t_2}}^T \\ \vdots \\ \hat{\mathbf{t_m}}^T \end{bmatrix} \cdot \begin{bmatrix} \sigma_\mathbf{1} & & \\ & \ddots & \\ & & \sigma_\mathbf{k} \end{bmatrix} \cdot \begin{bmatrix} \hat{\mathbf{d_1}} & \hat{\mathbf{d_2}} & \cdots & \hat{\mathbf{d_n}} \end{bmatrix} \quad \text{(B.6)}$$

dove U_k è la matrice formata dalle prime k colonne della matrice U, Σ_k è la matrice formata dalle prime k colonne e prime k righe di Σ e V_k^T è formata dalla prime k righe di V^T.

La scelta del parametro k può essere fatta imponendo che il rapporto tra le norme di Frobenius della matrice ridotta e la matrice originale sia più grande di una certa soglia :

$$\frac{\sqrt{\sum_{i=1}^{k} \sigma_i^2}}{\sqrt{\sum_{i=1}^{n} \sigma_i^2}} \geq \text{soglia} \quad \text{(B.7)}$$

Un'altro effetto della riduzione del rango è quello di rappresentare i documenti (e le query) in un spazio di dimensione k.

Appendice C

Classificazione e Support Vector Machine

Il problema della Classificazione è una parte importante del campo Data-mining, mentre le Support Vector Machine sono una particolare tecnica di classificazione. In questo capitolo vengono introdotti i passi fondamentali inerenti a questi due concetti.

C.1 Il problema della Classificazione

Il problema della classificazione, come facilmente intuibile, viene utilizzato per costruire una **target function** (o **modello**) a partire da un data set di esempi. Ogni esempio è una coppia $(\mathbf{x}, y)$ dove $\mathbf{x}$ è un insieme di attributi (*features*) che descrivono l'istanza in esame e y è un attributo speciale, detto *classe* o *categoria*, di tipo discreto. La target function è dunque una funzione che descrive accuratamente e sinteticamente le classi presenti nel dataset e che può essere utilizzata per classificare un nuovo elemento a partire dalle sue features.

Il processo di classificazione prevede almeno i seguenti passi :

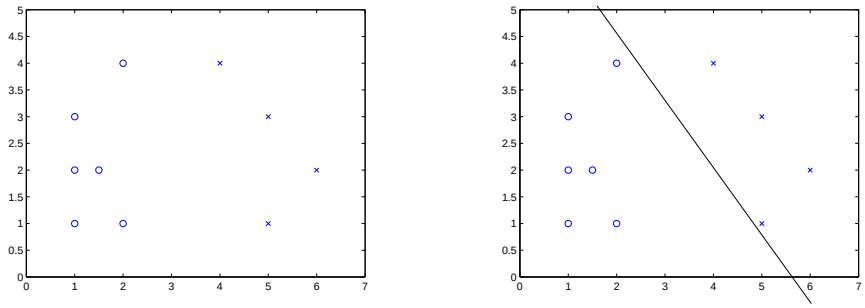
- Costruzione del data set ;
- Costruzione target function ;
- Validazione del modello ;
- Classificazione di nuovi elementi .

Usualmente si chiama *Train Set* l'insieme di esempi utilizzati per la costruzione del modello, mentre si chiama *Test Set* o *Validation Set* l'insieme di esempi utilizzati per la fase di validazione del modello. Sia il train set che il test set sono usualmente sottoinsiemi del data set ma è bene mantenere

questi due insiemi disgiunti : infatti, per valutare l'efficacia di un modello e la sua capacità di generalizzazione bisogna effetturare dei test senza utilizzare elementi che erano presenti nel train set.

Iperpiani e margine

Le svm sono una tecnica di classificazione supervisionata : a partire da una collezione di esempi classificati (training set) viene costruito un modello in grado di operare su nuovi dati, classificandoli in base agli esempi forniti. Usualmente un esempio viene descritto da un vettore $\mathbf{x_i} = (x_{i1}, \cdots, x_{in})^T$ e da una classe y_i che per il momento restringiamo a $\{-1, +1\}$.



(a) Esempio training set

(b) Esempio iperpiano di separazione

Nell'esempio in figura C.1 si vede immediatamente che le istanze appartenenti alle due categorie sono facilmente divisibili da una retta. Questo concetto di separabilità può essere esteso anche agli spazi $\mathbb{R}^n$ per n qualsiasi utilizzando il concetto di iperpiano :

Definizione C.1 (Iperpiano)**.** Si definisce iperpiano di $\mathbb{R}^n$ un qualsiasi suo sottospazio di $n - 1$ dimensioni.

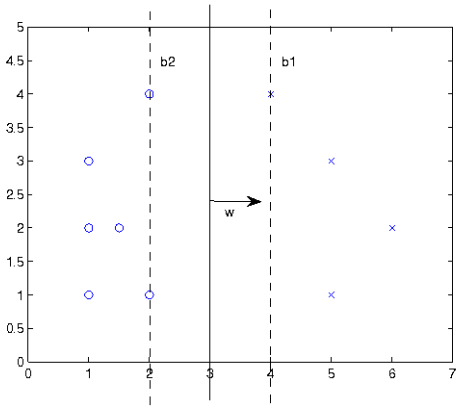
Si deve notare che non tutti i problemi di classificazione presentano dati separabili semplicemente da un iperpiano : in questi casi si dice che essi non sono linearmente separabili. Anche se le svm sono in grado di trattare dati non linearmente separabili è utile supporre almeno per il momento di trattare dati linearmente separabili.

Secondo la definizione data un iperspazio del piano è dunque una retta, mentre un iperspazio dello spazio tridimensionale è un qualsiasi piano bidimensionale. Un iperpiano può essere descritto da un'equazione del tipo :

$$\mathbf{w} \cdot \mathbf{x} + b = 0 \tag{C.1}$$

Si ricorda che il vettore $\mathbf{w}$ risulta essere perpendicolare a qualsiasi vettore contenuto nell'iperpiano dato dalla formula C.1. Fissato un iperpiano B che

divide le classi del trainig set sono automaticamente fissati i due iperpiani $b1$ e $b2$ paralleli a B e che passano per i punti del training set più vicini a B : $b1$ si ottiene traslando B fino a toccare un esempio con classe 1, mentre $b2$ si ottiene traslando B fino a raggiungere un esempio di classe -1.



La distanza tra $b1$ e $b2$ viene detta *Margine* dell'iperpiano B. E' comunemente accettata l'ipotesi che l'iperpiano di margine massimo è l'iperpiano migliore al fine della classificazione di nuovi elementi. Il calcolo del margine non presenta difficoltà. Per come sono stati definiti gli iperpiani $b1$ e $b2$ si può riscalare il vettore $\mathbf{w}$ in modo che gli iperpiani $b1$ e $b2$ siano definiti dalle seguenti equazioni :

$$\begin{aligned} b1 &: \mathbf{w} \cdot \mathbf{x} + b = 1 \\ b2 &: \mathbf{w} \cdot \mathbf{x} + b = -1 \end{aligned} \tag{C.2}$$

Siano $\mathbf{x_1}$ e $\mathbf{x_2}$ due punti di $b1$ e $b2$ rispettivamente, allora la distanza tra i due iperspazi può essere calcolata proiettando il vettore $\mathbf{x_1} - \mathbf{x_2}$ sul vettore $\mathbf{w}$:

$$\|p_{\mathbf{w}}(\mathbf{x_1} - \mathbf{x_2})\| = \frac{\mid (\mathbf{x_1} - \mathbf{x_2}) \cdot \mathbf{w} \mid}{\|\mathbf{w}\|} = \frac{1 - b - (-1 - b)}{\|\mathbf{w}\|} = \frac{2}{\|\mathbf{w}\|} \tag{C.3}$$

Trovato l'iperpiano (descritto dal vettore $\mathbf{w}$ e dal parametro b) che massimizza il margine, la classificazione di un nuovo elemento x viene effettuata valutando la funzione :

$$sgn(\mathbf{w} \cdot \mathbf{x} + b) \tag{C.4}$$

C.2 Support Vector Machine

C.2.1 SVM Lineari

Sia $D = \{x_1, \cdots, x_N\}$ un training set linearmente separabile. La costruzione di una svm lineare per D viene formalizzata nella definizione C.2.

Definizione C.2 (SVM Lineare)**.** La soluzione del seguente problema di minimizzazione rappresenta una svm lineare.

$$min_{\mathbf{w}} \ \tfrac{1}{2}\|\mathbf{w}\|^2$$

$$\text{ristretto a} \quad y_i(\mathbf{w} \cdot \mathbf{x_i} + b) \geq 1 \quad i = 1...N$$

Nella definizione viene utilizzata la funzione $\|\mathbf{w}\|^2/2$ anzichè $\|\mathbf{w}\|/2$ solo per semplificare la risoluzione del problema, mentre le condizioni $y_i(\mathbf{w} \cdot \mathbf{x_i} + b) \geq 1$ esprimono la condizione che l'iperpiano debba dividere l'iperspazio in due regioni : una contenente i punti di classe $+1$, l'altra contenente i punti di classe -1.

Il problema in C.2 può essere risolto utilizzando il metodo dei moltiplicatori di Lagrange. Definiamo quindi una nuova funzione obbiettivo che incorpori sia la ricerca del minimo della funzione obbiettivo sia i vincoli imposti. Tale funzione avrà la seguente struttura :

$$L = \frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{N} p_i$$

dove il termine p_i rappresenta una penalità infinita se il vincolo i-esimo non viene rispettato, nullo altrimenti :

$$p_i = max_{\lambda_i} \lambda_i(-1)(y_i(\mathbf{w} \cdot \mathbf{x_i} + b) - 1) \tag{C.5}$$

ottenendo quindi :

$$L = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{N} \lambda_i(y_i(\mathbf{w} \cdot \mathbf{x_i} + b) - 1) \tag{C.6}$$

dove λ_i sono delle nuove variabili. Tale funzione viene detta *Lagrangiana*, mentre i λ_i vengono detti *moltiplicatori di Lagrange*. Per trovare il punto di minimo del problema C.2 si deve quindi minimizzare L rispetto a $\mathbf{w}$ e b, ma massimizzandola rispetto ai λ_i. Il seguente teorema da delle informazioni circa i punti di minimo locale di L :

Teorema C.2.1 (Condizioni Karush-Kuhn-Tucker)**.** *Sia* $(\overline{\mathbf{w}}, \overline{b})$ *un punto di minimo locale per* $\frac{1}{2}\|\mathbf{w}\|^2$ *sotto le condizioni* $y_i(\mathbf{w} \cdot \mathbf{x_i} + b) \geq 1$*. Allora :*

1. $\lambda_i \geq 0$;
2. $(\overline{\mathbf{w}}, \overline{b})$ *è stazionario per* L ;
3. $y_i(\overline{\mathbf{w}} \cdot \mathbf{x_i} + \overline{b}) \geq 1$;
4. $\lambda_i(y_i(\overline{\mathbf{w}} \cdot \mathbf{x_i} + \overline{b}) - 1) = 0$.

Le condizioni KKT facilitano la ricerca di un minimo locale di L restingendo il campo ai soli punti che le soddisfano. Si osservi come molti moltiplicatori di lagrange sono in realtà nulli per la condizione 4 : solo i moltiplicatori che corrispondo a istanze $\mathbf{x_i}$ appartenenti agli iperspazi $b1$ e $b2$ possono essere non nulli.

Per minimizzare L, coerentemente alla seconda condizione KKT, si impongono nulle le sue derivate parziali :

$$\frac{\partial L}{\partial w_i} = 0 \Longrightarrow \mathbf{w} = \sum_{i=1}^{N} \lambda_i y_i \mathbf{x_i} \tag{C.7}$$

$$\frac{\partial L}{\partial b} = 0 \Longrightarrow \sum_{i=1}^{N} \lambda_i y_i = 0 \tag{C.8}$$

e sostituendo in L si ricava :

$$L_D = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i=1}^{N} \lambda_i \lambda_j y_i y_j \mathbf{x_i} \cdot \mathbf{x_j} \tag{C.9}$$

La funzione L_D viene detta "formulazione duale" del problema originario e deve essere massimizzata rispetto alle variabili λ_i : questo passo può essere risolto mediante metodi numerici standard. Infine grazie a (C.7) si può ricavare $\mathbf{w}$, mentre b viene ricavato dalle condizioni KKT di tipo 4. In realtà l'utilizzo di metodi numerici porta ad avere valori diversi per b, dipendentemente dalla condizione KKT utilizzata per il calcolo : solitamente quindi si prende il valore medio utilizzando tutte le condizioni KKT con $\lambda_i \neq 0$.

C.2.2 SVM Lineari con margine rilassato

Come si è visto le svm lineari dividono perfettamente l'iperspazio in due parti : una contenete gli esempi di classe $+1$ e l'altra contenente quelli di classe -1. In alcuni casi però l'iperpiano trovato risulta essere troppo vicino agli elementi delle due classi e quindi molto incline al problema dell'overfitting. In questi casi si permette la classificazione errata di alcuni elementi del training set introducendo delle nuove variabili ξ_i positive e un parametro C di penalità :

$$min_{\mathbf{w},\xi}\ \tfrac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i$$

$$\text{ristretto a} \qquad y_i(\mathbf{w}\cdot\mathbf{x_i}+b) \geq 1-\xi_i \qquad i=1...N$$

Procedendo come per le svm lineari si definisce la funzione Lagrangiana e si impongono nulle le sue derivate parziali :

$$L = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi - \sum_{i=1}^{N}\lambda_i(y_i(\mathbf{w}\cdot\mathbf{x_i}+b)-1+\xi) - \sum_{i=1}^{N}\mu_i\xi_i \qquad \text{(C.10)}$$

$$\frac{\partial L}{\partial w_j} = w_j - \sum_{i=1}^{N}\lambda_i y_i x_{ij} = 0 \Longrightarrow \mathbf{w} = \sum_{i=1}^{N}\lambda_i y_i x_i \qquad \text{(C.11)}$$

$$\frac{\partial L}{\partial b} = -\sum_{i=1}^{N}\lambda_i y_i = 0 \Longrightarrow \sum_{i=1}^{N}\lambda_i y_i = 0 \qquad \text{(C.12)}$$

$$\frac{\partial L}{\partial \xi} = C - \lambda_i - \mu_i = 0 \Longrightarrow \lambda_i + \mu_i = C \qquad \text{(C.13)}$$

Sorprendentemente sostituendo le ultime espressioni trovate in L si trova lo stesso problema di massimo C.9, anche se in questo caso le condizione sui moltiplicatori di Lagrange sono un po' diverse : infatti da C.13 (e dalle condizioni KKT che non vengono riportate) appare chiaro che :

$$0 \leq \lambda_i \leq C \qquad \text{(C.14)}$$

C.2.3 SVM Non Lineari

Nel caso di dati non linearmente separabili ci sono due possibilità : utilizzare una svm lineare con margine rilassato oppure utilizzare una svm non lineare. Quest'ultima in realtà è una svm lineare : la non separabilità lineare dei dati viene rimossa grazie a una funzione $\Phi : \mathbf{x} \longrightarrow \Phi(\mathbf{x})$. Questa funzione ha quindi lo scopo di trasformare i dati mappandoli su un nuovo spazio di $m \gg n$ dimensioni e rendendoli lineramente separabili in questo nuovo spazio.

Definizione C.3 (SVM Non Lineare)**.** La soluzione del seguente problema di minimizzazione rappresenta una svm non lineare.

$$min_{\mathbf{w}}\ \tfrac{1}{2}\|\mathbf{w}\|^2$$

$$\text{ristretto a} \quad y_i(\mathbf{w}\cdot\Phi(\mathbf{x_i})+b) \geq 1 \quad i=1...N$$

Data l'analogia evidente tra le svm lineari e le svm non lineari è chiaro che la risoluzione di quest'ultime procede passo passo quanto visto nelle sezioni precedenti, dove basta semplicemente sostituire $\mathbf{x_i}$ con $\Phi(\mathbf{x_i})$:

$$\text{sign}(\mathbf{w} \cdot \Phi(\mathbf{z}) + b) = \text{sign}\left(\sum_{i=1}^{N} \lambda_i y_i \Phi(\mathbf{x_i}) \cdot \Phi(\mathbf{z}) + b\right) \tag{C.15}$$

Riamangono aperte due questioni : come segliere la funzione Φ e come ridurre gli effetti negativi che possono essere introdotti dal fatto di lavorare in spazi m-dimensionali com m molto grande (*Curse of dimensionality problem*).

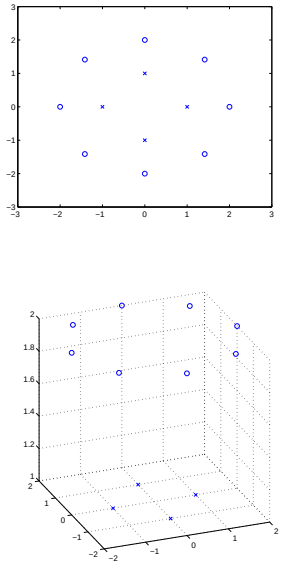


Figura C.1: Esempio di linearizzazione tramite la funzione $(x_1, x_2, \sqrt{x_1^2 + x_2^2})$.

C.2.4 Kernel Trick

Come di vede da (C.15) la classificazione di un elemento richiede il calcolo del prodotto interno di due vettori in uno spazio di grande dimensionalità. Questo fatto comporta alcuni svantaggi :

1. il calcolo risulta essere più oneroso ;
2. la costruzione della svm richiede più tempo ;
3. difficoltà a trovare la funzione Φ da utilizzare ;

4. si può incorrere nel Curse of dimensionality problem.

Per limitare questi problemi si utilizza il così detto "Kernel Trick" : una tecnica che permette di effettuare tutti i calcoli direttamente nell'iperspazio originale, eliminando l'utilizzo della funzione Φ. L'idea fondametale è che i termini del tipo $\Phi(\mathbf{u}) \cdot \Phi(\mathbf{v})$ rappresentano una misura di similitudine tra i vettori $\Phi(\mathbf{u})$ e $\Phi(\mathbf{v})$, ma poichè i vettori $\Phi(\mathbf{x})$ sono ottenuti dai dati originali allora è possibile calcolare tali similitudi direttamente dai vettori dello spazio originale mediante una funzione K detta *Kernel* :

$$\Phi(\mathbf{u}) \cdot \Phi(\mathbf{u}) = K(\mathbf{u}, \mathbf{v}) \tag{C.16}$$

Nella letteratura esistono alcuni kernel utilizzati frequentemente, che sono :

$$\begin{aligned} K(\mathbf{x}, \mathbf{y}) &= (\mathbf{x} \cdot \mathbf{y} + 1)^p \\ K(\mathbf{x}, \mathbf{y}) &= \exp^{-\|\mathbf{x}-\mathbf{y}\|^2/2\sigma^2} \\ K(\mathbf{x}, \mathbf{y}) &= \tanh\left(k\ \mathbf{x} \cdot \mathbf{y} - \delta\right) \end{aligned}$$

C.2.5 Ricerca parametri

Come visto la costruzione di una svm implica la scelta di alcuni parametri : il parametro di costo C nel caso di svm soft margin e di parametri ulteriori nel caso si utilizzi il kernel trick. Solitamente non esistono delle regole fisse e/o formule chiuse per la scelta di tali parametri e l'unico modo è di andare per tentativi, utilizzando una crescita esponenziale dei parametri. Ad esempio, nella costruzione del classificatore mediante svm lineare con soft margin si è proceduto utilizzando i valori $1, 10, 100, 1000, 5000, 10000$.

Bibliografia

[1] Marco Zanchetta *Progettazione di un Service Recommender System basato su Intelligenza Collettiva*, Università degli studi di Padova, 2011

[2] Tomas Olsson *Decentralised social filtering based on trust*, In working Notes of AAAI-98 Recommender System Workshop, 1998

[3] S. Benvenuti *Data aggregation and content refining system* Università degli studi di Genova, 2010

[4] Luca Bianco *Recommendation system basato su social network*, Università degli studi di Padova, 2012.

[5] Eustache Diemert, Gilles Vandelle *Unsupervised Query Categorization using Automatically-Built Concept Graphs* 2009

[6] Dou Shen, Rong Pan, Jian-Tao Sun, Jefrey Junfeng Pan, Kangheng Wu, Jie Yin, Qiang Yang *Query Enrichment for Web-query Classification*

[7] Shui-Lung Chuang , Lee-Feng Chien , Hsiao-Tieh Pu *Automatic Subject Categorization of Query Terms for Filtering Sensitive Queries in Multimedia Search*

[8] Hao Chen, Susan Dumais *Bringing order to the web : automatically categorizing search results*, Proceedings of the SIGCHI conference on Human factors in computing systems, 2000

[9] Dou, Jian-Tao Sun, Qiang Yang, Zheng Chen *Building bridges for web query classification*, Proceedings of the 29th annual international ACM SIGIR conference and development in information retrieval, 2006

[10] Oracle Tecnology Network *Java SE 6 HotSpot[tm] Virtual Machine Garbage Collection Tuning* `http://www.oracle.com/technetwork/java/javase/gc-tuning-6-140523.html`

[11] Oracle Sun Microsystem *Memory Management in the Java HotSpot Virtual Machine* `http://java.sun.com/j2se/reference/whitepapers/memorymanagement_whitepaper.pdf`, 2006

[12] David Detlefs, Christine Flood, Steve Heller, Tony Printezis *Garbage-First Garbage Collection*, Proceedings of the 4th international symposium on Memory management, 2004

[13] Stanislaw Osinski, *An Algorithm For Clustering Of Web Search Results* Poznan University of Technology, 2003

[14] Pang-Ning Tan, Micheal Steinbach, Vipin Kuma *Introduction to Data Mining*, Pearson Internation Edition, 2006

[15] Jonathon Shlens *A Tutorial on Principal Component Analysis* , Measurement, 2005

[16] Kurt Bollacker, Colin Evans, Praveen Parintosh, Tim Sturge, Jamie Taylor *Freebase : a collaboratively created graph database for structuring human knowledge*, Proceeding of the 2008 ACM SIGMOD international conference on Management of data, 2008

[17] Debasish Ghosh, Steve Vinoski *Scala and Lift - Functional Recipes for the Web*, IEEE Internet Computing , 2009

[18] David Pollak, Steve Vinosky *A Chat Application in Lift*, IEEE Internet Computing , 2010

[19] Martin Odersky, Philippe Altherr, Vincent Cremet, Iulian Dragos, Gilles Dubochet, Burak Emir, Sean McDirmid, Stéphane Micheloud, Nikolay Mihaylov, Michel Schinz, Erik Stenman, Lex Spoon, Matthias Zenger *An Overview of the Scala Programming Language - second edition*, École Polytechnique Fédérale de Lausanne (EPFL), Technical Report LAMP-REPORT-2006-001

[20] VirtualBox *https://www.virtualbox.org/*

[21] Jetty *http://jetty.codehaus.org/jetty/*

[22] Apache2 *http://httpd.apache.org/*