Università degli Studi di Padova
Dipartimento di Scienze Statistiche
Corso di Laurea Magistrale in
Scienze Statistiche

Strategie di Sentiment Analysis: confronti e nuove proposte.

Relatore: Prof. Livio Finos

Correlatore: Dott. Dario Solari

Laureanda: Maria Maddalena Branca
Matricola N 1013822

Anno Accademico 2013/2014

La felicità consiste nel non porsi mai il problema di misurarla
o di chiedersi se si è soddisfatti o meno.
-George Bernard Shaw-

Indice

Introduzione **6**

1 Raccolta e preprocessing dei dati **9**

1.1 Data-set 9

1.2 Fase di *Preprocessing* del testo 10

1.2.1 L'analizzatore lessicale 11

1.2.2 Stemming 12

1.3 Applicazione 13

2 Algoritmo basato sui dizionari ontologici **17**

2.1 I dizionari 17

2.2 L'algoritmo 18

2.3 Applicazione e Risultati 18

2.3.1 Analisi descrittive 19

3 Tree-Based Model per la Sentiment Analysis **23**

3.1 Data-set 23

3.2 Tree-Based Models 24

3.3 Analisi con il pacchetto Rpart 25

3.3.1 Analisi con il pacchetto `Rpart` integrando i risultati dell'algoritmo basato sui dizionari ontologici 27

4 La regressione multinomiale inversa (MNIR) **31**

4.1 Il metodo di Taddy 31

4.1.1 Regressione Inversa e Riduzione Sufficiente 32

4.1.2 Il modello 32
4.2 Analisi con il pacchetto `textir` 35
4.2.1 Analisi con il pacchetto `textir` integrando i risultati dell'algoritmo basato sui dizionari ontologici 38

5 Il metodo di Hopkings e King **43**
5.1 Il metodo 43
5.1.1 Classificazione aggregata 44
5.2 Analisi e valutazione dei risultati 45
5.3 Analisi con il pacchetto `ReadMe` 46
5.3.1 Analisi con il pacchetto `ReadMe` integrando i risultati dell'algoritmo 48

6 Vettori di probabilità **51**
6.1 Vettori di probabilità con il pacchetto `Rpart` 52
6.2 Vettori di probabilità con il pacchetto `textir` 53
6.3 Confronto e commenti 54
6.4 Vettori di probabilità cambiando le vere proporzioni nell'insieme di verifica 55

7 Esempio di applicazione ad un data-set sul tema dell'immigrazione **57**

Conclusioni **59**

Bibliografia **63**

Introduzione

Il crescente utilizzo di internet e l'avvento dei social media ha attirato l'interesse di aziende e scienziati sociali ad indagare e monitorare le opinioni di chi naviga nel web. In particolare, i social network hanno cambiato il modo in cui le aziende comunicano con i clienti e la proliferazione di recensioni, voti e consigli ha reso indispensabile il monitoraggio costante di queste informazioni, mentre dal punto di vista della ricerca sociale essi ci consentono di cogliere opinioni / emozioni / sentimenti liberamente espressi e non sollecitati dalle domande di un questionario. Ormai gran parte della raccolta pubblicitaria, del mondo della comunicazione e addirittura parte delle campagne elettorali si è orientata verso la rete. Questo saper "spiare" quello che pensano gli utenti è lo scopo di una disciplina che negli ultimi anni ha suscitato particolare interesse: la Sentiment Analysis. Ci si soffermerà sull'analisi di post reperiti da *Twitter*, che è un social network aperto (non bisogna essere utenti iscritti per reperire l'informazione) e dalla comunicazione incisiva. Infatti la caratteristica di Twitter è la possibilità di esprimersi in un messaggio di soli 140 caratteri, chiamato *tweet*.

Le prime tecniche di Sentiment Analysis basate sul conteggio delle parole, come quella dei dizionari ontologici, sono state sviluppate concentrandosi sulla lingua inglese. Quindi il primo passo che è stato fatto è costruire un algoritmo basato su dizionari ontologici in lingua italiana creati ad hoc. Tale algoritmo consta di più passi che portano a semplificare e ridurre il testo a dato quantitativo, assegnandogli un punteggio positivo o negativo quando c'è corrispondenza rispettivamente nel dizionario di parole positive o di parole negative.

Ovviamente esistono altri strumenti sviluppati più o meno recentemente,

in questo lavoro di tesi sono stati messi a confronto alcuni di tipo statistico, per capire se ad oggi esiste uno strumento valido e solido per analizzare un testo. Tale compito resta infatti arduo e ostacolato, in quanto è difficile per un calcolatore interpretare frasi del linguaggio quotidiano, l'ironia, il sarcasmo, le citazioni, le allusioni e le metafore.

Le tecniche su cui ci soffermeremo sono gli alberi di classificazione, la regressione multinomiale inversa (Matt Taddy, 2013) e il metodo ReadMe (Hopkins-King, 2010). L'idea di base è stata quella di confrontare tali metodi con gli stessi integrati con i risultati della classificazione dell'algoritmo basato sui dizionari ontologici. Si è voluto verificare infatti se tale aggiunta può aumentarne l'accuratezza e l'efficacia.

Il punto di partenza per i vari confronti delle diverse tecniche è stato un data-set di 2818 tweets che non trattano un argomento specifico, classificati manualmente in tre categorie: "pos" i positivi, "neg" i negativi e "neutro" i neutri. Un secondo confronto è stato fatto successivamente su un data-set di tweets riguardanti la percezione dell'immigrazione in Italia, per avere un esempio più realistico in quanto nella pratica chi si approccia alla Sentiment Analysis generalmente lo fa perchè vuole analizzare l'opinione per un determinato argomento.

Su questi data-set sono state effettuate due tipi di analisi: in un primo momento ci si è soffermati sulla classificazione dei singoli tweets, mentre in un secondo momento si è voluto confrontare i vettori di probabilità delle tre classi, individuando quindi l'andamento generale delle polarità nella totalità dei tweets. Si vuole verificare se le stime basate sui vettori risultino più accurate rispetto alle stime basate sulle proporzioni di classificati nelle tre categorie di sentiment.

Capitolo 1

Raccolta e preprocessing dei dati

Le prime tecniche di Sentiment Analysis basate sul conteggio delle parole, come quella dei dizionari ontologici, sono state sviluppate concentrandosi sulla lingua inglese. Quindi il primo passo che è stato fatto è costruire un algoritmo basato su dizionari ontologici in lingua italiana creati ad hoc. Tale algoritmo consta di più passi che portano a semplificare e ridurre il testo a dato quantitativo, in questo capitolo ci si concentrerà sulla prima fase, quella di preprocessing e semplificazione del testo.

1.1 Data-set

Per le analisi effettuate in questo lavoro di tesi è stato utilizzato un data-set di 2818 tweets raccolti nel mese di febbraio e nel mese di luglio del 2014. Il reperimento di questi tweets è stato effettuato tramite `dump_tweets.R` [10], che è un tool di R per acquisire i tweets di interesse. I dati vengono quindi salvati in un database MySQL per poi essere esportati con estensione .RData e manipolati con R. In fase di reperimento questo strumento permette di specificare alcune caratteristiche, come la lingua, la posizione geografica, il numero di tweets e una o più parole chiave. I tweets raccolti sono in lingua italiana e non trattano di un tema specifico, ma sono vari e casuali.

Una volta reperiti, questi 2818 tweets sono stati poi classificati manualmente in tre categorie: “pos” i positivi, “neg” i negativi e “neutro” i neutri. La classificazione manuale è avvenuta in due fasi. Una prima porzione di dati è stata classificata dagli studenti del corso di ”classificazione e analisi dei dati multidimensionali (AA 13/14)“ del professor Finos presso il Dipartimento di Scienze Statistiche. La seconda è stata classificata dal prof. Livio Finos, il dott. Dario Solari, dott. Duccio Schiavon, il dott. Andrea Sciandra, il dott. Andrea Zedda e da me. Nel corso del lavoro questa classificazione verrà chiamata per praticità ”vera classificazione“, anche se soggettività e differente interpretazione di uno stesso testo da parte di più individui sono le cause principali di una mancata classificazione oggettiva e chiara per tutti, che invece sarà soggetta ad errore. A prova di ciò è stata fatta una valutazione dei tweets che si ripetevano e che sono stati classificati da soggetti differenti o a volte addirittura dalla stessa persona, e circa il 20% non viene classificato nello stesso modo. Questo ad evidenziare che la classificazione manuale è soggetta a tanti problemi ed imprecisioni che rendono il nostro lavoro ancora più arduo.

1.2 Fase di *Preprocessing* del testo

La fase di analisi statistica del testo è preceduta da un’analisi lessicale che consiste nel rilevare ed estrarre i *token* (fase di indicizzazione del testo), ossia le potenziali parole chiave che verranno poi utilizzate come predittori nelle nostre analisi. Lo scopo è quindi di ridurre il testo a dato quantitativo. A tal fine il testo viene scansionato dall’algoritmo di scansione del testo, detto *analizzatore lessicale*, che dipende dalla lingua perché le parole sono scritte con alfabeti diversi e possono esserci diversi caratteri di separazione. Esso deve tenere conto della varietà di modi per esprimere la stessa informazione. In letteratura è difficile trovare delle applicazioni all’italiano di tali algoritmi, per questo ne è stato creato uno ad hoc in R. `TextWiller` [11] è il pacchetto R progettato a tale scopo e che verrà utilizzato nelle applicazioni successive.

1.2.1 L'analizzatore lessicale

Per prima cosa l'analizzatore lessicale riconosce ed elimina quelle che sono le *stop words* nel testo, ossia quelle parole poco o per niente specifiche che non forniscono informazione e che compaiono troppo frequentemente o troppo raramente, ad esempio congiunzioni, articoli, preposizioni e avverbi. È stata creata una *stop list*, chiamata `itastopwords` e contenuta in `TextWiller`, a partire da quella già esistente nel pacchetto `snowball` di `R` [1]. Un'idea per migliorare il risultato potrebbe essere quella di creare delle stop list alternative più specifiche, da affiancare a questa più generale già esistente e da scegliere in base all'argomento trattato. Si può notare infatti che un token apporta informazione condizionatamente al contesto, quindi un testo che tratta ad esempio di medicina sarebbe meglio analizzarlo con una stop list specifica, possibilmente curata da personale specializzato.

Successivamente si procede con il riconoscimento delle *emoticon*, fondamentali per quanto riguarda il nostro obiettivo finale di categorizzare i tweet con una polarità. Spesso infatti le emoticon chiariscono il significato dei tweet e disambiguano quelli sarcastici o ironici. É stata stilata una lista di emoticon, l'analizzatore scorre tale lista, e quando riconosce l'emoticon nel testo, la sostituisce con una stringa che la identifica. Ad esempio le emoticon che rappresentano felicità vengono sostituite dalla stringa `EMOTEGOOD`, invece quelle che rappresentano il pianto con la stringa `EMOTECRY`. La funzione di `TextWiller` che esegue tale operazione è `normalizzaemote`.

Con la funzione `normalizzaslang` vengono identificati *slang* e modi di dire tipici dell'ambiente social, che vengono poi sostituiti con le parole originarie o con delle stringhe alternative di cui si terrà conto in fase di assegnazione della polarità.

Infine si procede all'eliminazione di spazi, siti web e punteggiatura (`normalizzahtml` e `normalizzapunteggiatura`). Per quest'ultimo aspetto l'algoritmo tiene traccia della numerosità dei seguenti simboli per ciascun tweet: ?, !, @, #, €, $. In particolare si sottolinea la presenza del simbolo # che su twitter precede un *hashtag*, ossia una vera e propria etichetta formata da una o più parole concatenate. É evidente l'importanza che rivestirebbero

gli hashtag nell'individuazione della polarità, purtroppo però la concatenazione di più parole senza spazi impedisce di fatto la possibilità di tenerne conto se non con un elevatissimo onere computazionale. Se si potesse trovare però una soluzione a questo problema, probabilmente si otterrebbero notevoli vantaggi e miglioramenti ai fini della sentiment analysis. La ricerca di una soluzione a questo problema potrebbe essere oggetto di lavori futuri.

1.2.2 Stemming

Dopo il processo dell'analizzatore lessicale, quello che si è ottenuto è un insieme di parole senza un ordine preciso detto "bag of words". Per semplificare ulteriormente il procedimento successivo, l'algoritmo provvede alla riduzione di tali parole alla loro radice fondamentale, detta tema. Il tema non corrisponde necessariamente alla radice morfologica (lemma) della parola: normalmente è sufficiente che le parole correlate siano mappate allo stesso tema, anche se quest'ultimo non è una valida radice per la parola.[1] Tale riduzione si chiama *stemming*, il tema ottenuto è detto *stem* e lo strumento che si è utilizzato è lo *stemmer*.

Quello che ci aspettiamo da un buono stemmer è che riconosca (con la maggiore accuratezza possibile) la correlazione tra termini a cui attribuiamo una semantica comune, e sfrutti questa correlazione per sostituire tutti i termini correlati con il tema corrispondente. Ad esempio: 'gatto', 'gatta', 'gattino', 'gattaccio', ecc. li vorremmo tutti mappati sul tema 'gatto', o 'gatt'.

Questo procedimento è, insieme alla fase di analisi vista precedentemente, essenziale per semplificare la complessità computazionale che comporterebbe l'utilizzo di tutti i token presenti nei post, incrociati con tutti i termini italiani (tra i 215.00 e i 217.000 [2]), unificando quindi le parole riconducibili al medesimo concetto. Le tecniche di stemming sono studiate in informatica da ben 40 anni e sono uno dei metodi di base per ridurre la dimensionalità dei documenti di testo.[5] Nel 1980 Martin Porter [9] pubblica uno stem-

[1] Wikipedia, http://it.wikipedia.org/wiki/Stemming

[2] Treccani,http://www.treccani.it/magazine/lingua_italiana/domande_e_risposte/varie/varie_026.html

mer considerato tra i più raffinati, che si è imposto come metodo standard di stemming nell'Inglese, ed è distribuito liberamente nei vari linguaggi di programmazione.

Per la lingua italiana abbiamo usato uno stemmer preesistente in `R`, implementato nel pacchetto `SnowballC` [8].

Come ogni semplificazione, anche lo stemming presenta dei problemi:

1. in caso di multilinguismo e di parole in una lingua diversa da quella trattata dall'algoritmo di stemming;

2. in caso di parole riconducibili al medesimo stem ma con significati completamente diversi (vedi *bravo* e *bravata* hanno entrambi lo stem *brav*).

Lo stemming verrà applicato nell'algoritmo basato sui dizionari ontologici, ma sarebbe interessante in lavori futuri estenderne l'uso anche per i modelli statistici.

1.3 Applicazione

Riprendiamo le varie funzioni esplicate fin'ora e vediamole nel dettaglio con degli esempi presi dal nostro data-set.

```
> library(TextWiller)
> prova

[1] Bravi Ennio e Massimo! :) http://t.co/v5YUsoY9Wp
[2] @Grgre83_1 buonanotte col ballo dei matti. tie e tac :-) http://t.co/4aFh8uytBt
[3] @_soproud_  Lo faresti per favore? Grazie mille se lo fai :) https://t.co/ozfdRM
[4] Notizia riportata da tutti i Tg... R.I.P. :( http://t.co/AgQjvho4S7
[5] insulti a vittime delle foibe: oggi, #10febbraio 2014, su facebook. :( @il_picco
[6] @esenzioneticket lol lo stesso che ha messo insieme me e l'eternauta. pensa che
2787 Levels: :( @05123987 esatto :) ... '@zulidahannibal: giugno '95 Hip Hop Village

> normalizzahtml(prova)
```

```
[1] "Bravi Ennio e Massimo! :) WWWURLWWW "
[2] "@Grgre83_1 buonanotte col ballo dei matti. tie e tac :-) WWWURLWWW "
[3] "@_soproud_ Lo faresti per favore? Grazie mille se lo fai :) WWWURLWWW "
[4] "Notizia riportata da tutti i Tg... R.I.P. :( WWWURLWWW "
[5] "insulti a vittime delle foibe: oggi, #10febbraio 2014, su facebook. :( @
[6] "@esenzioneticket lol lo stesso che ha messo insieme me e l'eternauta. pe
```

```
> normalizzaemote(prova)
```

```
[1] "Bravi Ennio e Massimo! EMOTEGOOD http://t.co/v5YUsoY9Wp"
[2] "@Grgre83_1 buonanotte col ballo dei matti. tie e tac EMOTEGOOD http://t.
[3] "@_soproud_ Lo faresti per favore? Grazie mille se lo fai EMOTEGOOD https
[4] "Notizia riportata da tutti i Tg... R.I.P. EMOTEBAD http://t.co/AgQjvho4S
[5] "insulti a vittime delle foibe: oggi, #10febbraio 2014, su facebook. EMOT
[6] "@esenzioneticket lol lo stesso che ha messo insieme me e l'eternauta. pe
```

```
> normalizzaslang(prova)
```

```
[1] "Bravi Ennio e Massimo! :) http://t.co/v5YUsoY9Wp"
[2] "@Grgre83_1 buonanotte col ballo dei matti. tie e tac :-) http://t.co/4aF
[3] "@_soproud_  Lo faresti per favore? Grazie mille se lo fai :) https://t.c
[4] "Notizia riportata da tutti i Tg... R.I.P. :( http://t.co/AgQjvho4S7"
[5] "insulti a vittime delle foibe: oggi, #10febbraio 2014, su facebook. :( @
[6] "@esenzioneticket EMOTELOL lo stesso che ha messo insieme me e l'eternaut
```

In `TextWiller` tutte queste operazioni, così come le altre che vengono applicate ai fini di normalizzare il testo prima di sottoporlo all'algoritmo, vengono richiamate dalla funzione `normalizzaTesti()` che restituisce la lista dei tweets normalizzati e i conteggi dei vari simboli di cui si vuole tener traccia.

```
> normalizzaTesti(as.character(prova))
```

```
[1] "bravo ennio e massimo EMOTEGOOD wwwurlwww"
[2] "RT @grgre83 1 buonanotte col ballo dei matti tie e tac EMOTEGOOD wwwurlw
```

```
[3] "RT @ soproud lo faresti per favore grazie mille se lo fai EMOTEGOOD wwwurlwww"
[4] "notizia riportata da tutti i tg r i p EMOTEBAD wwwurlwww"
[5] "insulti a vittime delle foibe oggi #10febbraio 2014 su facebook EMOTEBAD @il pi
[6] "RT @esenzioneticket EMOTELOL lo stesso che ha messo insieme me e l eternauta pe
attr(,"counts")
     Conteggi.\\? Conteggi.\\! Conteggi.@ Conteggi.# Conteggi.(€|euro) Conteggi.(\\$
[1,]            0            1          0          0                 0
[2,]            0            0          1          0                 0
[3,]            1            0          1          0                 0
[4,]            0            0          0          0                 0
[5,]            0            0          3          1                 0
[6,]            0            0          1          0                 0
     Conteggi.SUPPRESSEDTEXT
[1,]                       0
[2,]                       0
[3,]                       0
[4,]                       0
[5,]                       0
[6,]                       0
```

Capitolo 2

Algoritmo basato sui dizionari ontologici

Dopo la prima fase dell'algoritmo, che si concentra appunto sulla pulizia del testo, si passa alla fase di assegnazione di polarità. Però prima di procedere a tale fase, si sono definiti i dizionari ontologici, più precisamente due liste di parole italiane che esprimono una polarità, che può essere positiva o negativa.

2.1 I dizionari

Il punto di partenza per la creazione dei dizionari ontologici italiani sono stati i dizionari inglesi utilizzati da Hu & Liu (2004) in una sentiment analysis su recensioni di prodotti. Il primo dizionario raccoglie circa 2000 parole "positive", mentre quello con le parole "negative" ne contiene circa 4800. È stata quindi effettuata la traduzione delle parole e si è proceduto con il miglioramento e l'arricchimento delle liste. Alcuni dei termini presenti nei dizionari, tuttavia, nella nostra analisi assumono una valenza differente, poiché il loro contesto applicativo è diverso; alcuni sono quindi stati rivalutati, altri eliminati. Ai dizionari sono stati aggiunti anche le stringhe corrispondenti alle emoticon e agli slang, individuati nel capitolo precedente. Infine su tutti i dizionari è stato applicato lo stemmer di Porter, e sono stati eliminati gli stem

doppi. In conclusione ci ritroviamo due liste di stem "positivi" e "negativi" rispettivamente di 980 e 2304 elementi.

La fase di preparazione di questi dizionari è fondamentale per ottimizzare l'accuratezza del risultato, ma è ovvio che i casi di ironia o sarcasmo, le citazioni, le allusioni e le metafore sono di difficile interpretazione in un metodo automatizzato. Includere le emoticon e le parole colloquiali (slang) ci può essere d'aiuto, ma volendo migliorare il risultato si potrebbe procedere in altri modi: ad esempio con l'integrazione di bigrammi e trigrammi nei dizionari, ossia insiemi di due o tre parole che nel complesso identificano uno stato d'animo, anche se prese singolarmente non avrebbero significato ai fini dell'analisi.

2.2 L'algoritmo

A questo punto si ha un vettore di liste di token, ognuna corrispondente ad un tweet, e si procede con la fase cruciale, quella di attribuzione della polarità. L'algoritmo scorre un elemento del vettore alla volta e confronta gli stem della lista con quelli presenti nei due dizionari. Gli stem vengono analizzati singolarmente e gli si assegna uno score positivo (+1) o negativo (-1) se lo si individua rispettivamente nel dizionario dei positivi o dei negativi. Nel caso di mancato riscontro lo score sarà pari a zero. Infine questi punteggi vengono sommati: se la somma è maggiore di zero, il tweet viene classificato come "positivo" (+1), se minore di zero come "negativo" (-1), se la somma è uguale a zero, allora è classificato "neutro".

2.3 Applicazione e Risultati

In `TextWiller` la funzione che confronta gli stem con i dizionari per poi assegnare la polarità è `sentiment()`, che applichiamo sempre al data-set presentato nel capitolo 1, tenendo conto che la funzione `sentiment()` richiama `normalizzaTesti()`.

Applichiamo l'algoritmo allo stesso sottoinsieme di prova che abbiamo analizzato nel capitolo precedente:

```
> sentiment <- sentiment(prova)
```

	sentiment	text
1	1	bravo ennio massimo emotegood wwwurlwww
2	1	rt @grgre83 buonanotte ballo matti tie tac emotegood wwwurlwww
3	1	rt soproud favore grazie mille emotegood wwwurlwww
4	-1	notizia riportata tg emotebad wwwurlwww
5	-1	insulti vittime foibe oggi #10febbraio 2014 facebook emotebad @il piccolo @messve
6	1	rt @esenzioneticket emotelol stesso messo insieme me eternauta pensa roba emotego

2.3.1 Analisi descrittive

Applichiamo la funzione `sentiment()` a tutto il data-set e ne presentiamo le analisi descrittive preliminari:

```
> sentiment <- sentiment(DataSet$text)
```

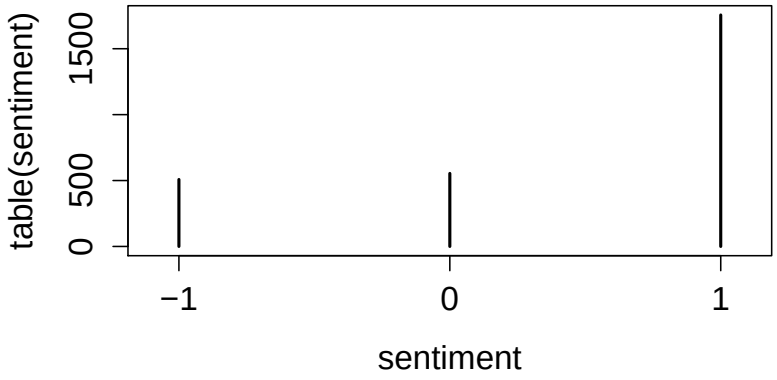
	sentiment
-1	508
0	554
1	1756

Tabella 2.1: Classificazione ottenuta con l'algoritmo ontologico

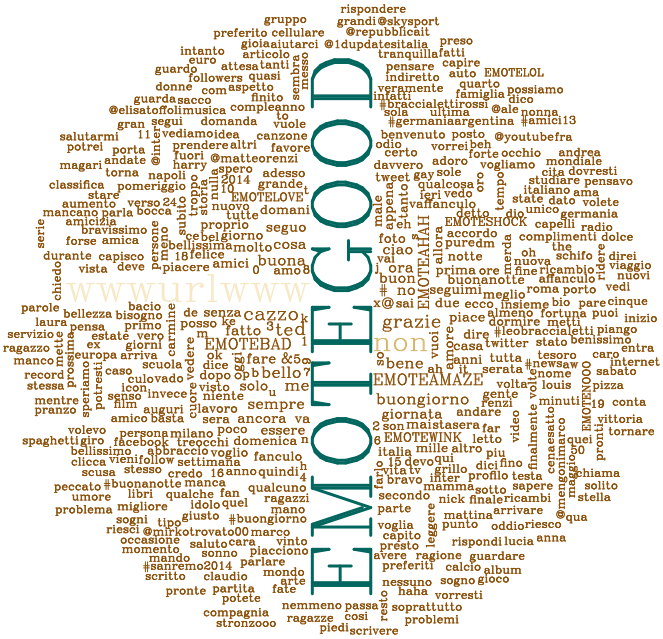
	sentiment
-1	0.18
0	0.20
1	0.62

Tabella 2.2: proporzioni delle classi ottenute con l'algoritmo ontologico

```
> plot(table(sentiment), xlim=c(-1.1,1.1))
```



Stampiamo la *wordcloud* per avere una panoramica generale delle parole presenti nei nostri tweets e della loro numerosità.



Notiamo quindi che vi è una netta prevalenza dell'uso di emoticon felici, rappresentate dalla stringa `EMOTEGOOD`. Questo è confermato anche dalle tabelle 6.1 e 6.2 che indicano una netta prevalenza di tweets classificati positivi.

Possiamo calcolare l'errore di assegnazione dell'algoritmo, confontando i risultati con quelli ottenuti dalla classificazione manuale.

	vero		
sentiment	-1	0	1
-1	253	216	39
0	113	339	102
1	112	722	922

Tabella 2.3: Tabella di errata classificazione dell'algoritmo basato sull'uso dei dizionari ontologici

Oltre all'errore totale, pari a 0.46, dalla tabella 6.3 sono stati calcolati anche gli errori relativi a ciascuna polarità, evidenziando che l'errore più alto (0.73) viene commesso nella classificazione dei tweets neutri, ossia che non hanno una polarità, al contrario i tweets classificati meglio sono quelli positivi, con un errore relativo pari a 0.13. Infine i negativi vengono classificati con un errore relativo pari allo 0.47.

Come già precisato, l'errore può essere ulteriormente abbassato affinando la fase di preprocessing e le parole contenute nei dizionari, individuando soprattutto gli n-grammi che assumono un significato differente dalle singole parole che li compongono e che aiuterebbero nell'individuazione di ironia e sarcasmo. Inoltre c'è da ricordare che la classificazione manuale stessa è soggetta ad errore.

Capitolo 3

Tree-Based Model per la Sentiment Analysis

Poiché il risultato della classificazione con l'algoritmo basato sui dizionari ontologici non risulta soddisfacente, si è passati ad un approccio di tipo statistico. Come punto di partenza utilizziamo gli alberi di classificazione, che sono uno strumento di semplice interpretazione e che si distinguono per la rapidità di calcolo. L'idea di base è quella di confrontare il metodo con lo stesso integrato però con i risultati della classificazione dell'algoritmo basato sui dizionari ontologici, per verificare se tale aggiunta porta dei miglioramenti nella classificazione.

3.1 Data-set

Fin'ora abbiamo usato i dati originari così come sono stati reperiti, ma per le analisi che seguiranno abbiamo bisogno di rielaborarli ulteriormente. Dopo aver "pulito" il testo con la solita funzione `normalizzaTesti()`, si procede con la creazione di una Document Term Matrix (DTM) (vedi tabella 6.1), una matrice con 2818 righe a cui corrispondono i tweets, e 445 colonne, di cui 444 corrispondono ai conteggi dei tokens che compaiono almeno cinque volte nei post e una indica il valore della classificazione manuale, quella che noi considereremo variabile risposta in tutte le nostre analisi.

qualcuno	sempre	piace	amore	buon	subito	italiano
0	2	0	0	0	0	0
0	2	0	0	0	0	0
0	2	0	0	0	0	0
0	0	0	0	0	2	0
0	0	0	2	0	0	0
0	2	0	0	0	0	0
0	2	0	0	0	0	0
0	0	2	0	0	0	0
0	0	2	0	0	0	0
0	0	0	0	2	0	0

Tabella 3.1: Piccola porzione della matrice DTM

Per evitare di imbattersi nel problema del sovra-adattamento, dividiamo il nostro data-set in insieme di stima (75% dei dati) e insieme di verifica (restante 25%): il primo funzionale a costruire i vari modelli ed il secondo ad un loro confronto (su di esso verrà infatti misurato, per ciascuno di essi, un indice del loro errore).

3.2 Tree-Based Models

Il partizionamento ricorsivo è uno strumento fondamentale nel data mining. Essa ci aiuta a esplorare la struttura di un insieme di dati, sviluppando regole decisionali, che siano facili da visualizzare, per la previsione di una variabile risposta categoriale (albero di classificazione).

Per generare tali alberi di classificazione ci sono due pacchetti in `R`: `Rpart` e `tree`. Sebbene i due metodi siano molto simili, nelle analisi successive utilizzeremo il primo pacchetto, che individua l'albero ottimo utilizzando una procedura suddivisa in due fasi: una prima fase vede la crescita dell'albero secondo il massimo decremento di impurezza (nel nostro caso adottiamo l'indice di Gini); in un secondo momento l'albero in questione viene potato tramite analisi di costo-complessità e stime di cross-validazione.

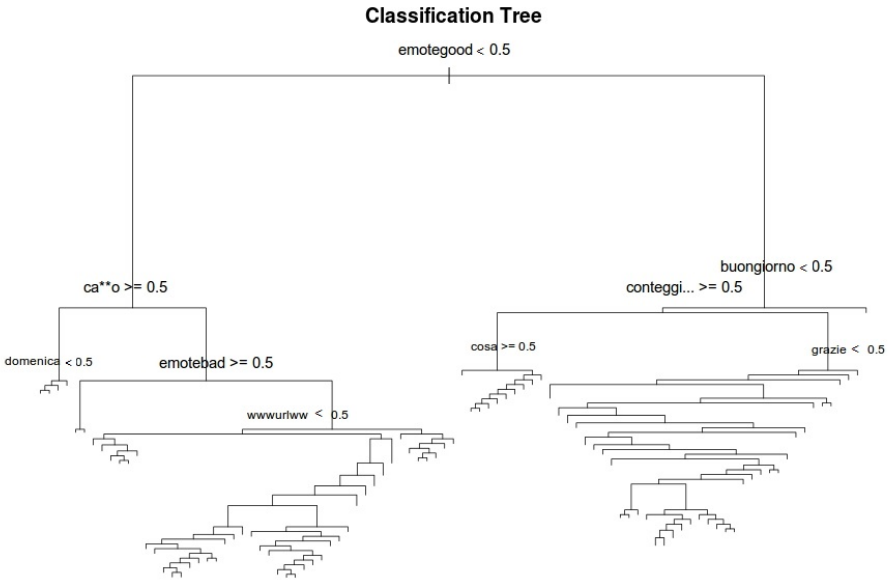
3.3 Analisi con il pacchetto Rpart

Faccio innanzitutto crescere il mio albero con il comando:

```
> library(rpart)
> treeRpart=rpart(truth~.,data=train1,method="class",xval=20,
+                 minsplit=2,cp=0.001)
```

Come detto precedentemente, la variabile risposta è chiamata "vero" (`truth`), ma essa si riferisce ai valori indicati in fase di classificazione manuale, quindi precisiamo che è comunque frutto di una classificazione soggettiva. Otteniamo quindi l'albero con il massimo numero di nodi, come rappresentato in figura.

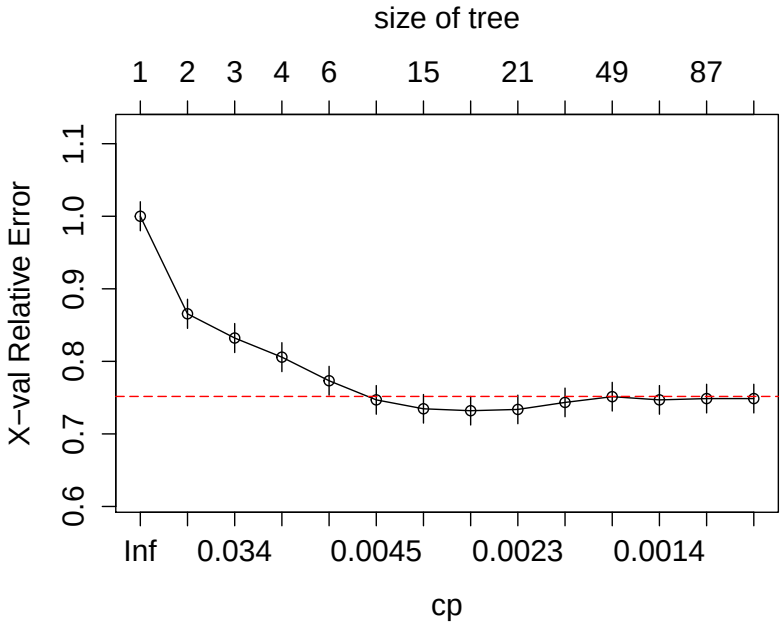
```
> plot(treeRpart, main="Classification Tree")
```



Esaminiamo il grafico seguente, che ci consente di confrontare i vari valori del parametro di penalizzazione per il costo-complessità dell'albero, in rela-

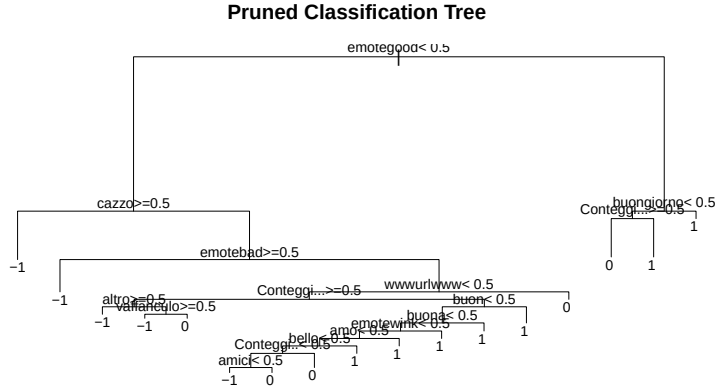
zione al numero di *split* che verrebbero mantenuti in seguito alla potatura dell'albero e all'errore di cross-validation.:

```
> plotcp(treeRpart, minline = TRUE, lty = 5, col = 2)
```



Scelgo quindi il parametro che minimizza l'errore e poto l'albero, ottenendo:

```
> cpmin<-treeRpart$cptable[which.min(treeRpart$cptable[,"xerror"]),"CP"]
> treeRpart1= prune(treeRpart, cp=cpmin)
> plot(treeRpart1, main="Pruned Classification Tree")
```



Utilizzo l'insieme di verifica per verificare la bontà del modello appena ottenuto:

```
> predict <- predict(treeRpart1,newdata=test1,type="class")
```

	vero		
predict	-1	0	1
-1	53	29	8
0	51	154	59
1	21	121	201

Tabella 3.2: Matrice di confusione del campione di verifica usando un albero di classificazione

Alla luce dei risultati ottenuti in tabella 3.2, l'errore di classificazione è pari a 0.41, quindi miglioriamo quello che era l'errore ottenuto con l'utilizzo dell'algoritmo basato sui dizionari ontologici.

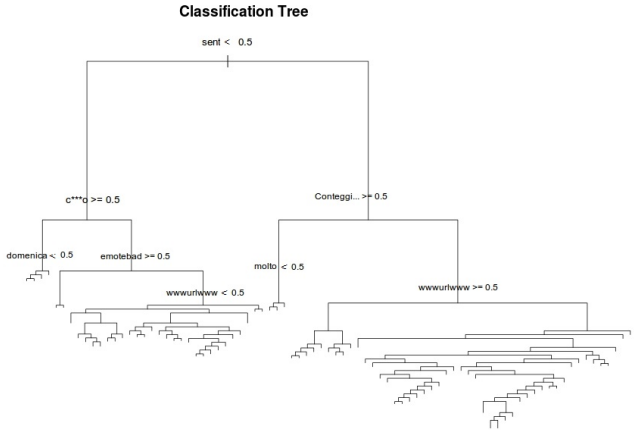
3.3.1 Analisi con il pacchetto `Rpart` integrando i risultati dell'algoritmo basato sui dizionari ontologici

Abbiamo evidenziato come l'errore di classificazione con `Rpart` sia diminuito rispetto a quello ottenuto con l'uso dell'algoritmo.

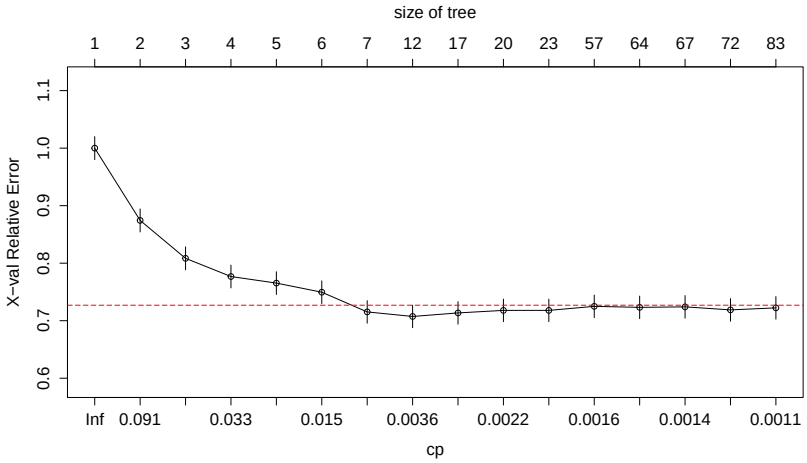
Quello che vogliamo verificare ora è se miglioriamo ulteriormente la classificazione integrando i risultati dell'algoritmo stesso. Riproponiamo quindi le stesse analisi della sezione precedente, con l'aggiunta di una colonna con la classificazione ottenuta con l'algoritmo, e ne valutiamo gli eventuali miglioramenti che questo apporterebbe alle previsioni.

Come prima, si fa crescere l'albero sui dati e ne analizziamo i risultati:

```
> library(rpart)
> treeRpart=rpart(truth~.,data=trainA, method="class", xval=20,
+                 minsplit=2, cp=0.001)

> plot(treeRpart)
```

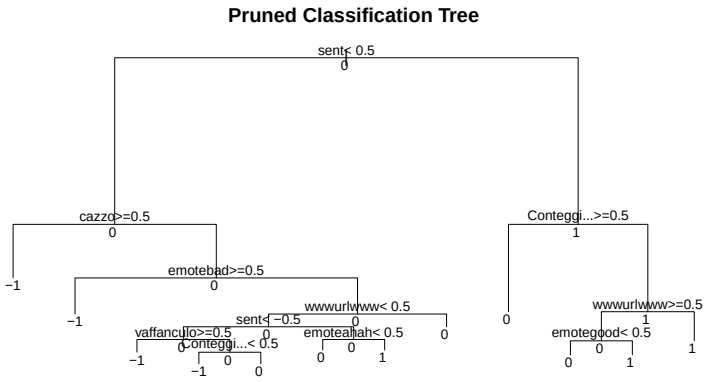



```
> plotcp(treeRpart, minline = TRUE, lty = 5, col = 2)
```



Sempre come prima, una volta individuato il parametro che minimizza l'errore si pota l'albero:

```
> cpmin<-treeRpart$cptable[which.min(treeRpart$cptable[,"xerror"]),"CP"]
> treeRpart1= prune(treeRpart, cp=cpmin)
> plot(treeRpart1, main="Pruned Classification Tree")
```



E infine verifico la bontà del modello, confrontandolo con il precedente:

```
> predictA <- predict(treeRpart1,newdata=testA,type="class")
```

	vero		
predict	-1	0	1
-1	63	31	8
0	38	163	60
1	24	110	200

Tabella 3.3: Matrice di confusione del campione di verifica usando un albero di classificazione, integrando il risultato ottenuto con l'algoritmo ontologico

Dalla tabella 3.3 otteniamo un errore di classificazione pari a 0.39, migliorando quindi di poco quello ottenuto precedentemente.

Capitolo 4

La regressione multinomiale inversa (MNIR)

Un approccio di tipo bayesiano al problema di Text Mining lo ritroviamo nel metodo della la regressione multinomiale inversa, proposto piuttosto recentemente da Taddy [13]. Anche in questo caso le analisi vengono riproposte due volte, nel secondo caso viene integrato il risultato della classificazione effettuata con l'algoritmo ontologico, sempre per verificarne l'eventuale miglioramento che ciò apporterebbe.

4.1 Il metodo di Taddy

Gli approcci classici nella Sentiment Analysis, come support vector machines (SMV), regressione delle componenti principali (PC), reti neurali e minimi quadrati penalizzati, utilizzano la frequenza delle parole come predittori. Questi metodi hanno dei limiti,non colgono i particolari (anche discriminanti) del testo e analisi indipendenti di più tabelle di contingenza portano a problemi di test multipli. Con la strategia alternativa proposta, invece, si adatta a p(x|y) un modello di regressione logistica multinomiale costruito sul vocabolario, utilizzando gli argomenti dei documenti come caratteristiche della distribuzione. Un esempio può essere la meglio conosciuta allocazione di Dirichlet latente (LDA), che considera un documento come un miscuglio di

argomenti, ciascuno con la propria distribuzione di Dirichlet e caratterizzato da un gruppo di parole (*stem*) con distribuzione multinomiale:

$$x_i \sim MN(w_{i1}\vartheta_1 + \ldots + w_{ik}\vartheta_k, m_i)$$

dove $\vartheta_k = [\vartheta_{k1} \ldots \vartheta kp]'$ è il vettore di probabilità degli argomenti e w_i i pesi.

4.1.1 Regressione Inversa e Riduzione Sufficiente

Il problema di fondo è principalmente la grande dimensione dei dati presi in analisi, impedendo di fatto di poter stimare la risposta condizionata $Y|X$ senza prima aver semplificato X. In questa fase ci viene in aiuto la tecnica della Regressione Inversa (IR) [4], che usa la distribuzione multivariata condizionata $X|Y$ per costruire dei sommari di X.
Si supponga che v_i sia un vettore di K fattori risposta, attraverso cui x_i dipenda da y_i, allora la formulazione lineare della Regressione Inversa sarà $x_i = \Phi v_i + \varepsilon_i$, dove $\Phi = [\varphi_1 \ldots \varphi_K]$ è una matrice $p \times K$ di coefficienti IR e ε_i è un vettore p di termini d'errore.
Sotto certe condizioni la proiezione $z_i = \Phi' x_i$ fornisce una Riduzione Sufficiente tale che y_i è indipendente da x_i dato z_i. Quindi bisogna stimare Φ, e quando tale stima è fattibile, la riduzione di dimensione da p a K rende queste proiezioni SR di più semplice utilizzo rispetto ai predittori originali.
L'innovazione della Regressione Inversa sta nell'analizzare le proiezioni della Riduzione Sufficiente che derivano da una specificazione di v_i come funzione non lineare di y_i.

4.1.2 Il modello

Il modello della Regressione Multinomiale Inversa è il seguente:

$$x_i \sim MN(q_i, m_i) \text{ con } q_{ij} = \frac{e^{\eta}_{ij}}{\sum_{l=1}^{p} e^{\eta}_{il}},$$

$$j = 1, \ldots, p \text{ dove } \eta_{ij} = \alpha_j + u_{ij} + v_i' \varphi_j$$

Abbiamo quindi un fattore v_i K-dimensionale e l'effetto dell'argomento $u_i = [u_{i1} \dots u_{ip}]'$.

Riduzione Sufficiente nella regressione multinomiale inversa

Stabiliamo a questo punto la sufficienza classica per y condizionatamente ai parametri di Regressione Inversa. Sotto il modello precedentemente definito e condizionatamente a m_i e u_i abbiamo $y_i \perp x_i \Rightarrow y_i \perp x_i \mid \Phi' x_i$. Utilizzando, invece, le frequenze al posto dei conteggi si ottiene $y_i \perp x_i \mid \Phi' x_i, m_i \wedge p(y|x_i) = p(y_i|f_i) \Rightarrow y_i \perp x_i \mid z_i = \Phi' \mathrm{f}_i$
La stima ottenuta dalla regressione multinomiale inversa è spesso basata sulla riduzione delle variabili nel modello, tenendo conto che le variabili omesse devono essere indipendenti da x_i condizionatamente a v_i, mentre le covariate che agiscono su x_i e sono indipendenti da v_i devono essere incluse nel modello.

Specificazione della distribuzione a priori

Per completare il modello MNIR, si stabilisce una distribuzione a priori per l'intercetta α, i pesi Φ e i possibili effetti casuali U.
Prima di tutto a ogni frase dell'intercetta viene assegnata una a priori normale $\alpha_i \sim N(0, 1)$. Successivamente per ogni φ_{ik} si propone una distribuzione a priori, con penalità λ_{ik} tali che $\pi(\varphi_{jk}) = \frac{\lambda_{jk}}{2\exp(-\lambda_{jk}|\varphi_{jk}|)}$ con $j = 1, \dots, p$ e $k = 1, \dots, K$. A ciascun λ_{jk} è assegnata una distribuzione gamma (detta iperpriori coniugata) $Ga(\lambda; s, r) = \frac{r^s}{\Gamma(s)\lambda_{jk}^{s-1}\exp -r\lambda_{jk}}$.
Se $s \leq 1$, la densità della iperpriori per φ_{jk} cresce al decrescere della penalità. Questa strategia funziona bene in molte applicazioni quando la dimensione non è molto più grande di 10^3. Nonostante ciò, quando il vocabolario contiene più di 10^5 vocabolari, è conveniente incrementare i valori di s e r per una rapida convergenza per mantenere la quantità di valori non nulli abbastanza piccola e gestibile.
Infine, usiamo $\exp[u_{ij}] \sim Ga(1, 1)$ indipendente per ogni i e j come modello illustrativo degli effetti casuali.
I conteggi non sono immediatamente riducibili in presenza di effetti casuali,

ma le ipotesi sul processo di generazione per x_i indipendente da m_i possono essere usate per costruire un modello a priori sul loro effetto sull'aggregazione di conteggi: se ciascun x_{ij} è tratto da una distribuzione Poisson $Po(e^{\alpha_j+u_{ij}+v_i\varphi_j})$ con $\exp[u_{ij}] \sim Ga(1,1)$ e $n_v = \sum_i \mathbb{I}_{[v_i=v]}$, allora $x_{vj} \sim Po(e^{\alpha_j+u_{vj}+v\varphi_j})$ con $\exp[u_{vj}] \overset{ind}{\sim} Ga(n_v, 1)$.
Per comodità usiamo una approssimazione log-normale per la gamma, in modo che aggregando un gran numero di osservazioni, la modellazione ad effetti casuali diventi obsoleta. Infatti essa è così definita: $u_{v,j} \sim N(log(n_v) - 0.5\sigma_v^2,\ \sigma_v^2)$ con $\sigma_v^2 = log(n_v+1) - log(n_v)$, quindi per n_v grande $\sigma_n^2 \to 0$.

Stima

In seguito alla specificazione del modello fin ora trattato, la distribuzione a posteriori di interesse sarà:

$$p(\Phi, \alpha, \lambda, U \mid X, V) \propto \prod_{i=1}^{n}\prod_{j=0}^{p} q_{ij}^{x_{ij}}\pi(u_{ij})N(\alpha_j; 0, \sigma_\alpha^2) \times \prod_{k=1}^{K} GL(|phi_{jk}, \lambda_{jk})$$

con $q_{ij} = \exp[\eta_{ij}]/\sum_{l=1}^{p}\exp[\eta_{il}]$ e $\eta_{ij} = \alpha_j + u_{ij} + \sum_{k=1}^{K} v_{ik}\varphi_{ik}$ e GL è il coefficiente di penalità a priori $Laplace(\varphi_{jk}; \lambda_{jk})Ga(\lambda_{jk}; r, s)$.
Stimare Φ e λ sotto il modello indipendente a priori Gamma-Laplace equivale a minimizzare la log-verosimiglianza per Φ soggetto ai costi:
$c(\Phi) = \sum_{j=1}^{k}\sum_{k=1}^{K} c(\varphi_{jk})$, dove $c(\varphi_{jk}) = s\log(1 + |\varphi_{jk}|/r)$.

Il modello di regressione multinomiale inverso fornisce quindi un metodo predefinito veloce per la riduzione del documento di interesse, grazie all'utilizzo dell'inferenza bayesiana che ne semplifica notevolmente i calcoli.

In R queste procedure possono essere applicate con il pacchetto `textir`, come si può vedere di seguito.

4.2 Analisi con il pacchetto `textir`

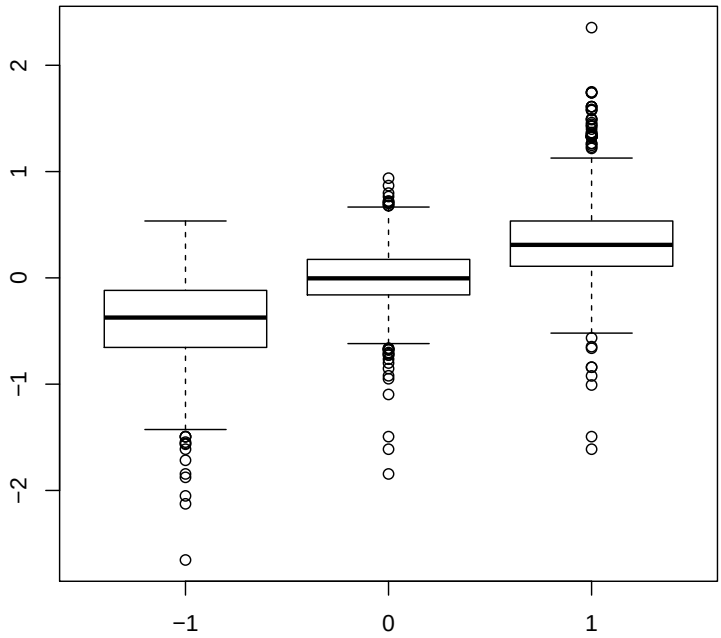
Carico il pacchetto `textir` che ci consente di applicare un modello di regressione multinomiale inversa ai nostri dati dell'insieme di stima. La funzione di nostro interesse è `mnlm`, che adatta i parametri della regressione logistica multinomiale a una verosimiglianza fattorizzata di Poisson, con penalizzazione Gamma-Lasso.

```
> library(textir)
> train$truth<-as.numeric(train$truth)
> a<-which(apply(train, 2, function(x) sum(x)) == 0)
> dati<-train[,-a]
> truth <- as.matrix(dati[,which(names(dati)=="truth")])
> #apply(dati, 2, function(x) sum(x>1))
>
> cl<-NULL
> words<-data.matrix(dati[,-which(names(dati)=="truth")])
> fits <- mnlm(cl, covars=truth, counts=words, bins=3)
> # calcolo i coef
> B <- coef(fits)
> ## some big loadings in IR
> B[2,order(B[2,])[1:20]]

     schifo       manco       merda   stronzooo  vaffanculo        odio #sanre
  -5.308007   -5.308007   -3.354067   -2.447987   -2.281189   -2.125709    -2.
     andate      potete      stessa     fanculo       posto        @ale
  -1.908089   -1.908089   -1.908089   -1.844529   -1.776247   -1.702497    -1.
     quarto      sembra     momento       senso
  -1.702497   -1.622344   -1.622344   -1.622344

> B[2,order(-B[2,])[1:20]]

      anna  compagnia     attesa      dolce compleanno complimenti  bravissimo
  4.903119   4.903119   4.903119   4.903119   4.903119   4.903119   4.903119
     adoro  abbraccio buongiorno  emotelove bellissimo  settimana       mille
```

```
   2.381977    2.291246    2.119544    1.893813    1.787718    1.787718    1.
      lucia       pizza
   1.600778    1.600778
```

```
> z <- srproj(B, words)
```

```
> boxplot(z[,1]~classe)
```



La funzione `srproj` calcola la Riduzione Sufficiente della variabile di interesse (ossia l'argomento `covars` in `mnlm`) a partire dai conteggi delle parole del testo (`counts` in `mnlm`). Adesso si può procedere con la regressione vera e propria: per farlo utilizziamo un modello lineare generalizzato vettoriale.

```
> library(VGAM)
> m1<-vglm(truth ~ z,family=multinomial(), data=
+             data.frame(truth=dati$truth, z=z[,1]))
> summary(m1)
```

```
Call:
vglm(formula = truth ~ z, family = multinomial(), data = data.frame(truth = dati$tru
    z = z[, 1]))

Pearson residuals:
                         Min       1Q   Median       3Q     Max
log(mu[,1]/mu[,3]) -103.311 -0.30579 -0.14756 -0.039823 10.3830
log(mu[,2]/mu[,3])  -98.971 -0.73792 -0.29773  0.812903  6.7684

Coefficients:
              Estimate Std. Error  z value
(Intercept):1  -0.8245   0.094978  -8.6809
(Intercept):2   0.7421   0.063320  11.7198
z:1            -6.6554   0.298202 -22.3186
z:2            -3.3731   0.200843 -16.7948

Number of linear predictors:  2

Names of linear predictors: log(mu[,1]/mu[,3]), log(mu[,2]/mu[,3])

Dispersion Parameter for multinomial family:   1

Residual deviance: 3323.545 on 4178 degrees of freedom

Log-likelihood: -1661.772 on 4178 degrees of freedom

Number of iterations: 6
```

Ora testiamo l'errore del modello sull'insieme di verifica.

```
> test$truth<-as.numeric(test$truth)
> testSet<- test[,-a]
> test1<-data.matrix(testSet[,-which(names(dati)=="truth")])
```

```
> zNew<- srproj(B, test1)
> p.vglm <-predict(m1, newdata=data.frame(z=zNew[,1]), type="response")
```

Otteniamo la tabella di contingenza 4.1, dalla quale calcoliamo un errore di classificazione pari a 0.41.

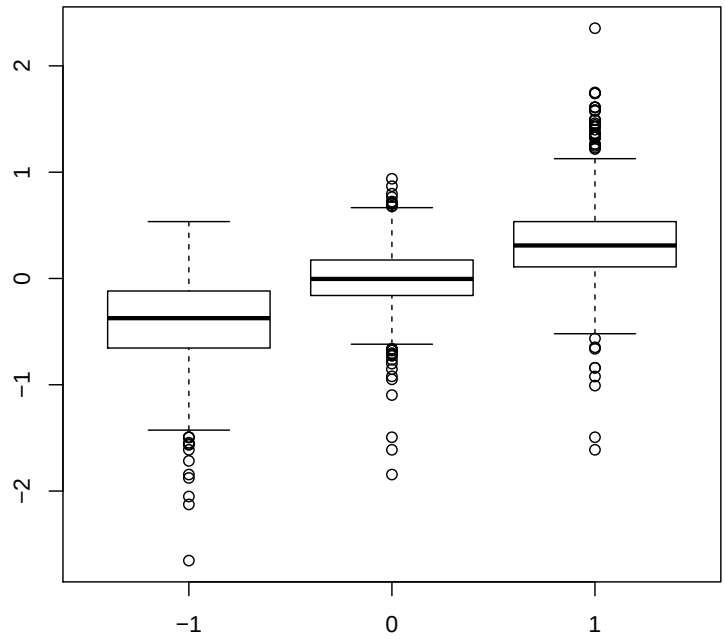
	vero		
sentiment	-1	0	1
-1	35	15	4
0	80	226	114
1	10	63	150

Tabella 4.1: Matrice di confusione del campione di verifica usando un modello di regressione multinomiale inversa

4.2.1 Analisi con il pacchetto `textir` integrando i risultati dell'algoritmo basato sui dizionari ontologici

Rifaccio tutti i passaggi precedenti, questa volta però includendo anche la classificazione risultante dall'applicazione dell'algoritmo. Ne verifichiamo quindi l'eventuale miglioramento.

```
> train$truth<-as.numeric(train$truth)
> a<-which(apply(train, 2, function(x) sum(x)) == 0)
> dati<-train[,-a]
> truth <- as.matrix(dati$truth)
> cl<-NULL
> words<-data.matrix(dati[,-which(names(dati)=="truth")])
> fits <- mnlm(cl,covars=truth, counts=words, bins=3)
> # calcolo i coef
> B <- coef(fits)
> z <- srproj(B, words)
> predinv=z[,1]

> boxplot(predinv~truth)
```




```
> library(VGAM)
> m1<-vglm(truth ~ z,family=multinomial(), data=
+             data.frame(truth=dati$truth, z=z[,1]))
> summary(m1)

Call:
vglm(formula = truth ~ z, family = multinomial(), data = data.frame(truth = dati$tru
    z = z[, 1]))

Pearson residuals:
                        Min       1Q   Median        3Q    Max
log(mu[,1]/mu[,3]) -187.06 -0.30475 -0.14986 -0.056267 7.7833
log(mu[,2]/mu[,3]) -183.41 -0.76170 -0.27238  0.816417 5.2297

Coefficients:
```

```
              Estimate Std. Error z value
(Intercept):1 -1.51101   0.100844 -14.984
(Intercept):2  0.36805   0.056037   6.568
z:1           -6.67443   0.295538 -22.584
z:2           -3.89387   0.229859 -16.940

Number of linear predictors:  2

Names of linear predictors: log(mu[,1]/mu[,3]), log(mu[,2]/mu[,3])

Dispersion Parameter for multinomial family:   1

Residual deviance: 3278.62 on 4178 degrees of freedom

Log-likelihood: -1639.31 on 4178 degrees of freedom

Number of iterations: 6
```

```
> test$truth<-as.numeric(test$truth)
> testSet<- test[,-a]
> test1<-data.matrix(testSet[,-which(names(testSet)=="truth")])
> zNew<- srproj(B, test1)
> p.vglm <-predict(m1, newdata=data.frame(z=zNew[,1]), type="response")
> summary(p.vglm)
```

```
      -1                  0                  1
 Min.   :0.0000152   Min.   :0.00537   Min.   :0.0000005
 1st Qu.:0.0383498   1st Qu.:0.36922   1st Qu.:0.1794040
 Median :0.0854671   Median :0.51240   Median :0.3672138
 Mean   :0.1657600   Mean   :0.45759   Mean   :0.3766521
 3rd Qu.:0.1844559   3rd Qu.:0.59824   3rd Qu.:0.5544962
 Max.   :0.9918578   Max.   :0.63885   Max.   :0.9946148
```

La tabella di contingenza ottenuta è:

	vero		
predict	-1	0	1
1	39	13	4
2	75	216	102
3	11	75	162

Tabella 4.2: Matrice di confusione del campione di verifica usando un modello di regressione multinomiale inversa integrando la classificazione ottenuta con l'algoritmo basato sui dizionari ontologici

L'errore di classificazione calcolato a partire dalla tabella 4.2 in questo caso è pari a 0.40, quindi leggermente meglio rispetto all'errore commesso con lo stesso metodo senza tenere conto della classificazione ottenuta con l'algoritmo.

Capitolo 5

Il metodo di Hopkings e King

A differenza dei metodi visti fin'ora, che si focalizzavano sulla classificazione individuale dei tweets (ciascun tweet classificato come "positivo"/"negativo"/"neutro"), il metodo di Hopkins e King, conosciuto anche come *ReadMe*, si concentra sulla classificazione aggregata, ossia la distribuzione di frequenza finale delle opinioni espresse nella totalità dei tweets, senza poter risalire alla classificazione del singolo post. [6]

5.1 Il metodo

Sia $D = (D_1, \ldots, D_k)$ l'insieme delle diverse opinioni espresse e $S_i = (S_1, \ldots, S_M)$ l'insieme delle parole utilizzate in un testo, con $i = 1, \ldots, N$, N il numero totale di testi presi in esame e M il numero totale di parole distinte. Calcolando per ciascuno degli N testi un vettore di probabilità di appartenere a una delle tre categorie, supponiamo di classificare i testi assegnando ciascuno alla categoria con probabilità di appartenenza più alta. Si passerà, dunque, alla classificazione aggregata contando il numero di volte in cui i testi sono classificati in un determinato modo: $P(D) = P(D|S)\ P(S)$, con $P(D)$ distribuzione aggregata delle opinioni espresso come vettore di probabilità di dimensione $K \times 1$, P(S) è un vettore di probabilità di dimensione $2^M \times 1$ e infine $P(D|S)$ è una matrice di dimensioni $K \times 2^M$.

5.1.1 Classificazione aggregata

L'intuizione di Hopkins e King è quella di cambiare punto di vista, concentrandosi sull'obiettivo finale che è l'ottenimento della distribuzione aggregata dei testi $P(D)$.

Si scomponga la distribuzione degli stem dell'intero insieme di dati $P(S)$ nel modo seguente:

$$P(S) = P(S|D)\ P(D) \tag{5.1}$$

dove $P(S|D)$ è una matrice di dimensione $2^M \times K$ e rappresenta la probabilità che una particolare sequenza di stem s di S compaia all'interno dei testi che sono classificati secondo una particolare categoria. Tale distribuzione è nota solo per i testi del training set (lo indichiamo con T), quindi bisogna fare l'assunzione che il linguaggio sia lo stesso in tutti i testi del corpus, cioè $P_T(S|D) = P(S|D)$ (questa assunzione verrà ripresa nel capitolo successivo). Detto ciò, posso sostituire la 5.1 con:

$$P(S) = P_T(S|D)\ P(D) \tag{5.2}$$

e la soluzione al nostro problema si otterrà invertendo la 5.2:

$$P(D) = P_T(S|D)^{-1}\ P(S) \tag{5.3}$$

A questo punto il problema 5.2 può essere visto come un modello classico di regressione, dove $P(D)$ rappresenta il vettore di coefficienti β, $P_T(S|D)$ una matrice X di regressione e $P(S)$ il vettore Y della variabile risposta, con l'usuale formula $Y = X\beta$ e soluzione data da $\beta = (X'X)^{-1}X'Y$.

La stima di β è resa difficoltosa dalla dimensionalità della matrice X (2^M cresce esponenzialmente al crescere del numero di stem) e dal fatto che è una matrice sparsa, ossia con molti zeri. (Chiedere aiuto al prof)

La caratteristica principale di questo approccio è che non si determina la classificazione aggregata della totalità dei post a partire dalla classificazione di ciascun tweet, ma si stimano direttamente le proporzioni aggregata $P(D)$, permettendoci di evitare la propagazione dell'errore dovuto all'aggregazione di singole classificazioni.

Anche dal punto di vista logico ha più senso considerare $P(S|D)$, ossia identificare le parole usate data l'opinione, invece che $P(D|S)$, ossia identificare l'opinione date le parole usate. Infatti solitamente chi scrive su un social network ha già chiara l'opinione che vuole esprimere, e solo successivamente sceglie le parole da usare per esprimere tale opinione.[3] Quindi in tale senso, il processo più naturale di generazione di dati è quello rappresentato dalla 5.2.

Uno dei punti di debolezza dell'analisi dei social media è quello di contenere molto rumore (testi fuori tema, altrimenti detti *OffTopics*, e testi che non esprimono un'opinione), creando di fatto una categoria aggiuntiva in cui rientrano questi casi e rendendo più difficile la stima di $P(D = D_j|S = s)$ in quanto il vettore di stem S comparirà molto più frequentemente per tale categoria (solitamente molto più grande) che nelle categorie semantiche di interesse. La presenza di questa categoria di OffTopics, o neutri nel nostro caso, fa aumentare la mole di lavoro, in quanto sarà necessario avere un training set più ampio che includa sufficienti casi per ciascuna categoria (solitamente 30-50).

5.2 Analisi e valutazione dei risultati

L'output delle nostre analisi, effettuate nel prossimo paragrafo con il pacchetto `R ReadMe`, sarà dunque un vettore di probabilità stimate. Per valutare la distanza tra tale vettore di probabilità e quello delle vere proporzioni delle classi, utilizziamo il seguente indice:

$$\log \frac{vero_{positivo}}{vero_{negativo}} - \log \frac{stimato_{positivo}}{stimato_{negativo}}$$

Ci concentriamo così sul confronto tra positivi e negativi, che sono le classi di maggiore interesse per chi affronta il problema della Sentiment Analysis. Tale indice ci consente di confrontare più vettori di probabilità e individuare quale ha le proporzioni più vicine alle vere proporzioni, ossia quello maggiore in valore assoluto, mentre il segno ci permette di capire quale è la classe che viene sovrastimata.

Volendo calcolare invece la distanza tra i due vettori, tenendo conto anche della classe "neutri", un indice valido potrebbe essere:

$$\sum_{i\ in\ \{neg;\ neutro;\ pos\}} \left(\frac{(n*dato\ stimato_i - n*dato\ vero_i)^2}{n*dato\ vero_i}\right)$$

Tale misura coincide con il test χ^2.

5.3 Analisi con il pacchetto `ReadMe`

Iniziamo le nostre analisi utilizzando il pacchetto `ReadMe`, che come spiegato precedentemente, fornisce come output finale un vettore di probabilità, corrispondente appunto alla distribuzione aggregata dei sentimenti.

Sappiamo che il metodo in questione riceve in input una serie di documenti, ognuno contenente un testo (o, nel nostro caso, un tweet). Quindi dobbiamo creare tali file di testo, sia per il traning set che per il test set:

```
> sample<-1:nrow(train)
> id<-NULL
> sapply(sample, function(id) cat(names(train[which(train[id,]!=0)]),
+        file=paste(id,sep="",".txt")))
> sample1<-1:nrow(test)
> sapply(sample1, function(id) cat(names(test[which(test[id,]!=0)]),
+        file=paste(id+nrow(train),sep="",".txt")))
```

Ora necessitiamo di un file di controllo che contenga tutti i file appena creati. Tale file di controllo deve contenere tre campi: `FILENAME`, che riporta il nome dei singoli file di testo, `TRAININGSET` che è impostato a 1 per le righe contenenti i file di tweets appartenenti al traningset e a 0 per quelli appartenenti al testset, `TRUTH` che contiene il valore della classificazione manuale e può essere omesso per le righe del testset; se presente, come nel nostro caso, non verrà usato durante la stima ma sarà utilizzato per la stampa e in fase di confronto.

```
> write.csv(file="controlTweet.txt",cbind(FILENAME=paste(1:2788,
+ ".txt",sep=""),TRUTH=c(vero,testvero),
+ TRAININGSET=c(rep(1,nrow(train)),
```

Ora si può dunque procedere con le analisi.Applico la prima funzione undergrad che converte un insieme di testi memorizzati in una singola cartella in un data set. Ciascun testo è rappresentato da una riga e ogni parola è una colonna, con 1 si indica se la parola appare nel testo, altrimenti il valore sarà 0. Ne stampiamo una porzione per avere un'idea:

```
> library(ReadMe)
> library(quadprog)

> undergrad.results<-undergrad(control="controlTweet.txt",
+                              sep=",",threshold=0.01)

> undergrad.results$testset[1:20,1:6]
```

	FILENAME	TRUTH	TRAININGSET	WORD.emotebad	WORD.conteggi	WORD.cosa
2092	2092.txt	1	0	0	1	0
2093	2093.txt	1	0	0	1	0
2094	2094.txt	1	0	0	1	0
2095	2095.txt	1	0	0	1	0
2096	2096.txt	-1	0	0	1	0
2097	2097.txt	0	0	0	1	0
2098	2098.txt	1	0	0	1	0
2099	2099.txt	1	0	0	1	0
2100	2100.txt	-1	0	0	1	0

La funzione appena applicata fornisce tutti i valori necessari a readme per effettuare le stime. Prima di applicarla, però, bisogna usare preprocess, che consente l'eliminazione delle colonne invarianti, se presenti:

```
> undergrad.preprocess<-preprocess(undergrad.results)

Removed No Invariant Columns
```

Nel nostro caso quindi rimaniamo con tutte le colonne e possiamo usare la funzione finale, che ci fa ottenere le stime delle probabilità di ciascun sentimento. Tali stime possono essere confrontate con il valore vero delle probabilità, rappresentate da quelle ottenute dalla classificazione manuale:

	negativo	neutro	positivo
vero	0.1793400	0.4361549	0.3845050
stimato	0.1885753	0.4278766	0.3835481

Tabella 5.1: Vettore di probabilità calcolato con `readme`

```
> readme.results<- readme(undergrad.preprocess)
```

Calcoliamo quindi l'indice per valutare la distanza tra questi due vettori e risulta pari a -0.075.

5.3.1 Analisi con il pacchetto `ReadMe` integrando i risultati dell'algoritmo

Per prima cosa bisogna integrare il risultato della classificazione dell'algoritmo nel testo dei tweets, quindi individuiamo tre stringhe che identifichino le tre possibili categorie e saranno le tre nuove colonne aggiuntive nella matrice sparsa. Avremo quindi le colonne *pospositivo*, *negnegativo* e *neuneutro* che assumeranno valore 1 nelle righe corrispondenti ai tweets che l'algoritmo ha classificato rispettivamente positivi, negativi o neutri, e 0 altrove. Questo lo applico a tutto il data-set, sia test set che training set.

```
> datiM$pospositivo[which(datiM$sent==1)]<-1
> datiM$negnegativo[which(datiM$sent==-1)]<-1
> datiM$neuneutro[which(datiM$sent==0)]<-1
> #
> testM$pospositivo[which(testM$sent==1)]<-1
> testM$negnegativo[which(testM$sent==-1)]<-1
> testM$neuneutro[which(testM$sent==0)]<-1
```

Come fatto precedentemente, creo i file che contengono un tweet ciascuno:

```
> sample<-1:nrow(train)
> id<-NULL
> sapply(sample, function(id) cat(names(train[which(train[id,]!=0)]),
+       file=paste(id,sep="",".txt")))
```

```
> sample1<-1:nrow(test)
> sapply(sample1, function(id) cat(names(test[which(test[id,]!=0)]),
+       file=paste(id+nrow(train),sep="",".txt")))
```

Scrivo il file di testo di controllo, che contiene tutti i file contenenti i singoli tweet, il vero valore della classificazione e il campo `TRAININGSET` è impostato a 1 per i file appartenenti al training set e a 0 per quelli contenuti nel test set.

```
> write.csv(file="controlTweet.txt",cbind(FILENAME=paste(1:2788,
+ ".txt",sep=""),TRUTH=c(sent,senttest),TRAININGSET=c(rep(1,nrow(train)),
+ rep(0,nrow(test)))), row.names=FALSE, quote=FALSE)
```

Dopo aver caricato le librerie necessario, si possono iniziare le analisi:

```
> library(ReadMe)
> library(quadprog)
> undergrad.results<-undergrad(control="controlTweet.txt",
+                                sep=",",threshold=0)
> undergrad.preprocess<-preprocess(undergrad.results)

Removed 34 Invariant Columns
```

Come spiegato precedentemente, la funzione (undergrad) legge i file di testo contenuti nel file di controllo, e costruisce un data frame in cui ciascun testo è rappresentato da una riga e ogni parola è una colonna. La presenza/assenza di una parola è indicata dai valori 1/0. L'opzione `threshold` è impostata a 0 per includere tutte le parole. Il valore predefinito sarebbe 0.01, che significherebbe includere nel data frame tutte le parole che si verificano in più dell'1% dei testi. Il data frame contiene tutti i valori necessari da passare alla funzione `readme`, prima bisogna però applicare `preprocess`, che rimuove le colonne invarianti.

Ora si può applicare la funzione `readme`, che effettua le analisi necessarie per ottenere il vettore di probabilità del sentimento dei nostri tweets:

```
> readme.results<- readme(undergrad.preprocess)
```

	negativo	neutro	positivo
vero	0.1793400	0.4361549	0.3845050
stimato	0.1552142	0.4278766	0.3845050

Tabella 5.2: Vettore di probabilità calcolato con `ReadMe` integrando i risultati dell'algoritmo

Ne stampo i risultati, prima quelli ottenuti dalla stima, poi quelli considerati veri, ossia frutto della classificazione manuale: Calcolo anche in questo caso la distanza, che è pari a -0,075. Quindi possiamo concludere che in questo caso il metodo funziona meglio se non viene integrato con la classificazione ottenuta con l'algoritmo basato sui dizionari ontologici.

Capitolo 6

Vettori di probabilità

Il metodo di Hopkins e King non è l'unico a fornire come output un vettore di probabilità di appartenere ad una classe.

I metodi visti precedentemente (gli alberi di classificazione e la regressione multinomiale inversa) producono un vettore di probabilità, tuttavia invece che calcolarlo sulla totalità dei tweets, lo fanno sulla singola unità statistica (il singolo post). Per ottenere quella che nel precedente capitolo chiamavamo distribuzione aggregata bisognerà quindi calcolare la media dei vettori di tutti i tweets.

L'utilità di questo metodo sta nell'evitare il problema della scelta dei valori soglia (*threshold* o *cut-off*), cioè quei valori assunti dalla variabile misurata nel test per definire gli intervalli di discriminazione delle tre classi . Ciò significa che scegliendo valori di cut-off differenti si ottengono classificazioni diverse. Una volta scelta la soglia, ciascun tweet viene assegnato alla classe che ha probabilità maggiore. Questa operazione comporta un errore, che moltiplicato per tutti i tweets si espande. Mantenere invece l'informazione sulla probabilità di ciascun tweet di appartenere alle tre classi possibili non ci consente di classificare il singolo post, ma permette di avere un'idea più chiara sull'andamento in generale ed evitare quindi di dover definire una soglia.

Per verificare se sia più ragionevole lavorare con tali vettori, per ciascun metodo stamperemo i valori e un indice di distanza dal vero vettore di

probabilità, e mostreremo che sono migliori delle percentuali ricavate dalle proporzioni delle classificazioni già effettuate nei capitoli precedenti.

L'indice utilizzato è lo stesso del precedente capitolo:

$$\delta = \log \frac{vero_{positivo}}{vero_{negativo}} - \log \frac{stimato_{positivo}}{stimato_{negativo}}$$

e sempre come nel precedente capitolo il confronto si limiterà a due classi, ossia "positivo" e "negativo", poiché sono quelle di principale interesse per chi si accinge a questo tipo di analisi. Un'ulteriore distanza verrà calcolata anche con l'indice di χ^2, considerando quindi tutte e tre le classi.

$$\chi^2 = \sum_{i\ in\ \{neg;\ neutro;\ pos\}} \left(\frac{(n*dato\ stimato_i - n*dato\ vero_i)^2}{n*dato\ vero_i}\right)$$

Anche in questa fase dell'analisi trattiamo ciascun metodo nei due casi, ossia integrato e non con l'algoritmo con l'uso di dizionari ontologici del capitolo 2.

6.1 Vettori di probabilità con il pacchetto `Rpart`

Riprendo le analisi del capitolo 4 e calcolo i vettori di probabilità, per poi farne la media.

```
> ##per ottenere un vettore di probabilità imposto type="prob"
> vettori <- predict(treeRpart1,newdata=test,type="prob")
> ##fare la media delle probabilità e confrontarle con i veri
> stimato1 <- colMeans(vettori)
```

Il vettore di probabilità ottenuto è rappresentato in 6.1.

	negativo	neutro	positivo	δ	χ^2
vero	*0.179*	*0.436*	*0.384*		
vettore di probabilità	0.161	0.458	0.380	-0.041	0.200
proporzioni classificazione	0.13	0.38	0.49	-0.245	4.246

Tabella 6.1: Vettore di probabilità calcolato con `Rpart`

Calcolo i vettori di probabilità considerando il risultato della classificazione effettuata con l'algoritmo basato sui dizionari ontologici e ne verifichiamo l'eventuale miglioramento nella 6.2.

```
> ##Imposto type="prob"
> vettori=predict(treeRpart2,newdata=testM,type="prob")
> stimato2<-colMeans(vettori)
> stimato2
```

	negativo	neutro	positivo	δ	χ^2
vero	*0.179*	*0.436*	*0.385*		
vettore di probabilità	0.168	0.453	0.379	-0.022	0.110
proporzioni classificazione	0.146	0.374	0.479	-0.185	3.706

Tabella 6.2: Vettore di probabilità calcolato con `Rpart` integrato con i risultati dell'algoritmo

6.2 Vettori di probabilità con il pacchetto `textir`

```
> p.vglm <-predict(m1, newdata=data.frame(z=zNew[,1]), type="response")
> ### p.vglm sono tanti vettori di probabilità, ne faccio la media
> stimato3<-colMeans(p.vglm)
```

In questo caso i vettori di probabilità e di proporzioni ottenuti sono rappresentati in 6.3.

	negativo	neutro	positivo	δ	χ^2
vero	*0.179*	*0.436*	*0.385*		
vettore di probabilità	0.163	0.459	0.378	-0.035	0.199
proporzioni classificazione	0.077	0.603	0.320	-0.287	10.887

Tabella 6.3: Vettore di probabilità calcolato con `textir`

Anche in questo caso integriamo i risultati ottenuti con l'algoritmo ontologico e procediamo con il calcolo del vettore di probabilità, indicato in tabella 6.4.

```
> p.vglm <-predict(m1, newdata=data.frame(z=zNew[,1]), type="response")
> ### faccio la media delle colonne di p.vglm
>
> stimato3<-colMeans(p.vglm)
```

	negativo	neutro	positivo	δ	χ^2
vero	*0.179*	*0.436*	*0.385*		
vettore di probabilità	0.165	0.458	0.377	-0.025	0.179
proporzioni classificazione	0.080	0.564	0.356	-0.317	6.420

Tabella 6.4: Vettore di probabilità calcolato con `textir` integrando i risultati dell'algoritmo

Quello che si nota dall'osservazione delle tabelle 6.1, 6.2, 6.3 e 6.4 è che in ogni caso lavorare con i vettori di probabilità porta a stime più accurate rispetto alle proporzioni di classificazione. Inoltre per tutti i metodi si può evidenziare che integrare la classificazione dell'algoritmo diminuisce la distanza del vettore stimato da quello vero. A questo punto vorremmo individuare quale metodo è più preciso degli altri, quindi ci prepariamo ad un confronto di tutti i metodi, compreso `ReadMe`.

6.3 Confronto e commenti

Riportiamo nella tabella 6.5 un riepilogo di tutti i vettori di probabilità ottenuti:

	neg	neutro	pos	δ	χ^2
vero	*0.179*	*0.436*	*0.384*		
algoritmo ontologico	0.176	0.189	0.634	-0.225	16.684
`Rpart`	0.161	0.459	0.38	-0.041	0.200
`Rpart` + alg. ontologico	0.168	0.453	0.379	**-0.022**	0.110
`textir`	0.163	0.459	0.378	-0.035	0.199
`textir` + alg. ontologico	0.166	0.458	0.377	-0.025	0.179
`ReadMe`	0.189	0.428	0.384	0.023	**0.032**
`ReadMe` + alg. ontologico	0.155	0.45	0.395	-0.075	0.158
proporzioni training set	0.165	0.456	0.379	-0.028	0.154

Tabella 6.5: Riepilogo vettori di probabilità e confronto indici di distanza

Dalla tabella 6.5 si può individuare il metodo che stima i vettori di probabilità con le proporzioni più vicini alle vere, ossia gli alberi di classificazione includendo i risultati dell'algoritmo di classificazione basato sui dizionari ontologici. Tale integrazione non sembra invece utile per il metodo di Hopkins

e King, che funziona molto meglio senza. Infatti se prendiamo in considerazione l'indice χ^2, questo ultimo è il metodo che produce un vettore di probabilità più vicino a quello vero. Infine notiamo che anche considerando solo le proporzioni del training set non ci allontaniamo troppo dal vero.

6.4 Vettori di probabilità cambiando le vere proporzioni nell'insieme di verifica

Nel precedente capitolo si è spiegato che il metodo di Hopkins e King si basa su una assunzione: $P_T(S|D) = P(S|D)$. Quindi si suppone che la distribuzione individuata nel training set $P_T(S|D)$ rispecchi quella reale $P(S|D)$. In via sperimentale si è allora costruito un test set dalle proporzioni totalmente differenti dal training set, per simulare la situazione inversa a quella presupposta dal metodo `ReadMe`, e che nei casi reali potrebbe presentarsi: non è inverosimile pensare ad un caso in cui le opinioni cambiano tra il campione che abbiamo etichettato (training set) e il campione su cui vogliamo stimare la classificazione (test set). Questa possibilità ci ha portati a testare le performance dei metodi nel caso in cui training e test set abbiano diverse proporzioni. Nel caso specifico si è considerato un test set che avesse lo stesso numero di classificati positivi e negativi.

	neg	neutro	pos	δ	χ^2
vero	*0.226*	*0.549*	*0.226*		
algoritmo ontologico	0.213	0.206	0.581	-0.436	51.516
`Rpart`	0.174	0.447	0.380	-0.340	6.474
`Rpart` + alg. ontologico	0.176	0.448	0.376	-0.331	6.244
`textir`	0.179	0.451	0.370	-0.316	5.786
`textir` + alg. ontologico	0.181	0.448	0.371	-0.310	5.977
`ReadMe`	0.208	0.566	0.226	**-0.036**	**0.131**
`ReadMe` + alg. ontologico	0.192	0.563	0.244	-0.104	0.249
proporzioni training set	0.155	0.427	0.419	-0.432	9.813

Tabella 6.6: Riepilogo vettori di probabilità e confronto indici di distanza con le nuove proporzioni

Dalla tabella 6.6 si può ben notare da entrambi gli indici che in questo caso il metodo che si avvicina notevolmente di più rispetto agli altri è quello di

`ReadMe`, in particolare se usato senza l'aggiunta dei risultati dell'algoritmo, nonostante non sia rispettata l'assunzione precedentemente illustrata. In generale però si nota che in questo caso un po' tutti i metodi si allontanano sensibilmente dal vero.

Capitolo 7

Esempio di applicazione ad un data-set sul tema dell'immigrazione

Proponiamo adesso un esempio applicativo ad un nuovo data-set, questa volta di post selezionati in base ad un argomento specifico.

Tratteremo quindi tweets riguardanti la percezione dell'immigrazione in Italia, rilevati nel settembre 2013 e presi dal lavoro di tesi di Bazzo, [2]. Per semplicità riportiamo schematicamente solo i risultati delle varie analisi, riassunti nelle tabelle 7.1, 7.2, 7.3, 7.4, 7.5 e 7.6.

	vero		
predict	-1	0	1
-1	122	243	16
0	71	481	39
1	34	194	29
errore di classificazione	0.486		

Tabella 7.1: Tabella di errata classificazione dell'algoritmo basato sull'uso dei dizionari ontologici

Di nuovo quindi notiamo che per quanto riguarda la classificazione, la costruzione dell'albero di classificazione è più accurata se si appoggia ai ri-

	vero		
predict	-1	0	1
-1	12	24	2
0	34	209	17
1	0	4	5
errore di classificazione	0.264		

Tabella 7.2: Matrice di confusione del campione di verifica usando un albero di classificazione

	vero		
predict	-1	0	1
-1	17	15	2
0	29	215	17
1	0	7	5
errore di classificazione	0.228		

Tabella 7.3: Matrice di confusione del campione di verifica usando un albero di classificazione, integrando il risultato ottenuto con l'algoritmo ontologico

sultati dell'algoritmo ontologico (tabella 7.3). Ciò invece non è valido per la regressione multinomiale inversa.

Per quanto riguarda i vettori di probabilità, chi mantiene più di tutti le proporzioni di positivi e negativi vicine al vero è il metodo di Hopkins e King, rimanendo però il più lontano dai veri valori del vettore delle proporzioni. La stima che più si avvicina, infatti, è quella dell'albero di classificazione integrato con i risultati dell'algoritmo ontologico. (Vedere tabella 7.6)

Vettori di probabilità cambiando le vere proporzioni nell'insieme di verifica

Come spiegato nel capitolo 6, proviamo a cambiare le proporzioni delle classi nell'insieme di verifica, e verifichiamo il comportamento degli strumenti fin qui utilizzati osservandone i risultati riportati in tabella 7.7.

Notiamo che come prima `ReadMe`, questa volta integrato con i risultati dell'algoritmo, mantiene più degli altri quasi inalterate le proporzioni di positivi e negativi, ma per quanto riguarda la distanza dai veri valori, il metodo `Rpart` sempre integrato con l'algoritmo è quello che si avvicina di più. Infine

	vero		
predict	-1	0	1
-1	6	7	2
0	40	230	21
1	0	0	1
errore di classificazione	0.228		

Tabella 7.4: Matrice di confusione del campione di verifica usando un modello di regressione multinomiale inversa

	vero		
predict	-1	0	1
-1	5	8	3
0	41	229	21
1	0	0	0
errore di classificazione	0.238		

Tabella 7.5: Matrice di confusione del campione di verifica usando un modello di regressione multinomiale inversa integrando la classificazione ottenuta con l'algoritmo basato sui dizionari ontologici

notiamo e evidenziamo che in questo particolare caso anche i risultati dell'algoritmo ontologico si avvicinano molto di più che nei precedenti casi alle vere proporzioni.

	negativo	neutro	positivo	δ	χ^2	err. di classif.
VERO	*0.150*	*0.772*	*0.078*			
algoritmo ontologico	0.270	0.534	0.195	-0.143	14.416	0,486
`Rpart`	0.198	0.750	0.053	0.288	0.236	0.264
`Rpart` + alg. ontologico	0.163	0.775	0.062	0.136	**0.016**	0.228
`textir`	0.185	0.749	0.065	0.170	0.186	0.228
`textir` + alg. ontologico	0.183	0.753	0.064	0.172	0.140	0.238
`ReadMe`	0.260	0.633	0.107	**0.102**	5.156	-
`ReadMe` + alg. ontologico	0.150	0.732	0.118	-0.180	0.418	-
proporzioni training set	0.196	0.739	0.065	0.195	0.360	-

Tabella 7.6: Tabella di riepilogo con i vettori di probabilità stimati e gli errori di classificazione

	negativo	neutro	positivo	δ	χ^2
VERO	*0.084*	*0.832*	*0.084*		
algoritmo ontologico	0.256	0.537	0.207	0.092	21.706
`Rpart`	0.190	0.746	0.064	0.473	2.032
`Rpart` + alg. ontologico	0.180	0.758	0.062	0.463	**1.531**
`textir`	0.199	0.735	0.066	0.479	2.555
`textir` + alg. ontologico	0.192	0.741	0.066	0.464	2.251
`ReadMe`	0.260	0.633	0.107	0.358	11.239
`ReadMe` + alg. ontologico	0.150	0.732	0.118	**0.003**	1.760
proporzioni training set	0.215	0.721	0.064	0.526	3.342

Tabella 7.7: Nuove vere proporzioni: Tabella di riepilogo con i vettori di probabilità stimati e gli errori di classificazione

Conclusioni

Riepilogando i risultati ottenuti nei vari capitoli di questo lavoro si giunge alla conclusione che l'algoritmo di classificazione basato sui dizionari ontologici non è uno strumento affidabile usato singolarmente. Di contro se questo viene integrato con gli altri metodi, nella maggior parte dei casi ne migliora i risultati, portando a stime (sia puntuali che di probabilità) più vicine al vero. In particolare si nota che il metodo di Hopkins e King, che si basa sulla classificazione aggregata e quindi produce vettori di probabilità, non sempre è quello che funziona meglio, portandoci quindi a preferire spesso gli alberi di classificazione, che ci facilitano compiti come la stratificazione del campione (ad esempio per genere, classi di età o tipo di utente) senza dover riclassificare i dati per ogni strato, inoltre risultano maggiormente comprensibili e si distinguono per la rapidità di calcolo.

Gli alberi di classificazione, quando integrano i risultati dell'algoritmo ontologico, portano a risultati soddisfacenti anche nella classificazione individuale dei singoli tweet, portandoci a pensare che un giusto investimento nei futuri lavori potrebbe essere quello del perfezionamento dell'algoritmo stesso, aggiungendo l'analisi dei bigrammi e trigrammi, che consentirebbero l'interpretazione (e la conseguente assegnazione di polarità) di frasi e modi di dire più complessi, e l'analisi delle negazioni; rendendo i dizionari realizzati durante questo lavoro più accurati e precisi oppure creando dizionari alternativi specifici dell'argomento che si va ad analizzare; perfezionando lo stemming così come trovando un modo computazionalmente poco oneroso di analizzare gli hashtag.

Se invece si cambiano sensibilmente le proporzioni del test set si nota che nessun metodo è particolarmente affidabile. L'ipotesi che la distribuzione

individuata nel training set rispecchi quella reale potrebbe essere violata, ad esempio in tutti i casi reali in cui le opinioni cambiano tra il campione che abbiamo etichettato e il campione su cui vogliamo stimare la classificazione. Nel nostro caso si è proceduto considerando un test set che pareggiasse il numero di classificati positivi e negativi, ma come già detto i vettori di probabilità stimati con i vari metodi sono risultati distanti dal vero, a parte qualche eccezione.

Riferimenti bibliografici

[2] Enrico Bazzo. ■*Un'applicazione del metodo di Hopkins-King per la sentiment analysis*■. Relatore: Prof. Ruggero Bellio. Tesi di laurea triennale. Università degli studi di Udine, Anno Accademico 2012/13.

[3] Andrea Ceron, Luigi Curini e Stefano M. Iacus. *Social Media e Sentiment Analysis, L'evoluzione dei fenomeni sociali attraverso la Rete*. 2013.

[4] R.D. Cook. ■*Fisher Lecture: Dimension Reduction in Regression*■. In: *Statistical Science* (2007).

[6] Daniel J. Hopkins e Gary King. ■*A Method of Automated Nonparametric Content Analysis for Social Science*■. In: *American Journal of Political Science* (January 2010).

[7] Melucci Massimo. *Information Retrieval, Metodi e modelli per i motori di ricerca*. 2013.

[12] Matt Taddy. ■*Measuring political sentiment on Twitter: Factor-optimal design for multinomial inverse regression.*■ In: *Technometrics 55* (2013).

[13] Matt Taddy. ■*Multinomial Inverse Regression for Text Analysis*■. In: *Journal of the American Statistical Association* (2013).

Siti Web consultati

[1] *An italian stop word list*. URL: http://snowball.tartarus.org/algorithms/italian/stop.txt.

[5] Antonino Freno. *Tecniche di Stemming*. URL: www.dii.unisi.it/~freno/files/didattica/slides_stemming.pdf.

[8] *pacchetto R SnowballC*. (2013). URL: http://cran.r-project.org/web/packages/SnowballC/index.html.

[9] *Porter Stemmer*. URL: http://tartarus.org/~martin/PorterStemmer/.

[10] Matteo Redaelli. *dump_tweets.R*. (2014). URL: https://github.com/matteoredaelli/dump_tweets.R.

[11] Dario Solari et al. *TextWiller: Collection of functions for text mining, specially devoted to the italian language*. URL: https://github.com/livioivil/TextWiller.