

**Realizzazione di un filtro antispam**  
**basato su Weka**  
**e analisi delle sue prestazioni**

*Stefano Stabellini*

## Indice

Introduzione.....	3
Il Dataset.....	3
Datasetcreator.....	5
Datamodelcreator.....	5
POPClient.....	6
Performance di spambase.arff.....	7
Performance di un dataset personalizzato.....	8
Conclusioni.....	11
Bibliografia.....	12

## Introduzione

L'obiettivo che mi sono preposto è quello di realizzare tutti gli strumenti necessari per poter utilizzare Weka come sistema alla base di un filtro antispam e, una volta fatto questo, eseguire su di esso alcuni test di performance.

Per fare ciò prima di tutto è stato necessario creare un software in grado di generare un training set nel formato utilizzato da Weka, ovvero arff, a partire da una semplice mailbox. Successivamente ho scritto un programma che da un dataset in arff e un algoritmo di classificazione genera un datamodel, ovvero un classificatore addestrato sul particolare training set, basandosi naturalmente su Weka. Infine ho creato un semplice client POP in grado di utilizzare i datamodel per classificare la posta scaricata dai server al volo. Tutto il software è stato realizzato in Java, per poter sfruttare al meglio gli strumenti messi a disposizione da Weka.

A questo punto ho utilizzato differenti training set, validation set e algoritmi di apprendimento per stressare il filtro sotto diversi aspetti e cercare di capire quale sia il modo migliore per trattare il problema della classificazione dello spam.

## Il Dataset

Solitamente per la classificazione di segmenti di testo si utilizzano dataset caratterizzati da attributi che rappresentano la frequenza di alcune parole chiave nel testo, e la classificazione di mail non fa eccezione. Per aumentare le prestazioni del filtro però è opportuno aggiungere anche alcuni attributi quali la frequenza di caratteri speciali e la lunghezza delle sequenze di caratteri maiuscoli.

Il dataset iniziale su cui mi sono basato è spambase.arff, reperibile all'indirizzo <http://www.hakank.org/weka>. I suoi attributi in formato arff sono i seguenti:

```
@attribute word_freq_make      real
@attribute word_freq_address   real
@attribute word_freq_all       real
@attribute word_freq_3d        real
@attribute word_freq_our       real
@attribute word_freq_over      real
@attribute word_freq_remove    real
@attribute word_freq_internet  real
@attribute word_freq_order     real
@attribute word_freq_mail      real
@attribute word_freq_receive   real
@attribute word_freq_will      real
@attribute word_freq_people    real
@attribute word_freq_report    real
@attribute word_freq_addresses real
@attribute word_freq_free      real
@attribute word_freq_business  real
@attribute word_freq_email     real
@attribute word_freq_you       real
@attribute word_freq_credit    real
@attribute word_freq_your      real
@attribute word_freq_font      real
```

```

@attribute word_freq_000      real
@attribute word_freq_money    real
@attribute word_freq_hp       real
@attribute word_freq_hpl     real
@attribute word_freq_george   real
@attribute word_freq_650     real
@attribute word_freq_lab      real
@attribute word_freq_labs    real
@attribute word_freq_telnet   real
@attribute word_freq_857     real
@attribute word_freq_data     real
@attribute word_freq_415     real
@attribute word_freq_85      real
@attribute word_freq_technology real
@attribute word_freq_1999    real
@attribute word_freq_parts    real
@attribute word_freq_pm       real
@attribute word_freq_direct   real
@attribute word_freq_cs       real
@attribute word_freq_meeting  real
@attribute word_freq_original real
@attribute word_freq_project  real
@attribute word_freq_re       real
@attribute word_freq_edu      real
@attribute word_freq_table    real
@attribute word_freq_conference real
@attribute 'char_freq_;'      real
@attribute 'char_freq_('      real
@attribute 'char_freq['      real
@attribute 'char_freq!'      real
@attribute 'char_freq_$'     real
@attribute 'char_freq_#'     real
@attribute capital_run_length_average real
@attribute capital_run_length_longest real
@attribute capital_run_length_total real
@attribute class {1, 0}

```

Come si può notare, oltre la frequenza di una cinquantina di parole, sono presenti anche le frequenze di “;()!\$#”, la lunghezza della massima sequenza di lettere maiuscole, la lunghezza media delle sequenze di lettere maiuscole e lunghezza totale di tali sequenze. Questi ultimi tre attributi, anche se un po' inusuali, aumentano notevolmente le prestazioni.

## Datasetcreator

Il primo software che ho scritto è stato Datasetcreator, una piccola utility a linea di comando che permette di creare un proprio dataset con attributi scelti dall'utente; tali attributi devono comunque essere frequenze di parole, frequenze di lettere oppure lunghezze di sequenze di caratteri maiuscoli come gli ultimi tre di spambase.arff.

Il programma prende come parametri il file arff con la descrizione di tali attributi e della classe, la directory contenente i messaggi di posta, e un booleano che rappresenta la classe di appartenenza per tutte le istanze, ovvero se si tratta di spam oppure di ham.

```
root@kaball:~/ingegneria/internet# ./DatasetCreator.sh
Usage: java DatasetCreator <arff> <dir containing msgs> <spam> <options>
```

```
where <options> is:
    -d    Dummy Mode
```

```
root@kaball:~/ingegneria/internet#
```

Fig. 1: Screenshot di Datasetcreator

A causa di un piccolo bug nella classe ArffLoader fornita da Weka, utile per leggere il file arff, Datasetcreator non è in grado di scrivere le nuove istanze direttamente sullo stesso file arff fornito in input, ma semplicemente fornisce i risultati in stdout.

Nella directory fornita come input possono essere contenuti uno o più file, tutti però devono essere nel formato standard mailbox unix oppure semplici file di testo contenenti un solo messaggio; il programma li legge tutti uno alla volta e li passa ad una classe chiamata MesFilter implementata con tutte le funzionalità necessarie per calcolare i valori degli attributi per ogni istanza.

## Datamodelcreator

L'utente ora è in grado di creare un dataset a piacimento a partire dalle proprie mail, quindi ha necessità di generare un classificatore, e di effettuare il suo training sul proprio dataset.

A questo scopo ho realizzato il programma Datamodelcreator: esso prende da linea di comando il nome del file contenente il dataset con le istanze, il file di output e l'algoritmo di apprendimento, dopodiché genera il classificatore, effettua il suo training e lo salva sul file. Tutto questo utilizzando pesantemente le classi e le interfacce messe a disposizione da Weka.

L'algoritmo di apprendimenti può essere J48, NaiveBayes, SMO (Support Vector Machine), Ibk, o Jrip: ho scelto questi tra tutti quelli che Weka offre per poter fare delle prove con un po' tutte le diverse categorie di classificatori, ma in realtà è veramente molto semplice aggiungere anche altri algoritmi di Weka da utilizzare. Datamodelcreator è anche in grado di effettuare una classificazione cost-sensitive, ovvero pesando in maniera differente gli errori falsi positivi e i falsi negativi, basta specificare l'opzione -c e un file contenente la matrice dei costi. Nel caso dei filtri antispam ciò risulta particolarmente utile visto che di solito per gli utenti è molto meno grave se una mail di spam viene classificata come buona che non il contrario.

Un esempio di matrice dei costi nel formato utilizzato da Weka è il seguente, in cui si

pone a 1 il costo di un falso negativo e a 2 il costo di un falso positivo:

```
0 1 1
1 0 2
```

la prima cifra è l'indice della reale classe dell'istanza, la seconda l'indice della classe con cui viene classificata l'istanza, la terza il costo dell'errore.

```
root@kaball:~/ingegneria/internet# ./DataModelCreator.sh

Usage: java DataModelCreator <dataset> <algorithm> <datamodel> <options>

where <algorithm> is:
    J48
    NaiveBayes
    SMO
    IBk
    JRip

where <options> is:
    -c <costmatrix file> Cost Sensitive Classifier

root@kaball:~/ingegneria/internet#
```

Fig. 2: screenshot di Datamodelcreator

## POPClient

Creato il classificatore non serve altro che un'applicazione in grado di utilizzarlo. A questo scopo ho realizzato POPClient, un semplice client di posta in grado di scaricare mail da un server tramite il protocollo standard rfc 1939, ovvero il pop, salvare i messaggi su file ed infine stabilire se ognuno di essi è spam oppure no.

POPClient è basato principalmente su tre differenti classi: PopClient, che implementa i semplici comandi Pop di base, PopFileStore che utilizza PopClient per scaricare i messaggi poi successivamente li salva su file in un formato molto simile a quello mailbox unix; ed infine PopGUI, ovvero la classe che contiene l'interfaccia testuale per dialogare con l'utente.

PopGUI riceve da linea di comando, oltre i parametri necessari per connettersi al server remoto, anche il datamodel da utilizzare per la classificazione e un file arff che rappresenta il formato degli attributi, poi li passa a PopFileStore. Quest'ultimo al momento di salvare le mail scaricate su file, genera per ogni messaggio un'istanza utilizzando ancora una volta MesFilter e il file arff, poi determina se è spam tramite il classificatore deserializzato dal datamodel.

Se un messaggio viene giudicato spazzatura allora per ricordarlo PopFileStore scrive "SPAM" sulla mailbox; in questo modo al momento di stampare a video l'elenco dei messaggi ricevuti è in grado di comunicare all'utente se ogni mail è stata giudicata spam dal filtro.

Purtroppo a causa del fatto che la maggior parte degli algoritmi di Weka non sono in grado di continuare l'apprendimento finito il training iniziale, non ho potuto permettere all'utente di addestrare il filtro segnalando personalmente i falsi positivi e i falsi negativi.

In figura 3 è possibile vedere uno screenshot durante il funzionamento di PopClient:

```
root@kaball:~/ingegneria/internet# ./PopGUI.sh
Usage: java PopGUI <host> <user> <pass> <spamdatabase> <arff dataset>
root@kaball:~/ingegneria/internet# ./PopGUI.sh 127.0.0.1 neo thematrix my.J48 my.arff
? get
? list
1 From: root@kaball.darklight.net Data: Mon, 21 Feb 2005 15:33:38 +0000 0 Subject: Cracckare il formato 802.11
2 From: root@kaball.darklight.net Data: Mon, 21 Feb 2005 15:34:41 +0000 0 Subject: Fwd: Incontri e corso Linux a San La
3 From: root@kaball.darklight.net Data: Mon, 21 Feb 2005 15:35:12 +0000 0 Subject: Mount all'avvio di KDE
4 From: root@kaball.darklight.net Data: Mon, 21 Feb 2005 15:35:43 +0000 0 Subject: Package nagios-1.2
5 From: root@kaball.darklight.net Data: Mon, 21 Feb 2005 15:36:08 +0000 0 Subject: [AppDB] Warning: inactivity detected
6 From: root@kaball.darklight.net Data: Mon, 21 Feb 2005 15:36:36 +0000 1 Subject: Re: Postfix Slackware package
7 From: root@kaball.darklight.net Data: Mon, 21 Feb 2005 15:38:19 +0000 0 Subject: Refill Order Shipped at Thu, 17 Feb 2
8 From: root@kaball.darklight.net Data: Mon, 21 Feb 2005 15:38:38 +0000 1 Subject: Doct'or discovers s,perm increasement
9 From: root@kaball.darklight.net Data: Mon, 21 Feb 2005 15:39:01 +0000 0 Subject: Better than the real thing
10 From: root@kaball.darklight.net Data: Mon, 21 Feb 2005 15:39:22 +0000 0 Subject: Popular Soft
? □
```

Fig. 3: screenshot di PopClient

## Performance di spambase.arff

I primi test li ho eseguiti su spambase.arff: prima ho provato ad utilizzare algoritmi differenti e mi sono basato sulla cross-validation direttamente sul training set come misura della performance, poi ho provato ad aggiungere una matrice dei costi come quella vista in precedenza stimando un falso negativo a 1 e i falsi positivi prima a 2, poi a 5 ed infine a 10. I risultati sono stati i seguenti:

	No CostMatrix	CostSensitive: 2	CostSensitive: 5	CostSensitive: 10
Ibk	90.76			
NaiveBayes	79.29	79.46	79.55	79.59
J48	92.98	93.2	90.89	87.74
Jrip	92.39	92.68	90.55	88.31
SMO	90.44	86.57	78.72	70.42

I valori riportati corrispondono alla percentuale di istanze classificate correttamente sul totale; ho effettuato inoltre solo un test con l'algoritmo Ibk per l'alto costo in termini di tempo richiesto da tale algoritmo per eseguire la classificazione, certamente inadeguato per essere utilizzato al volo in un client di posta.

Come è evidente dai risultati introdurre una matrice dei costi generalmente peggiora la performance globale se il costo degli errori diventa troppo sbilanciato, anche se naturalmente continua sempre a diminuire il numero di falsi positivi. Così ho deciso di effettuare tutti i successivi test stimando a 2 un falso positivo, poiché risulta essere decisamente il miglior compromesso possibile.

A questo punto ho ritestato spambase.arff con gli stessi algoritmi di prima, ma utilizzando anche un validation test costruito ad hoc tramite Datasetcreator: come mail pulite ho utilizzato la mia mailbox personale, mentre per lo spam ho scaricato un grosso archivio da <http://www.spamarchive.org>. Ho ottenuto in questo modo un validation set di oltre 2000 istanze, approssimativamente metà positive e metà negative, i risultati sono stati i seguenti:

	Training set	Validation set
NaiveBayes	79.63	63.08
J48	96.46	61.69
SMO	86.81	41.35
Jrip	94.39	55.53

Sono presenti a fianco anche le percentuali di istanze del training set classificate correttamente per poter apprezzare meglio il peggioramento di prestazioni degli algoritmi.

I risultati dimostrano che le stime teoriche della performance degli algoritmi effettuate direttamente sul training set possono essere davvero troppo ottimistiche, visto che i primi tre algoritmi del test precedente hanno avuto tutti un calo di almeno il 30%. Senza dubbio NaiveBayes è quello che si comporta meglio: sul training set è l'ultimo con un 79% di istanze classificate correttamente, mentre sul validation set è primo con un calo di solo un 16%.

## Performance di un dataset personalizzato

Effettuati questi test ho provato a costruire un mio dataset personalizzato, scegliendo come attributi una cinquantina di parole (perlopiù diverse da quelle di spambase), sei caratteri speciali e mantenendo sempre i tre attributi che rappresentano la lunghezza della massima sequenza di lettere maiuscole, la lunghezza media delle sequenze di lettere maiuscole e lunghezza totale di tali sequenze. Gli attributi in formato arff sono i seguenti:

```
@attribute word_freq_adult numeric
@attribute word_freq_address numeric
@attribute word_freq_credit numeric
@attribute word_freq_soft numeric
@attribute word_freq_refill numeric
@attribute word_freq_order numeric
@attribute word_freq_discount numeric
@attribute word_freq_drugs numeric
@attribute word_freq_sperm numeric
@attribute word_freq_viagra numeric
@attribute word_freq_doctor numeric
@attribute word_freq_now numeric
@attribute word_freq_price numeric
@attribute word_freq_retails numeric
@attribute word_freq_shipping numeric
@attribute word_freq_ship numeric
@attribute word_freq_today numeric
@attribute word_freq_free numeric
@attribute word_freq_prescription numeric
@attribute word_freq_online numeric
@attribute word_freq_pill numeric
@attribute word_freq_cheap numeric
@attribute word_freq_alert numeric
```

```

@attribute word_freq_money numeric
@attribute word_freq_24 numeric
@attribute word_freq_hours numeric
@attribute word_freq_wealth numeric
@attribute word_freq_internet numeric
@attribute word_freq_opportunity numeric
@attribute word_freq_waste numeric
@attribute word_freq_site numeric
@attribute word_freq_sex numeric
@attribute word_freq_meet numeric
@attribute word_freq_loss numeric
@attribute word_freq_cash numeric
@attribute word_freq_offer numeric
@attribute word_freq_special numeric
@attribute word_freq_extra numeric
@attribute word_freq_brand-new numeric
@attribute word_freq_contact numeric
@attribute word_freq_business numeric
@attribute word_freq_securiy numeric
@attribute word_freq_bill numeric
@attribute word_freq_sorry numeric
@attribute word_freq_porn numeric
@attribute word_freq_interest numeric
@attribute word_freq_tax numeric
@attribute word_freq_make numeric
@attribute word_freq_better numeric
@attribute word_freq_best numeric
@attribute char_freq_ ; numeric
@attribute 'char_freq_,' numeric
@attribute char_freq_ ` numeric
@attribute char_freq_! numeric
@attribute char_freq_$ numeric
@attribute char_freq_# numeric
@attribute capital_run_length_average numeric
@attribute capital_run_length_longest numeric
@attribute capital_run_length_total numeric
@attribute class {1,0}

```

Dopodiché ho popolato il dataset utilizzando sempre la mia mailbox personale per le mail pulite e l'archivio di spamarchive.org per lo spam; in questo modo ho ottenuto un dataset di poco più di 10000 istanze, sempre uniformemente distribuite tra spam e ham. Ho creato subito anche un validation set di 2000 elementi per effettuare test più significativi, i cui risultati sono stati i seguenti:

	Training set	Validation set
NaiveBayes	44.67	53.28
J48	89.19	82.56
SMO	43.34	53.14
Jrip	81.37	81.38

Confrontandoli con quelli ottenuti con spambase.arff, si può notare che i valori sono tutti generalmente più bassi sul training set, ma calano molto meno passando al validation set.

In particolare NaiveBayes e SMO ottengono solo un 44% sul primo, ma addirittura aumentano la loro performance sul secondo passando entrambi a un 53% circa. Jrip sostanzialmente ottiene lo stesso risultato su tutti e due, un ottimo 81%, mentre l'unico a calare è J48, che perde 7 punti percentuali ma rimane comunque l'algoritmo con la performance migliore di tutti.

Rispetto ai rispettivi valori ottenuti da spambase sul validation set, quelli ottenuti dal mio dataset sono tutti superiori tranne il 53% di NaiveBayes.

Ciò dimostra che un'accurata scelta degli attributi del dataset è almeno importante quanto la scelta dell'algoritmo di classificazione delle istanze. Differenze così elevate in termini di prestazioni tra il mio dataset e spambase.arff sono anche dovute probabilmente al fatto che le mail che ho usato per il training e il validation set sono risultate più simili nel mio caso rispetto a spambase.

Per effettuare un ultimo test ho infine creato un altro training set, sempre con gli stessi attributi del precedente ma popolato con più di 25000 elementi, la maggior parte dei quali costituito da esempi di spam. Per provarne l'efficienza con i vari algoritmi ho anche realizzato un ulteriore validation set di 7000 elementi, anche in questo caso per la maggior parte spam; ecco i risultati:

	Training set	Validation set
NaiveBayes	32.61	20.43
J48	92.28	82.42
SMO	83.61	98.65
Jrip	82.18	77.59

NaiveBayes peggiora sensibilmente le sue prestazioni, J48 ottiene un leggero miglioramento sul training set ma gli stessi risultati sul validation set; su quest'ultimo invece Jrip risulta leggermente peggiorato.

Il caso che si discosta di più dai risultati precedenti è quello di SMO che passa da un 40% di istanze classificate correttamente sul training set nel test precedente ad un 80% in questo, e soprattutto sul validation set incrementa ancora di più le sue prestazioni arrivando addirittura al 98%.

Quest'ultimo algoritmo risulta essere certamente quello più sensibile alle variazioni dei dati di ingresso e della matrice dei costi, dato che nei vari test ha avuto le maggiori discrepanze sui risultati; in quest'ultima prova probabilmente è stato anche favorito dallo sbilanciamento delle classi delle istanze nel training e nel validation set.

Caso SMO a parte, si può desumere dai risultati che a un decisivo aumento delle istanze del training set non ci si deve aspettare un corrispettivo miglioramento delle prestazioni degli algoritmi, anzi, in alcuni casi si ottengono solo peggiori risultati.

## Conclusioni

Il problema della classificazione dello spam è molto sentito sia dagli utenti che dagli sviluppatori: moltissime soluzioni in ambito commerciale e open source sono state proposte negli ultimi anni, ma siamo ancora molto lontani da una soluzione definitiva, anche perché ad ogni miglioramento delle tecniche di filtraggio delle mail, di solito c'è un corrispettivo miglioramento delle tecniche di spamming.

Lo sviluppo e l'affermazione di potenti sistemi open source di apprendimento automatico come Weka offrono un'opportunità che non può venire trascurata da chi opera in questo settore. Con questa ricerca ho cercato di realizzare gli strumenti necessari per rendere direttamente interoperabili Weka e qualsivoglia software scritto in Java che necessiti di filtrare lo spam. I risultati dimostrano che la flessibilità che si può raggiungere in questo modo è davvero molto elevata: a partire dallo stesso dataset si possono costruire con molta semplicità decine di classificatori differenti, e il software che li va ad utilizzare li può scegliere e cambiare a piacimento.

In questo modo risulta anche molto facile eseguire test di performance, i quali hanno dimostrato che una scelta accurata del dataset è molto importante, che aumentare a sproposito il numero di istanze del training set non migliora di certo la situazione e che gli algoritmi che hanno buoni risultati sul training set possono peggiorare moltissimo sul validation set.

Sicuramente non esiste un unico algoritmo e un unico dataset che si comportano al meglio in tutte le condizioni in quanto caso per caso le differenze possono essere davvero rilevanti, e inoltre ogni soluzione ha i propri pro e contro ed il suo impiego va valutato di volta in volta in base alle caratteristiche della situazione.

Per esempio l'algoritmo NaiveBayes, che spesso non si è dimostrato particolarmente brillante, in realtà nel caso di utilizzo di un semplice client di posta quale Thunderbird, potrebbe essere il migliore grazie alla sua capacità di continuare ad apprendere anche durante il suo utilizzo. Se invece si cerca la prestazione migliore assoluta, allora dopo aver costruito in maniera particolarmente accurata un dataset, probabilmente si vorrà utilizzare J48 oppure SMO.

Il grosso vantaggio che si ottiene nell'utilizzare un potente engine come Weka alla base della propria applicazione sta proprio in questa grande libertà di scelta che esso conferisce agli sviluppatori degli strati di software sovrastanti.

## **Bibliografia**

Thorsten Joachims – Learning to classify text using support vector machines

Ian H. Witten, Eibe Frank – WEKA Machine Learning Algorithm in Java

Weka - <http://www.cs.waikato.ac.nz/~ml>

Spam Archives - <http://www.spambase.org>

spambase.arff - <http://www.hakank.org/weka>

JavaMail - <http://www.sun.com>